QAnywhere™

2009 年 2 月

11.0.1 版

版权和商标

目录

关于本手册

主题

本手册介绍 QAnywhere，一个用于移动、无线、台式机和膝上型客户端的消息传递平台。

目标读者

本手册面向的读者是 SQL Anywhere 和其它关系数据库系统的用户，他们希望为其移动应用程序添加消息传递功能，或者希望建立新的移动应用程序到应用程序消息传递解决方案。

关于 SQL Anywhere 文档

完整的 SQL Anywhere 文档以四种形式提供，但所包含信息均相同。

- **HTML 帮助** 联机帮助文档包含完整的 SQL Anywhere 文档，其中包括手册和 SQL Anywhere 工具的上下文相关帮助。

  如果使用 Microsoft Windows 操作系统，则联机帮助文档以 HTML 帮助 (CHM) 格式提供。若要访问此文档，请选择 [开始] » [程序] » [**SQL Anywhere 11**] » [文档] » [联机手册]。

  管理工具使用同一联机文档来实现帮助功能。

- **Eclipse** 在 Unix 平台上以 Eclipse 格式提供完整的联机帮助。要访问文档，请从 SQL Anywhere 11 安装的 *bin32* 或 *bin64* 目录下运行 *sadoc*。

- **DocCommentXchange** DocCommentXchange 是一个用于访问和讨论 SQL Anywhere 文档的社区。

  使用 DocCommentXchange 可以执行以下任务：

  - 查看文档
  - 检查是否有用户对文档各部分所做出的阐明
  - 提供建议和修正意见以在将来的版本中为所有用户改进文档

  访问 http://dcx.sybase.com。

- **PDF** 整套 SQL Anywhere 手册会以一组 Portable Document Format (PDF) 文件的形式提供。您必须有 PDF 阅读器才能查看信息。要下载 Adobe Reader，请访问 http://get.adobe.com/reader/。

  若要在 Microsoft Windows 操作系统上访问 PDF 文档，请选择 [开始] » [程序] » [**SQL Anywhere 11**] » 文档 » [联机手册 - PDF 格式]。

  要在 Unix 操作系统上访问 PDF 文档，请使用 Web 浏览器打开 *install-dir/documentation/zh/pdf/index.html*。

关于文档集中的手册

SQL Anywhere 文档由以下手册组成：

- **SQL Anywhere 11 - 简介** 本手册介绍 SQL Anywhere 11，一个提供数据管理和数据交换技术的综合数据包，通过它可以为服务器环境、台式机环境、移动环境以及远程办公环境快速开发由数据库驱动的应用程序。

- **SQL Anywhere 11 - 更改和升级** 本手册介绍 SQL Anywhere 11 以及该软件以前版本中的新功能。

- **SQL Anywhere 服务器 - 数据库管理** 本手册介绍如何运行、管理及配置 SQL Anywhere 数据库。它介绍了数据库连接、数据库服务器、数据库文件、备份过程、安全性、高可用性、使用复制服务器进行复制以及管理实用程序和选项。

- **SQL Anywhere 服务器 - 编程** 本手册介绍如何使用 C、C++、Java、PHP、Perl、Python 和 .NET 编程语言（例如 Visual Basic 和 Visual C#）建立和部署数据库应用程序。其中介绍了各种编程接口，如 ADO.NET 和 ODBC。
- **SQL Anywhere 服务器 - SQL 参考** 本手册提供了系统过程和目录（系统表和视图）的参考信息。也介绍了 SQL 语言（搜索条件、语法、数据类型和函数）的 SQL Anywhere 实现。
- **SQL Anywhere 服务器 - SQL 的用法** 本手册介绍如何设计和创建数据库；如何导入、导出和修改数据；如何检索数据以及如何建立存储过程和触发器。
- **MobiLink - 入门** 本手册介绍基于会话的关系数据库同步系统 MobiLink。MobiLink 技术支持双向复制并且非常适用于移动计算环境。
- **MobiLink - 客户端管理** 本手册介绍如何设置、配置和同步 MobiLink 客户端。MobiLink 客户端可以是 SQL Anywhere 或者 UltraLite 数据库。本手册同时也介绍了 Dbmlsync API，通过它可无缝地将同步集成到 C++ 或 .NET 客户端应用程序中。
- **MobiLink - 服务器管理** 本手册说明如何设置和管理 MobiLink 应用程序。
- **MobiLink - 服务器启动的同步** 本手册介绍 MobiLink 服务器启动的同步，这种功能允许 MobiLink 服务器启动同步或在远程设备上进行操作。
- **QAnywhere** 本手册介绍 QAnywhere，一个用于移动、无线、台式机和膝上型客户端的消息传递平台。
- **SQL Remote** 本手册介绍用于移动计算的 SQL Remote 数据复制系统，此系统支持使用电子邮件或文件传输等间接链接共享 SQL Anywhere 统一数据库和多个 SQL Anywhere 远程数据库之间的数据。
- **UltraLite - 数据库管理和参考** 本手册介绍适用于小型设备的 UltraLite 数据库系统。
- **UltraLite - C 及 C++ 编程** 本手册介绍 UltraLite C 和 C++ 编程接口。利用 UltraLite，可以开发数据库应用程序，并将它们部署到手持式设备、移动设备或嵌入式设备。
- **UltraLite - M-Business Anywhere 编程** 本手册介绍 UltraLite for M-Business Anywhere。利用 UltraLite for M-Business Anywhere，用户可以开发基于 Web 的数据库应用程序，并将它们部署到运行 Palm OS、Windows Mobile 或 Windows 的手持式设备、移动设备或嵌入式设备。
- **UltraLite - .NET 编程** 本手册介绍 UltraLite.NET。利用 UltraLite.NET，您可以开发数据库应用程序，并将它们部署到计算机、手持式设备、移动设备或嵌入式设备。
- **UltraLiteJ** 本手册介绍 UltraLiteJ。利用 UltraLiteJ，可以在支持 Java 的环境中开发和部署数据库应用程序。UltraLiteJ 支持 BlackBerry 智能手机和 Java SE 环境。UltraLiteJ 基于 iAnywhere UltraLite 数据库产品。
- **错误消息** 本手册提供了 SQL Anywhere 错误消息及其诊断信息的完整列表。

文档约定

本节列出了本文档中使用的约定。

操作系统

SQL Anywhere 可以在各种平台上运行。在大多数情况下，该软件在所有平台上的行为都是相同的，但也有变动或限制。这些变动或限制通常基于基础操作系统（Windows、Unix），很少基于特定变型（AIX、Windows Mobile）或版本。

为了简化对操作系统的提及，本文档按如下方式对支持的操作系统进行分组：

- **Windows** Microsoft Windows 系列包括 Windows Vista 和 Windows XP（主要用于服务器、台式计算机和膝上型计算机），以及 Windows Mobile（用于移动设备）。

  除非另外指定，否则当本文档提及 Windows 时，是指所有基于 Windows 的平台，包括 Windows Mobile。

- **Unix** 除非另外指定，否则当本文档提及 Unix 时，是指所有基于 Unix 的平台，包括 Linux 和 Mac OS X。

目录和文件名

大部分情况下，对目录和文件名的引用在所有支持的平台上都是类似的，只需在不同形式之间进行简单的转换。这时需使用 Windows 约定。在细节更为复杂的情况下，文档显示所有相关形式。

下面是文档编写中用于简化目录和文件名的约定：

- **大写和小写目录名** 在 Windows 和 Unix 上，目录和文件名可以包括大写和小写字母。创建目录和文件时，文件系统会保留字母大小写。

  在 Windows 上，对目录和文件的提及*不区分大小写*。混合使用大小写的目录和文件名很常见，但使用所有小写字母来提及目录和文件的形式也很常见。SQL Anywhere 安装包含诸如 *Bin32* 和 *Documentation* 的目录。

  在 Unix 上，对目录和文件的提及*区分大小写*。混合使用大小写的目录和文件名不常见。大多数的目录和文件名全部使用小写字母。SQL Anywhere 安装包含诸如 *bin32* 和 *documentation* 的目录。

  本文档采用 Windows 形式的目录名。大多数情况下，在 Unix 上可以将大小写混合形式的目录名转换成小写字母的等效目录名。

- **分隔目录和文件名的斜线** 文档使用反斜线作为目录分隔符。例如，PDF 格式的文档位于 *install-dir\Documentation\zh\PDF*（Windows 形式）。

  在 Unix 上，用正斜线替换反斜线。PDF 文档位于 *install-dir/documentation/zh/pdf* 下。

- **可执行文件** 文档使用 Windows 约定显示可执行文件名（带有诸如 *.exe* 或 *.bat* 后缀）。在 Unix 上，可执行文件名没有后缀。

  例如，在 Windows 上，网络数据库服务器是 *dbsrv11.exe*。在 Unix 上是 *dbsrv11*。

- ***install-dir*** 在安装过程中，选择 SQL Anywhere 的安装位置。创建环境变量 SQLANY11，用来表示此位置。文档中以 *install-dir* 表示此位置。

  例如，本文档将此文件表示为 *install-dir*\readme.txt。在 Windows 上，这等同于 *%SQLANY11%\readme.txt*。在 Unix 上，这等同于 *$SQLANY11/readme.txt* 或 *${SQLANY11}/readme.txt*。

有关 *install-dir* 缺省位置的详细信息，请参见“SQLANY11 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

- ***samples-dir*** 在安装过程中，选择 SQL Anywhere 随附的示例的安装位置。创建环境变量 SQLANYSAMP11，用来表示此位置。文档中以 *samples-dir* 表示此位置。

  要在 *samples-dir* 中打开 Windows 资源管理器窗口，请在 [**开始**] 菜单中，选择 [**程序**] » [**SQL Anywhere 11**] » [**示例应用程序和项目**]。

  有关 *samples-dir* 缺省位置的详细信息，请参见“SQLANYSAMP11 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

命令提示符和命令 shell 语法

大多数操作系统都提供一种或多种使用命令 shell 或命令提示符来输入命令和参数的方法。Windows 命令提示符包括 Command Prompt（DOS 提示符）和 4NT。Unix 命令 shell 包括 Korn shell 和 bash。每个 shell 都具有一些功能，其能力不仅仅局限于简单命令。这些功能通过特殊字符来驱动。特殊字符和功能随 shell 的不同而不同。如果没有正确使用这些特殊字符，通常会导致语法错误或意外行为。

本文档以普通形式提供命令行示例。如果这些示例中包含 shell 的特殊字符，则命令需要根据特定 shell 进行修改。修改方法不在本文档所述范围之内，但通常是在包含这些特殊字符的参数两旁加上引号，或是在特殊字符前面使用转义字符。

下面是命令行语法的一些示例，不同的平台可能会有不同的形式：

- **括号和大括号** 有些命令行选项需要一个参数，该参数将以列表形式接受详细的值指定。该列表通常用括号或大括号括起来。本文档使用括号。例如：

  ```
  -x tcpip(host=127.0.0.1)
  ```

  如果括号导致出现语法问题，用大括号替代：

  ```
  -x tcpip{host=127.0.0.1}
  ```

  如果两种形式都将产生语法问题，应按照 shell 的要求，用引号将整个参数括起来：

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **引号** 如果必须在参数值中指定引号，该引号可能会与用于括参数的引号的传统用法发生冲突。例如，要指定值中包含双引号的加密密钥，则可能必须用引号括起密钥，然后转义嵌入的引号：

  ```
  -ek "my \"secret\" key"
  ```

  在许多 shell 中，密钥的值为 `my "secret" key`。

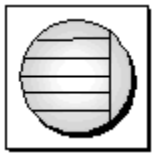
- **环境变量** 本文档介绍设置环境变量。在 Windows shell 中，环境变量使用语法 *%ENVVAR%* 来指定。在 Unix shell 中，环境变量使用语法 *$ENVVAR* 或 *${ENVVAR}* 来指定。

图标

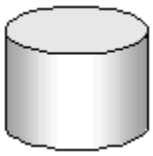
本文档中使用了下列图标。

- 客户端应用程序。

- 数据库服务器，如 Sybase SQL Anywhere。



- 数据库。在某些高水平的图中，可以使用此图标表示数据库和管理该数据库的数据库服务器。



- 复制或同步中间件。用于帮助在数据库之间共享数据。例如 MobiLink 服务器和 SQL Remote 消息代理。

- 编程接口。

联系文档小组

我们欢迎您就本帮助文档提出意见、建议和反馈信息。

要提交意见和建议，请发送电子邮件到 SQL Anywhere 文档小组，地址为 iasdoc@sybase.com。虽然我们不对这些电子邮件进行回复，但您的反馈会帮助我们改进文档，因此我们真诚地欢迎您提出宝贵的意见和建议。

DocCommentXchange

也可以使用 DocCommentXchange 将意见或建议直接置于帮助主题中。DocCommentXchange (DCX) 是一个用于访问和讨论 SQL Anywhere 文档的社区。使用 DocCommentXchange 可以执行以下任务：

- 查看文档
- 检查是否有用户对文档各部分所做出的阐明
- 提供建议和修正意见以在将来的版本中为所有用户改进文档

访问 http://dcx.sybase.com。

查找详细信息并请求技术支持

附加信息和资源可从 Sybase iAnywhere 开发人员社区获得，网址是 http://www.sybase.com/developer/library/sql-anywhere-techcorner。

如果您有问题或是需要帮助，可将邮件发布到下面所列的 Sybase iAnywhere 新闻组。

当您向这些新闻组发布邮件时，请务必提供问题的详细信息，包括 SQL Anywhere 版本的内部版本号。可以通过运行以下命令找到此信息：**dbeng11 -v**。 **dbeng11 -v**.

新闻组位于 *forums.sybase.com* 新闻服务器上。

这些新闻组包括：

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

有关 Web 开发问题，请访问 http://groups.google.com/group/sql-anywhere-web-development。

新闻组免责声明

iAnywhere Solutions 没有义务为其新闻组提供解决方案、信息或建议，除提供系统操作员监控服务和确保新闻组的运行和可用性外，iAnywhere Solutions 也没有义务提供任何其它服务。

如果时间允许，iAnywhere 技术顾问以及其他员工也会对新闻组服务提供帮助。他们是在自愿的基础上提供帮助的，所以可能无法定期提供解决方案和信息。他们可以提供多少帮助取决于他们的工作量。

QAnywhere 技术简介

目录

QAnywhere 应用程序到应用程序消息传递

QAnywhere 帮助您开发针对移动设备的应用程序到应用程序的消息传递。应用程序到应用程序消息传递允许在移动或无线设备上运行的自定义程序和位居中央的服务器应用程序之间进行通信。

QAnywhere 使用存储并转发技术为远程应用程序和移动应用程序提供安全而保险的消息传送。因为 QAnywhere 能够自动处理网络速度慢和不可靠的问题，所以您可将精力集中在应用程序功能（而非有关连接、通信和安全的问题）之上。QAnywhere 存储并转发技术确保即使在网络连接不可用时，您的应用程序也始终可用。

QAnywhere 可实现在不定时连接环境中进行通信。它解决了无线网络的一些难题（如速度低、地理位置覆盖不平衡和网络连接易断等问题）。QAnywhere 消息传递的存储并转发性质意味着即使无法通过网络到达目标应用程序也可构造消息；网络可用时再传递消息。

QAnywhere 消息通过一个中心服务器交换，因此不需要消息发送者和接收者同时连接到网络。

QAnywhere 具有以下附加功能：

- QAnywhere 可以通过对公共网络上发送的所有消息进行加密来保护专有或敏感信息。
- 可以使用传输规则自定义消息的传递，例如可以在非高峰的时候传输大量低优先级的消息。
- QAnywhere 消息可通过 TCP/IP、HTTP 或 HTTPS 协议传输。还可以通过 ActiveSync 从 Windows Mobile 手持式设备传送。消息本身独立于网络协议，并且可由通过不同网络通信的应用程序接收。
- QAnywhere 压缩在移动应用程序和企业服务器之间发送的数据。
- QAnywhere 提供 C++、Java、.NET 和 SQL API，为具有不同技能的开发人员提供了解决方案。
- QAnywhere 允许与其它具有 JMS 接口的消息传递系统进行无缝通信。这样就可进行与 J2EE 应用程序的集成。
- QAnywhere 包括一个移动 Web 服务接口，该接口可帮助您基于企业 Web 服务来创建可靠的移动应用程序。

QAnywhere 基于 MobiLink 同步技术而构建。

QAnywhere 的作用是什么

QAnywhere 提供下面的应用程序到应用程序功能和组件。

- **QAnywhere API** 面向对象的 QAnywhere API 为构建用于 Windows 桌面操作系统和 Windows Mobile 设备的消息传递应用程序提供了基础结构。QAnywhere API 可用于 Java、C++、.NET 和 SQL。
- **存储并转发** 在客户端和服务器之间的连接可用于传输数据前，QAnywhere 应用程序将消息进行本地存储。
- **补充数据同步** QAnywhere 应用程序将关系数据库用作临时消息存储库。关系数据库确保消息存储库具有关系数据库的安全性、基于事务的计算和其它优点。

  将 SQL Anywhere 关系数据库用作消息存储库可使将 QAnywhere 与数据同步解决方案一起使用变得容易。这两者都将 MobiLink 同步用作在客户端和服务器之间交换信息的基础机制。
- **与外部消息系统的集成** 除在 QAnywhere 应用程序之间交换消息外，还可以将 QAnywhere 客户端集成到支持 JMS 接口的外部消息系统中。
- **加密** 通过传送层安全性可将消息加密发送。此外，可使用简单加密或任何经过 FIPS 认可的 AES 算法加密消息存储库。
- **压缩** 使用常用的 ZLIB 压缩库，可将消息内容进行压缩存储。
- **验证** 可通过内置工具或通过自定义的验证脚本（包括在您组织中使用的现有验证服务）验证 QAnywhere 客户端。
- **多个网络** 在任何支持 TCP/IP 或 HTTP 的有线或无线网络上，QAnywhere 都可以运行。
- **故障转移** 可以运行多个 MobiLink 服务器，以便其中一个服务器发生故障时可使用备用服务器。
- **管理** QAnywhere 应用程序可在客户端和服务器端浏览及操作消息。
- **多个队列** 在客户端设备上支持多个任意命名的队列允许在单个设备上共存多个客户端应用程序。应用程序可以在任意数目的队列上发送和接收。消息可以在共存于同一设备的应用程序之间传送，也可以在不同设备上的应用程序之间传送。
- **服务器启动的发送和接收** QAnywhere 可以向客户端设备推送消息，允许客户端应用程序实现消息驱动的逻辑。
- **传输规则** 可以创建规则指定何时发生消息传输。
- **断点续传下载** 较大的消息或消息组以分段形式传送到 QAnywhere 客户端，以使网络故障期间的数据重新传输减到最小。
- **有保证的收发** QAnywhere 可保证消息的传送。
- **移动 Web 服务** 移动 Web 服务有助于通过 QAnywhere 来传输 Web 服务请求和响应。

QAnywhere 体系结构

本节介绍 QAnywhere 消息传递应用程序的体系结构。讨论从简单的消息传递方案开始，然后再讨论更高级的方案。

客户端应用程序使用 QAnywhere API 发送和接收消息。消息在客户端消息存储库中排队。消息传输是通过中央 QAnywhere 服务器消息存储库、在客户端消息存储库之间进行的消息交换。

下面是 QAnywhere 支持的典型消息传递方案：

- **简单消息传递** 用于在 QAnywhere 客户端之间交换消息。客户端应用程序控制何时在客户端和服务器消息存储库之间传输消息。

  请参见“简单消息传递方案”一节第 4 页。

- **使用推式通知进行消息传递** 用于在 QAnywhere 客户端之间交换消息。在本方案中，MobiLink 服务器可以启动客户端之间的消息传输。这通过在客户端和服务器消息存储库之间交换消息实现。

  请参见“使用推式通知进行消息传递的方案”一节第 6 页。

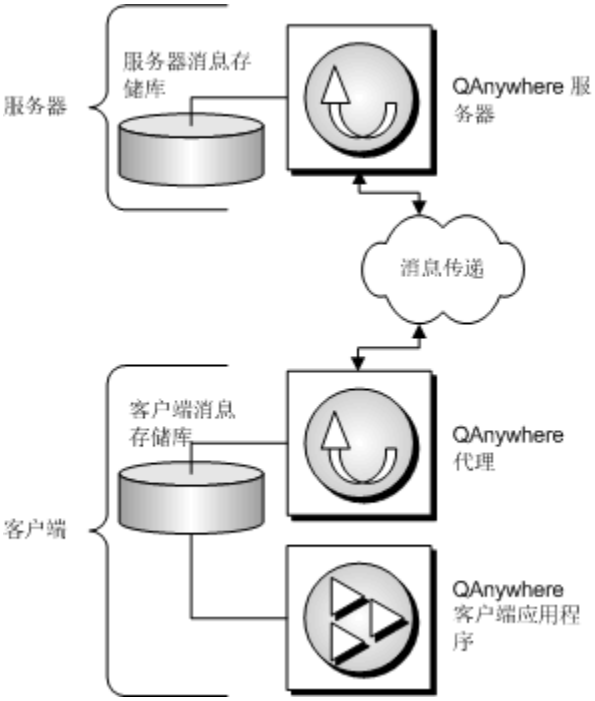
- **使用外部消息系统进行消息传递** 用于在提供 JMS 提供程序的外部系统（如 BEA WebLogic 或 Sybase EAServer）上的 QAnywhere 客户端之间交换消息。

  请参见“使用外部消息系统进行消息传递的方案”一节第 7 页。

可以同时使用推式通知和外部消息系统来提供最通用的解决方案。

简单消息传递方案

下图说明了一个简单 QAnywhere 消息传递设置。简单起见，只显示一个客户端。但是，典型方案中存在多个客户端，服务器消息存储库在这些客户端之间传输消息。



此设置包含以下组件：

- **服务器消息存储库** 在服务器上，消息存储在关系数据库中。数据库必须设置为 MobiLink 统一数据库，并且可以是任何受支持的统一数据库。
- **客户端消息存储库** 每个客户端上的消息都存储在一个关系数据库中。QAnywhere 支持 SQL Anywhere 和 UltraLite 数据库。对于数据同步应用程序，建议使用 SQL Anywhere 数据库。对于专用于存储并转发消息的应用程序，建议使用 UltraLite 数据库。
- **QAnywhere 服务器** QAnywhere 服务器是为消息传递而启用的 MobiLink 服务器。MobiLink 同步提供在 QAnywhere 客户端和服务器之间传输和跟踪消息的传输功能。MobiLink 提供安全性、验证、加密和灵活性。还允许组合消息传递与数据同步。

  要启动 QAnywhere 服务器，使用 -m 选项启动 MobiLink 服务器。请参见“启动启用了 MobiLink 的 QAnywhere”一节第 30 页。
- **QAnywhere 代理** QAnywhere 代理管理客户端的消息传输。此过程与 QAnywhere 客户端应用程序无关。

  请参见“启动 QAnywhere 代理”一节第 48 页。
- **QAnywhere 客户端应用程序** 使用 QAnywhere C++、Java 或 .NET API 编写的应用程序调用方法发送和接收消息。客户端应用程序使用的基本对象是 QAManager。

  请参见“编写 QAnywhere 客户端应用程序”第 53 页。

消息由 QAnywhere 客户端发送和接收。客户端启动消息传输后才会拾取服务器上的消息。QAnywhere 客户端使用**策略**确定何时执行消息传输。策略分为 [要求时]、[自动]、[调度] 和 [自定义] 四种。[要

求时] 策略允许用户控制消息传输。每当在客户端上有消息准备收发时，[自动] 策略便会启动消息传输。[自定义] 策略使用各种传输规则对消息传输进行更多的控制。

请参见“确定在客户端进行消息传输的时间”一节第 51 页。

使用推式通知进行消息传递的方案

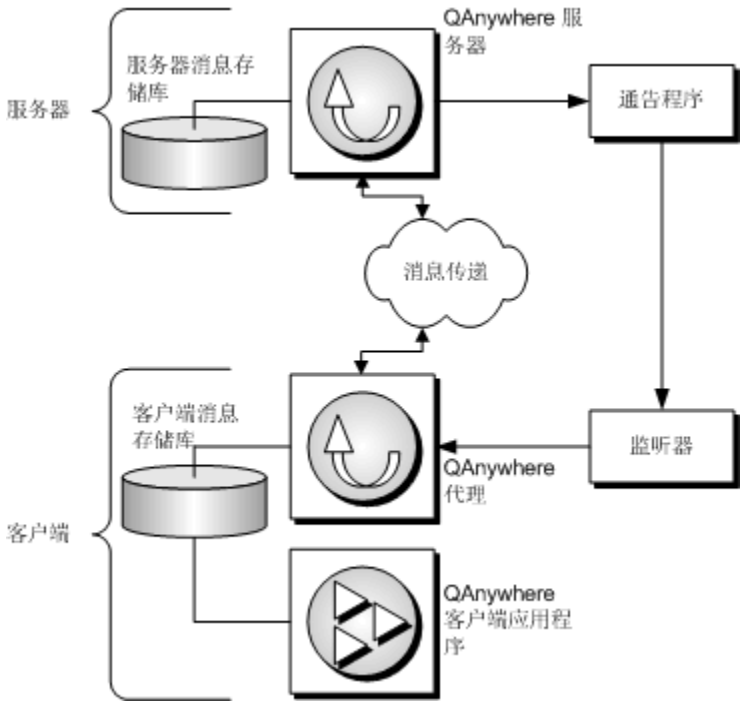
推式通知是一种从服务器传送到 QAnywhere 客户端的特殊消息。当某个消息到达服务器消息存储库时，将出现推式通知。消息传递服务器自动通知推式请求的接收者客户端监听器。客户端启动消息传输以接收在服务器等待的消息，或者执行自定义操作。

有关客户端对推式通知的响应的详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

推式通知向 QAnywhere 体系结构引入了两个额外组件。在服务器端，QAnywhere 通告程序发送推式通知。在客户端，QAnywhere 监听器接收这些推式通知，并将这些通知传递给 QAnywhere 代理。

如果不使用推式通知，消息还是可以从服务器消息存储库传输到客户端消息存储库，但必须在客户端启动传输（如通过使用调度传输策略）。

使用推式通知进行消息传递的体系结构是“简单消息传递方案”一节第 4 页介绍的体系结构的扩展。下图显示了该体系结构：



以下组件添加到简单消息传递方案中以启用推式通知：

- **QAnywhere 通告程序** 通告程序是 MobiLink 服务器的一个组件，用于传送推式通知。

QAnywhere 通告程序是通告程序（有消息准备收发时发送推式通知）的特殊配置的实例。

- **监听器** 监听器是在客户端运行的独立进程。它接收推式通知并把它们传送到 QAnywhere 代理。QAnywhere 代理策略确定推式通知是否自动触发消息传输。

  请参见“确定在客户端进行消息传输的时间”一节第 51 页。

另请参见

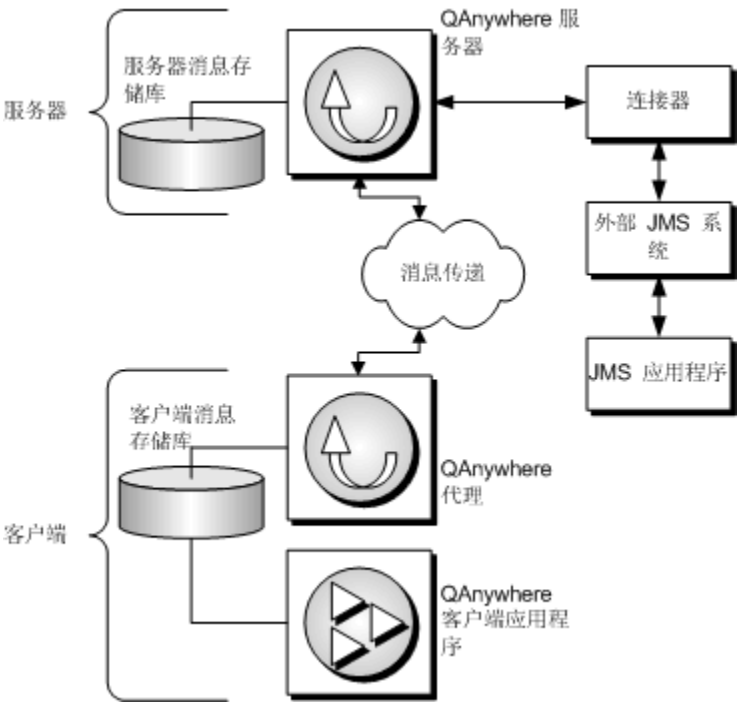
- “使用推式通知”一节第 34 页
- “异步接收消息”一节第 75 页
- “服务器启动的同步简介” 《MobiLink - 服务器启动的同步》

使用外部消息系统进行消息传递的方案

除在 QAnywhere 应用程序之间交换消息外，还可以使用特殊配置的客户端（称为连接器）与具有 JMS 接口的系统交换消息。JMS 是用于向 Java 应用程序添加消息传递能力的 Java 消息服务 API。

外部消息传递系统被设置为特殊客户端。它有自己的地址和配置。

使用外部消息系统进行消息传递的体系结构是“简单消息传递方案”一节第 4 页介绍的体系结构的扩展。下图显示了该体系结构：



为能够使用外部消息传递系统进行消息传递而添加到简单消息传递方案中的组件如下：

- **QAnywhere JMS 连接器** JMS 连接器在 QAnywhere 和外部消息系统之间提供了一个接口。

JMS 连接器是在 QAnywhere 和外部 JMS 系统之间移动消息的特殊 QAnywhere 客户端。

另请参见

- “连接器”第 153 页
- “教程：使用 JMS 连接器”一节第 166 页

QAnywhere 消息传送

消息从某个客户端消息存储库发送到服务器消息存储库，然后再转到另一个客户端消息存储库。QAnywhere 通过队列实现该操作：将某个消息添加到客户端消息存储库的队列中；服务器消息存储库接收该消息后，将其添加到用于传送到一个或多个客户端消息存储库的队列中；某个客户端消息存储库接收到该消息后，将其添加到队列中以进行拾取。

消息一经发送，除非发生以下情况之一，否则都将进行传送：

- 消息已到期（仅适用于指定到期时间的情况）。
- 已通过 Sybase Central 或 cancelMessage API 调用取消该消息。
- 发出该消息的设备丢失并且在该设备可与服务器消息存储库同步之前不能找回（或者，因其它原因无法进行同步）。

消息仅传送一次。如果应用程序成功地确认或表示收到某个消息，则不再传送同一消息。对于 JMS 服务器来说，可能发生一种例外情况：如果 MobiLink 服务器或 JMS 服务器发生崩溃，则有可能将某个消息传送两次。

SQL Anywhere 和 UltraLite 之间的选择

QAnywhere 客户端应用程序现在可以将 UltraLite 数据库用作客户端消息存储库。这为移动设备上的纯消息传递应用程序提供了更轻量化的解决方案。对于纯消息传递应用程序，我们是指具有存储并转发消息传递功能，但不进行数据同步的应用程序。

UltraLite 的一些主要优点：

- 它的应用程序占用资源空间更少且不要求完整的 SQL Anywhere 安装。
- 它的进程占用资源空间更少。QAnywhere 代理仅需要 3 个进程（uleng11、dblsn 和 qauagent），而不是 4 个（dbeng11、dbmlsync、dblsn 和 qaagent）。

UltraLite 限制

在 SQL Anywhere 和 UltraLite 之间进行选择时，请记住 UltraLite 具有以下限制：

- 传输规则条件语法中没有对 "ESCAPE" 关键字的支持。
- 对属性特性的支持有限。只能将属性特性功能与预定义属性 ias_Network 配合使用。请参见“自定义客户端消息存储库属性”一节第 747 页。

建议

通常，在所有情况下，如果 SQL Anywhere 尚未实现，则使用 UltraLite，而不是 SQL Anywhere。如果想要在已实现的 SQL Anywhere 数据同步解决方案中添加消息传递功能，则可以使用 SQL Anywhere。但是，在所有纯消息传递环境中，建议使用 UltraLite。

QAnywhere 插件

Sybase Central QAnywhere 插件可帮助您创建和管理 QAnywhere 应用程序。使用该插件可执行以下操作：

- 创建客户端和服务器消息存储库。
- 为 QAnywhere 代理创建和维护配置文件。
- 浏览 QAnywhere 代理日志文件。
- 创建或修改目标别名。
- 创建 JMS 连接器和 Web 服务连接器。
- 创建和维护传输规则文件。
- 远程浏览消息存储库。
- 跟踪消息。

♦ 启动 QAnywhere 插件

1. 启动 Sybase Central：

   选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 选择 [**连接**] » [**使用 QAnywhere 11 连接**]。
3. 指定一个 [**ODBC 数据源名称**] 或 [**ODBC 数据源文件**]、[**用户 ID**] 和 [**口令**]（如果需要）。
4. 单击 [**OK**]。

QAnywhere 快速入门

以下步骤概述了设置和运行 QAnywhere 消息传递所需执行的任务。

♦ 设置并运行 QAnywhere 消息传递

1. 设置服务器消息存储库或使用现有的 MobiLink 统一数据库。

   请参见“设置服务器消息存储库”一节第 21 页。
2. 使用 -m 选项和到服务器消息存储库的连接启动 MobiLink 服务器。

   请参见“启动启用了 MobiLink 的 QAnywhere”一节第 30 页。
3. 设置客户端消息存储库。这些是用来临时存储消息的 SQL Anywhere 或 UltraLite 数据库。

   请参见“设置客户端消息存储库”一节第 23 页。
4. 为每个客户端编写一个消息传递应用程序。

   请参见“编写 QAnywhere 客户端应用程序”第 53 页。
5. 如果希望与外部 JMS 消息系统集成，则为 QAnywhere 设置 JMS 消息传递。

   请参见“连接器”第 153 页。
6. 使用到本地客户端消息存储库的连接对每个客户端启动 QAnywhere 代理。

   请参见“启动 QAnywhere 代理”一节第 48 页。

有关设置移动 Web 服务的信息，请参见“移动 Web 服务”第 101 页。

其它入门资源

- “教程：探讨 TestMessage”第 197 页
- “教程：使用 JMS 连接器”一节第 166 页
- 示例应用程序安装在 *samples-dir\QAnywhere* 中。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）
- 可在 QAnywhere 新闻组上发布问题：ianywhere.public.sqlanywhere.qanywhere

消息

目录

QAnywhere 消息简介

QAnywhere 消息由以下部分组成：

- 标头
- 属性
- content

消息属性可以在传输规则、删除规则或应用程序中引用。

以下各节将介绍消息标头和属性以及在 QAnywhere 消息中设置它们的方法。

注意

- 发送消息后，不能变更消息标头、消息属性和消息内容。
- 收到消息后，可以读取消息标头、消息属性和消息内容。如果使用的是 QAnywhere SQL API，则在提交或回退后这些内容将不可读。
- 在所有 API 中确认或提交后，内容将不可读。

消息标头

所有 QAnywhere 消息都支持相同的标头字段集。标头字段包含客户端和提供程序用来标识和路由消息的值。标头的使用方法取决于您所拥有的客户端应用程序的类型。

QAnywhere 支持以下预定义的消息标头：

- 消息 ID
- 消息创建时间戳
- 回复地址
- 消息地址
- 消息的重新传送状态
- 消息的到期时间
- 消息的优先级
- 此消息所回复的消息的消息 ID

有关消息标头的详细信息，请参见“消息标头”一节第 684 页。

消息属性

每条消息都包含用于支持应用程序定义的属性值的内置工具。这些消息属性允许您执行应用程序定义的消息过滤。

消息属性是名称值对，可选择将其插入消息以提供结构。例如，在 .NET API 中，由常量 MessageProperties.ORIGINATOR 标识的预定义消息属性 ias_Originator 将提供发送消息的消息存储库 ID。消息属性可在传输规则中使用，以确定消息是否适合传输。

消息属性的类型共有两种：

- **预定义的消息属性** 这些消息属性始终用 ias_ 或 IAS_ 作为前缀。请参见“预定义的消息属性”一节第 687 页。
- **自定义消息属性** 这些消息属性可由您进行定义。不能以 ias_ 或 IAS_ 作为其前缀。请参见“自定义消息属性”一节第 689 页。

在这两种情况下，均可使用 get 和 set 方法来访问消息存储库属性，并可将预定义属性或自定义属性的名称作为第一个参数来进行传递。请参见“管理消息属性”一节第 689 页。

预定义的消息属性

为方便起见，预定义了某些消息属性。预定义的消息属性可以读取但不得设置。预定义的消息属性有：

- ias_Adapters
- ias_DeliveryCount
- ias_MessageType
- ias_RASNames
- ias_NetworkStatus
- ias_Originator
- ias_Status
- ias_StatusTime

有关消息属性的详细信息，请参见“消息属性”一节第 687 页。

了解目标

在 QAnywhere 中，消息被发送到目标。目标总是由标识符和队列名称组成，以反斜线 (\) 分隔。例如：

```
ianywhere.connector.tibco\SomeQueue
 DEV007\app_queue1
 SalesTeam\queue1
```

反斜线前的标识符的意义取决于消息是被发送到 JMS 应用程序、目标别名还是移动应用程序。

第一个示例说明消息被发送到 JMS 应用程序的情况。这种情况下，标识符是运行在 MobiLink 服务器中的 JMS 连接器的 ID。请参见“JMS 连接器”一节第 154 页。

第二个示例说明消息被发送到移动应用程序的情况。这种情况下，标识符是 QAnywhere 消息存储库的 ID。请参见“设置客户端消息存储库”一节第 23 页和“-id 选项”一节第 709 页。

第三个示例说明消息被发送到目标别名的情况。这种情况下，标识符是目标别名。请参见“目标别名”一节第 150 页。

标识符是 JMS 连接器 ID 时，目标中的队列名称是指在 JMS 系统中定义的队列；而标识符是消息存储库 ID 或目标别名时，队列名称指 QAnywhere 应用程序队列。

注意
应始终使用英语字符指定 QAnywhere 目标。

有关目标和发送 QAnywhere 消息的详细信息，请参见：

- “发送 QAnywhere 消息”一节第 66 页
- “确定在客户端进行消息传输的时间”一节第 51 页

消息存储库

目录

消息存储库简介

QAnywhere 消息存储库是临时存储消息的存储库。在服务器上，可以临时将消息保存在**服务器消息存储库**或**档案消息存储库**中；而在客户端上，可将消息存储在**客户端消息存储库**中，消息可分别在这些存储库中保存指定的一段时间。

有关消息存储库的详细信息，请参见：

- “服务器消息存储库”一节第 21 页
- “客户端消息存储库”一节第 23 页

服务器消息存储库

服务器消息存储库是服务器上的一个关系数据库，该数据库将临时存储消息，直到将消息传输到客户端消息存储库、Web 服务或 JMS 系统为止。消息通过服务器消息存储库在各客户端之间进行交换。

服务器消息存储库是一个 MobiLink 统一数据库，可以是 MobiLink 支持的任意 RDBMS，只有 MySQL 和 DB2 主机除外。可以为此新建一个数据库，也可以使用现有数据库。

设置服务器消息存储库

设置服务器消息存储库时，会创建档案消息存储库。档案消息存储库是与服务器消息存储库共存的一组表，它存储所有等待被删除的消息。定期执行的系统进程在消息存储库之间传送消息的方式为：通过删除服务器消息存储库中达到最终状态的所有消息，然后将它们插入到档案消息存储库中。消息一直保留在档案消息存储库中，直到服务器删除规则将其删除为止。使用档案消息存储库可以最小化同步期间需要过滤的消息数量，从而能够提高服务器消息存储库的性能。请参见“档案消息存储库请求”一节第 174 页。

若要设置数据库用作服务器消息存储库，请运行安装脚本。统一数据库应配置为在比较操作和字符串操作中都不区分大小写。如果使用 [**创建同步模型向导**] 创建统一数据库，则该向导自动为您进行设置。

请参见“建立统一数据库”一节 《MobiLink - 服务器管理》。

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果使用的 SQL Anywhere 数据库是使用 10.0.0 以前的版本创建的，则必须对数据库进行升级。

有关升级数据库的信息，请参见“升级到 SQL Anywhere 11” 《SQL Anywhere 11 - 更改和升级》。

注意
创建和维护服务器消息存储库的最简单方法存在于 Sybase Central 中。从 QAnywhere 插件任务窗格中，选择 [**服务器消息存储库**]。

示例

要创建名为 *qanytest.db* 的 SQL Anywhere 数据库，请运行以下命令：

```
dbinit -s qanytest.db
```

在该数据库上运行 MobiLink 安装脚本：

```
%SQLANY11%\MobiLink\setup\syncsa.sql
```

此数据库现在可以用作服务器消息存储库了。

服务器管理请求简介

QAnywhere 客户端应用程序可以将称为**服务器管理请求**的特殊消息发送到服务器。这些消息包含格式化为 XML 并要发送到 QAnywhere 系统队列的内容。它们需要一个特殊的验证字符串。服务器管理请求可以执行许多功能，例如查询活动客户端、查询消息存储库属性以及查询消息。

有关可使用服务器管理请求执行的功能以及如何使用它们的详细信息，请参见“编写服务器管理请求”一节第 174 页。

客户端消息存储库

客户端消息存储库可以是远程设备上的 SQL Anywhere 或 UltraLite 数据库。对于数据同步应用程序，建议使用 SQL Anywhere 数据库。对于专用于存储并转发消息的应用程序，建议使用 UltraLite 数据库。应用程序通过 QAnywhere API 连接到此消息存储库。将 UltraLite 用作客户端消息存储库时，QAnywhere API 使用 UltraLite 引擎而非 UltraLite 运行时进程来访问存储库。

客户端消息存储库必须专用于 QAnywhere 应用程序。QAnywhere 应用程序之外的任何其它应用程序都不应使用 QAnywhere API 访问 QAnywhere 消息存储库。但是，可以在数据库服务器内运行其它数据库。如果您的 QAnywhere 客户端消息存储库和 MobiLink 同步客户端运行在同一设备上，这将是很有用的。

使用关系数据库作为消息存储库提供了安全且高性能的存储库。

请参见“创建安全的客户端消息存储库”一节第 132 页。

设置客户端消息存储库

♦ **创建客户端消息存储库**

1. 创建 SQL Anywhere 或 UltraLite 数据库。

   请参见“创建数据库”一节 《SQL Anywhere 服务器 - 数据库管理》。

2. 使用以下选项，通过运行 QAnywhere 代理或 QAnywhere UltraLite 代理对每个客户端消息存储库进行初始化：

   - **-c 选项** 指定连接到您刚创建的数据库的连接字符串。

     请参见“-c 选项”一节第 706 页。

   - **-si 选项** 初始化数据库。-si 选项创建缺省数据库用户和口令。代理在初始化数据库后关闭。

     通过使用 -si 选项运行 qaagent 来初始化 QAnywhere 时，QAnywhere 代理会创建 QAnywhere 消息传递所需的客户端系统表。QAnywhere 还使用服务器系统表。这些服务器系统表在您安装 MobiLink 设置时进行创建。所有 QAnywhere 系统表的名称都以 ml_qa_ 开始且不能变更。

     请参见“-si 选项”一节第 719 页。

   - **-id 选项** 在需要预先指派客户端消息存储库 ID 时可以选择使用此选项。

     请参见“创建客户端息存储库 ID”一节第 24 页和“-id 选项”一节第 709 页。

   - **-mu 选项** 要创建用户名以用于 MobiLink 服务器验证时可以选择此选项。如果此时不使用 -mu，可在每次启动 QAnywhere 代理时随时指定此选项，如果用户名尚未存在，便会创建用户名。

3. 如果使用 -mu 选项创建用户名，则需要将用户名添加到服务器消息存储库。此操作可使用 mlsrv11 -zu+ 选项自动完成，也可采用其它方法完成。

   请参见“注册 QAnywhere 客户端用户名”一节第 31 页。

4. 更改缺省口令并执行其它步骤，以确保客户端消息存储库是安全的。

   请参见“创建安全的客户端消息存储库”一节第 132 页。

也可以升级使用 QAnywhere 的以前版本创建的客户端消息存储库。

请参见“-su 选项”一节第 720 页和“-sur 选项”一节第 721 页。

> **注意**
> 创建和维护客户端消息存储库的最简单方法存在于 Sybase Central 中。从 QAnywhere 插件任务窗格，选择 [**客户端消息存储库**]。

创建客户端息存储库 ID

如果没有指定客户端消息存储库 ID，则在使用 -si 选项运行 qaagent 后首次运行 qaagent 时，会将设备名指派为客户端消息存储库 ID。此 ID 即会显示在 [QAnywhere 代理] 窗口中。

您可能会发现手工指定 ID 很方便。您可以通过以下方法完成此操作：

- 在使用 qaagent -si 选项初始化客户端消息存储库时，使用 qaagent -id 选项指定 ID。
- 初始化客户端消息存储库后首次运行 qaagent 时，使用 -id 选项指定 ID。

请参见“QAnywhere 代理实用程序参考”第 703 页。

客户端消息存储库 ID 除了大小写之外还必须有其它不同。例如，不能有两个称为 AAA 和 aaa 的消息存储库 ID。

客户端消息存储库 ID 的字符数量限制在 128 个以下。

事务日志

建议您使用事务日志，既是因为 SQL Anywhere 数据库在使用事务日志时效率最高，也因为事务日志在数据库出现故障时会提供保护。但是，事务日志可能会变得相当大。为此，QAnywhere 代理会缺省设置 dbsrv11 -m 选项，它会导致在检查点处删除事务日志的内容。建议使用该选项。如果在 qaagent -c 选项中指定 StartLine 参数，应指定 -m。

保护客户端消息存储库

有关备份和恢复的信息，请参见“设计备份和恢复计划”一节 《SQL Anywhere 服务器 - 数据库管理》。

创建客户端消息存储库的示例

下面的命令创建名为 qanyclient.db 的 SQL Anywhere 数据库。（dbinit -i 和 -s 不是必需的选项，但用在小型设备上时效果会不错。）

```
dbinit -i -s qanyclient.db
```

以下命令连接到 *qanyclient.db* 并将其初始化为 QAnywhere 客户端数据库：

```
qaagent -si -c "DBF=qanyclient.db"
```

请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》和“QAnywhere 代理实用程序参考”第 703 页。

SQL Anywhere 和 UltraLite 客户端的差异

QAnywhere 客户端应用程序现在可以将 UltraLite 数据库用作客户端消息存储库。这为移动设备上的纯消息传递应用程序提供了更轻量化的解决方案。对于纯消息传递应用程序，我们是指具有存储并转发消息传递功能，但不进行数据同步的应用程序。

UltraLite 的一些主要优点：

- 它的应用程序占用资源空间更少且不要求完整的 SQL Anywhere 安装。
- 它的进程占用资源空间更少。QAnywhere 代理仅需要 3 个进程（uleng11、dblsn 和 qauagent），而不是 4 个（dbeng11、dbmlsync、dblsn 和 qaagent）。

在 SQL Anywhere 和 UltraLite 之间进行选择时，请记住 UltraLite 具有以下限制：

- 传输规则条件语法中没有对 "ESCAPE" 关键字的支持。
- 对属性特性的支持有限。只能将属性特性功能与预定义属性 ias_Network 配合使用。请参见“自定义客户端消息存储库属性”一节第 747 页。

通常，在所有情况下，如果 SQL Anywhere 尚未实现，则使用 UltraLite，而不是 SQL Anywhere。如果想要在已实现的 SQL Anywhere 数据同步解决方案中添加消息传递功能，则可以使用 SQL Anywhere。但是，在所有纯消息传递环境中，建议使用 UltraLite。

从应用程序的角度看，UltraLite 和 SQL Anywhere 的客户端 API 相同，除了以下例外：QAManager 配置属性应包括针对 UltraLite 消息存储库的 DATABASE_TYPE=UltraLite 设置。如果未设置属性 DATABASE_TYPE，缺省设置为 SQLAnywhere。

UltraLite 支持的客户端 API 为 C#（用于 Microsoft .NET）和 Java。UltraLite 不支持 C++ 和 SQL API。

在应用程序方面的另一个差异是 UltraLite 的 QAnywhere 代理是 qauagent.exe。UltraLite 的 QAnywhere 代理支持的选项大多与 QAnywhere 代理相同，除了以下例外：

- -sv 不适用于 UltraLite
- qauagent 不支持 -pc[+|-]
- -sur 不适用于 UltraLite
- -c 连接参数被限制为“UltraLite 连接参数” 《UltraLite - 数据库管理和参考》中介绍的那些参数

通常，UltraLite 的 QAnywhere 代理完全支持传输规则。唯一的限制就是对属性特性的支持。传输规则只能使用预定义属性 ias_Network 的以下特性：

- ias_Network.Cost
- ias_Network.CommunicationAddress
- ias_Network.CommunicationType

另请参见

- “自定义客户端消息存储库属性”一节第 747 页
- “qauagent 实用程序”一节第 725 页

客户端消息存储库属性

客户端消息存储库属性的类型共有两种：

- **预定义的消息存储库属性** 这些消息存储库属性始终用 ias_ 或 IAS_ 作为前缀。
- **自定义消息存储库属性** 以下是可以自定义的消息存储库属性。不能以 ias_ 或 IAS_ 作为其前缀。

可以使用在适当的类中定义的 get 和 set 方法来访问客户端消息存储库属性，并可将预定义属性或自定义属性的名称作为第一个参数来进行传递。

请参见“管理客户端消息存储库属性”一节第 148 页。

还可以在传输规则、删除规则和消息选择程序中使用消息存储库属性。请参见：

- “QAnywhere 传输和删除规则”第 761 页

预定义的客户端消息存储库属性

为方便起见，预定义了一些客户端消息存储库属性。预定义的消息存储库属性为：

- ias_Adapters
- ias_MaxDeliveryAttempts
- ias_MaxDownloadSize
- ias_MaxUploadSize
- ias_Network
- ias_Network.Adapter
- ias_Network.RAS
- ias_Network.IP
- ias_Network.MAC
- ias_RASNames
- ias_StoreID
- ias_StoreInitialized
- ias_StoreVersion

有关客户端消息存储库预定义属性的详细信息，请参见：

- “预定义的客户端消息存储库属性”一节第 746 页

自定义客户端消息存储库属性

QAnywhere 允许使用 QAnywhere C++、Java、SQL 或 .NET API 定义自己的客户端消息存储库属性。这些属性由连接到同一消息库的应用程序共享。它们还与服务器消息存储库同步，以便可用于此客户端的服务器端传输规则。

客户端消息存储库属性名不区分大小写。可以使用字母、数字和下划线序列，但第一个字符必须是字母。以下名称将被保留，不能用作消息存储库属性名：

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE（仅限 SQL Anywhere 消息存储库）
- 所有以 **ias_** 开头的名称

客户端消息存储库属性可以具有您定义的特性。可通过在属性名后面追加一个点，并在其后跟有特性名，来定义特性。此功能的主要用途是：可以在传输规则中使用有关您的网络的信息。

将 UltraLite 用作客户端消息存储库时，对属性特性的支持有限。UltraLite 消息存储库只支持预定义的 ias_Network 属性。

有关使用自定义客户端消息存储库属性的更多详细信息，请参见：

- “使用自定义客户端消息存储库属性特性”一节第 747 页
- “预定义的客户端消息存储库属性”一节第 746 页

设置 QAnywhere 消息传递

目录

设置服务器端组件

♦ **QAnywhere 服务器端组件设置概述**

1. 设置并启动一个服务器消息存储库。此消息存储库可以是任意 MobiLink 统一数据库。

   请参见“设置服务器消息存储库”一节第 21 页。

2. 使用 -m 选项和到服务器消息存储库的连接启动 mlsrv11。

   请参见“启动启用了 MobiLink 的 QAnywhere”一节第 30 页。

3. 将客户端用户名添加到服务器消息存储库。

   请参见“注册 QAnywhere 客户端用户名”一节第 31 页。

注意
创建和维护服务器消息存储库的最简单方法存在于 Sybase Central 中。从 QAnywhere 插件任务窗格中，选择 [**服务器消息存储库**]。

启动启用了 MobiLink 的 QAnywhere

QAnywhere 使用 MobiLink 同步来传送消息。QAnywhere 服务器是启用了消息传递的 MobiLink 服务器。请参见“建立统一数据库”一节 《MobiLink - 服务器管理》。

要运行 QAnywhere 服务器，请使用以下选项启动 MobiLink 服务器 (mlsrv11)：

- **-c *connection-string*** 指定连接字符串以连接到服务器消息存储库。这是必需的 mlsrv11 选项。

  请参见“-c 选项”一节 《MobiLink - 服务器管理》。

- **-m** 启用 QAnywhere 消息传递。

  请参见“-m 选项”一节 《MobiLink - 服务器管理》。

您还可以使用其它 MobiLink 服务器选项对操作进行自定义。有关详细信息，请参见“mlsrv11 语法”一节 《MobiLink - 服务器管理》。

注意
如果要与 JMS 消息传递系统集成，启动 MobiLink 服务器时还必须指定其它的几个选项。

请参见“启动 MobiLink 服务器以进行 JMS 集成”一节第 156 页。

示例

要在使用示例服务器消息存储库 (*samples-dir\QAnywhere\server\qanyserv.db*) 时启动 QAnywhere 消息传递，请运行以下命令：

```
mlsrv11 -m -c "dsn=QAnywhere 11 Demo"
```


QAnywhere 示例服务器消息存储库使用以下 ODBC 数据源：**QAnywhere 11 Demo**。

有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。

注册 QAnywhere 客户端用户名

每个 QAnywhere 客户端消息存储库都用一个唯一的 ID 来标识自己。另外，客户端消息存储库还有一个 MobiLink 用户名，您可以选择使用此用户名在 MobiLink 服务器上验证客户端消息存储库。可使用 qaagent -mu 选项指定 MobiLink 用户名。如果不指定，则创建一个与客户端消息存储库 ID 同名的用户名。

必须使用服务器消息存储库注册 MobiLink 用户名。可以使用以下几种方法完成此操作：

- 使用 mluser 实用程序。

  请参见“MobiLink 用户验证实用程序 (mluser)”一节 《MobiLink - 服务器管理》。
- 使用 Sybase Central 中的 [MobiLink 管理] 模式
- 指定 mlsrv11 的 -zu+ 选项。在这种情况下，任何尚未添加到统一数据库中的现有 MobiLink 用户都会在第一次进行同步时添加。此方法在开发时有用，但不建议在生产环境中使用。

  请参见“-zu 选项”一节 《MobiLink - 服务器管理》。

有关 MobiLink 用户名的详细信息，请参见“MobiLink 用户简介”一节 《MobiLink - 客户端管理》。

有关客户端消息存储库 ID 的详细信息，请参见“-id 选项”一节第 709 页。

在 QAnywhere 服务器上设置客户端属性

为方便起见，可使用 QAnywhere 插件在 QAnywhere 服务器上设置 QAnywhere 客户端的属性。这样做时，必须将客户端添加到服务器。首次与客户端同步时将下载这些属性。

♦ 使用 Sybase Central 添加客户端用户名

1. 启动 Sybase Central：
   - 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
   - 选择 [**连接**] » [**使用 QAnywhere 11 连接**]。
   - 指定一个 [**ODBC 数据源名称**] 或 [**ODBC 数据源文件**]、[**用户 ID**] 和 [**口令**]（如果需要）。单击 [**OK**]。
2. 选择 [**文件**] » [**新建**] » [**客户端**]。
3. 键入客户端名称。
4. 单击 [**OK**]。

另请参见

- “注册 QAnywhere 客户端用户名”一节第 31 页

记录 QAnywhere 服务器

QAnywhere 服务器是启用了消息传递的 MobiLink 服务器。QAnywhere 服务器日志文件为 MobiLink 日志文件。

有关 MobiLink 日志文件的信息，请参见“记录 MobiLink 服务器操作”一节 《MobiLink - 服务器管理》。

MobiLink 服务器日志文件查看器

要查看 QAnywhere 服务器的日志文件，打开 Sybase Central，然后选择 [**工具**] » [**QAnywhere 11**] » [**MobiLink 服务器日志文件查看器**]。将提示您选择要查看的日志文件。

日志查看器将读取存储在 MobiLink 日志文件中的信息。它不连接到 MobiLink 服务器或更改日志文件的组成。

日志查看器可用于过滤所查看的信息。此外，它根据日志中的信息提供统计信息。

使用中继服务器

要使用中继服务器与 MobiLink 服务器进行通信，配置 QAnywhere 代理以使用 HTTP 或 HTTPS 作为网络协议。

例如：

```
qaagent -c
"dbf=mystore.db;eng=mystore;dbn=mystore;uid=ml_qa_user;pwd=qanywhere"
     -x http(host=webserver01;port=80;url_suffix=/ias_relay_server/client/
rs_client.dll/farm1/)
```

另请参见

- “中继服务器” 《MobiLink - 服务器管理》
- “在服务器群中运行 MobiLink 服务器”一节 《MobiLink - 服务器管理》

设置客户端组件

♦ **设置客户端组件概述**

1. 创建一个 SQL Anywhere 数据库并将其初始化为客户端消息存储库。

   请参见“设置客户端消息存储库”一节第 23 页。

2. 编写客户端应用程序。

   请参见“编写 QAnywhere 客户端应用程序”第 53 页。

3. 启动 QAnywhere 代理。

   请参见“启动 QAnywhere 代理”一节第 48 页。

注意
创建和维护客户端消息存储库的最简单方法存在于 Sybase Central 中。从 QAnywhere 插件任务窗格，选择 [**客户端消息存储库**]。

使用推式通知

推式通知是从服务器消息存储库传送到 QAnywhere 客户端的特殊消息，该消息提示客户端开始消息传输。缺省情况下，启用推式通知，但此功能是可选的。推式通知向 QAnywhere 体系结构引入了额外组件：

- 在服务器端，QAnywhere 通告程序发送推式通知。
- 在客户端，QAnywhere 监听器接收这些推式通知，并将这些通知传递给 QAnywhere 代理。
- 在客户端，将每个推式通知的通知发送到系统队列。

如果您使用 [调度] 或 [自动] QAnywhere 代理策略，推式通知将自动促使客户端启动消息传输。如果使用 [要求时] 策略，则必须使用事件处理程序手工处理推式请求。

有关手工处理推式通知的详细信息，请参见“有关推式通知的通知”一节第 65 页。

有关 QAnywhere 代理策略的详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

缺省情况下将启用推式通知：因为 qaagent -push 选项缺省设置为已连接。在已连接模式下，将通过 TCP/IP 持久连接发送推式通知。

如果使用的是 UDP，推式通知将很可能无需进行任何配置即可工作，但由于 ActiveSync 在 UDP 实现方面存在限制，因此推式通知不能与 ActiveSync 一起使用。

另请参见

- “使用推式通知进行消息传递的方案”一节第 6 页
- “有关推式通知的通知”一节第 65 页
- “-push 选项”一节第 717 页

配置推式通知

推式通知是一种特殊消息，当消息到达该客户端指定的服务器消息存储库时，此消息将从 QAnywhere 服务器发送到 QAnywhere 客户端。推式通知由服务器上运行的名为**通告程序**的程序发送，由在客户端上运行的名为**监听器**的程序接收。推式通知通过**网关**发送。客户端接收到推式通知时，它将启动消息传输以接收在服务器处等候的消息，或采取某种自定义操作。

通告程序、监听器和网关已经配置，无需任何改动即可在 QAnywhere 中使用。在极少数情况下，可能希望对它们进行配置。此外，可能希望更改某些通告程序设置。请参见：

- “配置 QAnywhere 通告程序”一节第 35 页
- “配置监听器”一节第 36 页
- “配置 QAnywhere 网关”一节第 37 页

可以禁用推式通知，从而不使用通告程序或监听器。请参见“-push 选项”一节第 717 页。

有关客户端对推式通知的响应的信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

配置 QAnywhere 通告程序

QAnywhere 通告程序由 MobiLink 安装脚本创建，并在使用 -m 选项运行 MobiLink 服务器时启动。QAnywhere 通告程序称为 QAnyNotifier_client。

QAnyNotifier_client 使用“用于服务器启动的同步的 MobiLink 服务器设置” 《MobiLink - 服务器启动的同步》中介绍的缺省设置，但以下设置除外：

- gui 属性设置为 off，这意味着在运行通告程序的计算机上不显示 [通告程序] 窗口。
- enable 属性设置为 no，这意味着必须使用 -m 选项运行 mlsrv11，进而启动通告程序。
- poll_every 属性设置为 5，这意味着通告程序每五秒轮询一次，以查看是否需要发送推式通知。

可以更改以下通告程序属性：

- poll_every 属性
- request_cursor 属性中的 resend interval
- request_cursor 属性中的 time to live

注意
除上述列出的三个属性之外，不要更改任何通告程序属性。不要更改 request_cursor 中的任何其它列。

Poll_every 属性

通过更改以下代码中的值 5，然后针对您的统一数据库运行该代码，可以更改 QAnyNotifier_client 的缺省轮询间隔：

```
CALL ml_add_property( 'SIS', 'Notifier(QAnyNotifier_client)', 'poll_every',
'5' )
```

请参见“通告程序属性”一节 《MobiLink - 服务器启动的同步》。

Resend interval 和 time to live

QAnywhere 通告程序包含缺省 request_cursor。request_cursor 确定在推送请求中发送哪些信息、接收信息的对象、时间以及地点。除 resend interval 和 time to live 之外，不要更改任何缺省设置。resend interval 指定未接收到的推式通知缺省情况下每 5 分钟重新发送一次。time to live 指定未接收到的推式通知在缺省情况下重新发送 3 小时。大多数情况下，这些缺省设置是最佳的。以下是随 QAnNotifier_client 提供的缺省 request_cursor：

```
SELECT
    u.user_id,
    'Default-DeviceTracker',
    'qa',
    u.name,
    u.name,
    '5M',
    '3H'
    FROM ml_qa_notifications u
    WHERE EXISTS(SELECT *
        FROM ml_listening l
        WHERE l.name = u.name AND l.listening = 'y')
```

有关 request_cursor 中列的详细信息，请参见“推式请求要求”一节 《MobiLink - 服务器启动的同步》。

通过更改以下代码中的值 5M，可以将 resend interval 从缺省值 5 分钟更改为其它值。同样，通过更改值 3H 可以更改 time to live 的缺省值 3 小时。

```
CALL ml_add_property(
  'SIS',
  'Notifier(QAnyNotifier_client)',
  'request_cursor',
  'select u.user_id,
  ''Default-DeviceTracker'',
  ''qa'',
  u.name,
  u.name,
  ''5M'',
  ''3H''
     FROM ml_qa_notifications u
     WHERE EXISTS(
        SELECT *
         FROM ml_listening l WHERE l.name = u.name AND l.listening = ''y'')' )
```

有关详细信息，请参见“request_cursor 事件”一节 《MobiLink - 服务器启动的同步》。

另请参见

- “配置通告程序事件和属性”一节 《MobiLink - 服务器启动的同步》
- “用于服务器启动的同步的 MobiLink 服务器设置” 《MobiLink - 服务器启动的同步》
- “通告程序”一节 《MobiLink - 服务器启动的同步》
- “ml_add_property 系统过程”一节 《MobiLink - 服务器管理》
- “推式请求”一节 《MobiLink - 服务器启动的同步》

配置监听器

监听器与客户端消息存储库运行在同一台设备上。监听器从服务器接收推式通知并将它们传递到 QAnywhere 代理。

监听器已经配置可在 QAnywhere 中使用。在极少数情况下，可能希望更改缺省行为。

例如，如果将 QAnywhere 使用的网关更改为 SMS 网关，需要使用不同选项手工启动监听器。假定 QAnywhere 消息存储库 ID 为 mystore，您的 MobiLink 主机为 *acme.com*，并且您希望使用 SMS 库 *maac555.dll* 启动监听器，用于在 AirCard 555 上监听 SMS 消息。那么，您需要使用以下命令启动监听器：

```
dblsn.exe -u ias_mystore_lsn -e mystore -t+ mystore
  -x "tcpip(host=acme.com)" -pc- -d lsn_udp.dll -a "port=5001" -d
maac555.dll
  -i 60
```

为使 QAnywhere 代理找到刚启动的监听器，还需要按如下所示重新启动 QAnywhere 代理：

```
qaagent -c "dbf=mystore.db;eng=mystore;dbn=mystore" -id mystore
  -lp 5001-x tcpip(host=acme.com)
```


另请参见

- “监听器”一节 《MobiLink - 服务器启动的同步》
- “用于 Windows 设备的监听器实用程序”一节 《MobiLink - 服务器启动的同步》
- “配置 QAnywhere 网关”一节第 37 页

配置 QAnywhere 网关

发送推式通知是通过网关进行的。缺省情况下，QAnywhere 使用缺省设备跟踪器网关。设备跟踪器网关首先尝试使用 SYNC 网关，SYNC 网关使用的协议与用于 MobiLink 同步的协议相同，并且 SYNC 网关是永久性的。多数情况下，缺省设备跟踪器网关是发送推式通知的最佳途径。不过，也可以选择使用 SMS 或 UDP 网关。

要配置网关，请参见“用于服务器启动的同步的 MobiLink 服务器设置” 《MobiLink - 服务器启动的同步》和“网关属性”一节 《MobiLink - 服务器启动的同步》。

要使用 SMS 网关，需要使用新选项启动监听器。请参见“配置监听器”一节第 36 页。

要使用 UDP 网关，需要设置 qaagent 的 -push 断开连接选项。请参见“-push 选项”一节第 717 页。

另请参见

- “网关和运营公司”一节 《MobiLink - 服务器启动的同步》

设置故障转移机制

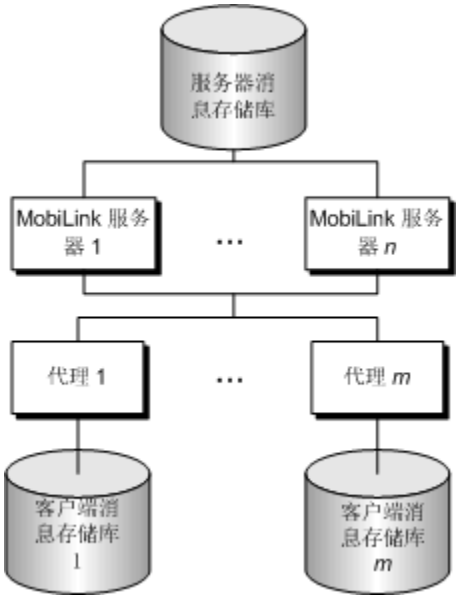
可以为 QAnywhere 应用程序设置故障转移机制，以便在 MobiLink 服务器出现故障时可以用备用服务器替代。要支持故障转移，每个 QAnywhere 代理在启动时都必须带有一个 MobiLink 服务器列表。列表中指定的第一个 MobiLink 服务器是主服务器。列表中剩余的服务器则是备用服务器。

例如，在远程设备上运行下面的命令会启动 QAnywhere 代理，并且指定一个主服务器和一个备用服务器：

```
qaagent -x tcpip(host=ml1.ianywhere.com)
        -x tcpip(host=ml2.ianywhere.com)
```

每个 QAnywhere 代理可以有一个不同的主服务器。

下图描述了一个故障转移配置，其中有多个 MobiLink 服务器和多个 QAnywhere 代理。您有多个客户端消息存储库，但所有 MobiLink 服务器都连接到同一个服务器端消息存储库。



此配置具有下列特点：

- 传输消息时，不论 QAnywhere 代理连接到哪个服务器，服务器消息存储库中的所有消息都将被传送到该客户端消息存储库。
- 只有在 QAnywhere 代理连接到自己的主服务器时，才会向 QAnywhere 代理发送推式通知。
- 这存在一个缺陷。如果服务器消息存储库所在的计算机不可用，则无法执行消息传递。

缺省情况下，如果设置了故障转移 MobiLink 服务器，QAnywhere 代理始终会在连接到主服务器失败后立即尝试连接到备用服务器。如果要更改此缺省行为，可使用 QAnywhere 代理 -fr 选项，此选项可使 QAnywhere 代理在转向备用服务器之前再次尝试连接到主服务器，并可指定重试次数。可使用 -fd 选项指定重试连接主服务器之间的间隔时间。

-fr 和 -fd 选项仅适用于主服务器。如果进行指定的尝试次数后仍无法建立与主服务器的连接，QAnywhere 代理便尝试连接到备用服务器。此代理只尝试连接到每个备用服务器一次。如果代理无法建立与备用服务器的连接，则会发出错误消息。

另请参见

- “-x 选项”一节第 723 页
- “-fd 选项”一节第 707 页
- “-fr 选项”一节第 708 页
- “启动 QAnywhere 代理”一节第 48 页

QAnywhere 代理简介

目录

QAnywhere 代理 (qaagent) 是在客户端设备上运行的独立进程。它监控客户端消息存储库并确定应何时进行消息传输。

QAnywhere 代理在服务器消息存储库和客户端消息存储库之间传输消息。可在同一台设备上运行 QAnywhere 代理的多个实例，但每个实例必须连接到其自己的消息存储库。每个消息存储库必须有唯一的消息存储库 ID。

消息传输策略

QAnywhere 客户端使用策略确定何时进行消息传输。策略包括：

- “[调度] 策略”一节第 42 页
- “[自动] 策略”一节第 43 页
- “[要求时] 策略”一节第 43 页
- “自定义策略”一节第 44 页

[调度] 策略以指定的时间间隔启动消息传输。每当在客户端上有消息准备收发时，[自动] 策略便会启动消息传输。[要求时] 策略允许用户控制消息传输。[自定义] 策略使用各种传输规则对消息传输进行更多的控制。

[调度] 策略

[调度] 策略指示代理以指定的时间间隔执行消息传输。

若要调用 [调度] 策略，在 [**命令文件属性**] 窗口中选择 **scheduled** 或在启动 QAnywhere 代理时指定此关键字：

qaagent -policy scheduled [*interval*] ...

其中 *interval* 以秒为单位。

缺省值为 900 秒（15 分钟）。

指定调度后，只要满足以下任何一个条件，代理就将每隔 *n* 秒钟执行一次消息传输：

- 前一个时间间隔结束后在客户端消息存储库中放入了新的消息。
- 前一个时间间隔结束后某条消息的状态发生了更改。当某条消息得到应用程序的确认时通常会发生这种情况。

  有关确认的详细信息，请参见：

  - .NET： “AcknowledgementMode 枚举”一节第 212 页
  - C++： “AcknowledgementMode 类”一节第 386 页
  - Java： “AcknowledgementMode 接口”一节第 498 页
- 前一个时间间隔结束后接收到推式通知。
- 前一个时间间隔结束后接收到网络状态更改通知。
- 推式通知被禁用。

可调用 TriggerSendReceive 方法来替换时间间隔。该方法会强制在时间间隔结束前传输消息。请参见：

- .NET： “TriggerSendReceive 方法”一节第 297 页
- C++： “triggerSendReceive 函数”一节第 453 页
- Java： “triggerSendReceive 方法”一节第 570 页
- SQL： “ml_qa_triggersendreceive”一节第 682 页

[自动] 策略

[自动] 策略会通过每当发送或接收消息时进行同步，来使客户端消息存储库和服务器消息存储库尽可能地保持最新。建议经常发送和接收消息的应用程序不要使用此策略。

使用 [自动] 策略时，如果出现以下任一种情况，就会执行消息传输：

- 调用了 PutMessage()。请参见：
  - .NET： “PutMessage 方法”一节第 283 页
  - C++： “putMessage 函数”一节第 444 页
  - Java： “putMessage 方法”一节第 560 页
  - SQL： “ml_qa_putmessage”一节第 681 页
- 消息状态发生了更改。当某条已接收消息得到应用程序的确认时通常会发生这种情况。请参见：
  - .NET： “AcknowledgementMode 枚举”一节第 212 页
  - C++： “AcknowledgementMode 类”一节第 386 页
  - Java： “AcknowledgementMode 接口”一节第 498 页
  - SQL：使用 SQL API 进行的所有消息传递均是事务性的
- 接收到推式通知。

  请参见“使用推式通知”一节第 34 页。
- 接收到网络状态更改通知。

  请参见“有关推式通知的通知”一节第 65 页。
- 调用了 TriggerSendReceive()。请参见：
  - .NET： “TriggerSendReceive 方法”一节第 297 页
  - C++： “triggerSendReceive 函数”一节第 453 页
  - Java： “triggerSendReceive 方法”一节第 570 页
  - SQL： “ml_qa_triggersendreceive”一节第 682 页

[要求时] 策略

[要求时] 策略指定只在某个应用程序要求执行消息传输时才会传输消息。

应用程序通过调用 TriggerSendReceive() 来强制执行消息传输。

当代理收到推式通知或网络状态更改通知时，会将相应的消息发送到系统队列。这使得应用程序可以检测这些事件，并可通过调用 TriggerSendReceive() 方法来强制执行消息传输。请参见：

- .NET： “TriggerSendReceive 方法”一节第 297 页
- C++： “triggerSendReceive 函数”一节第 453 页
- Java： “triggerSendReceive 方法”一节第 570 页
- SQL： “ml_qa_triggersendreceive”一节第 682 页

有关处理推式通知和网络状态更改的详细信息，请参见“系统队列”一节第 64 页。

自定义策略

自定义策略可用于定义何时传输消息以及传输哪些消息。

为您的应用程序创建 [自定义] 策略规则时，建议您包括一个缺省的包含所有情况的规则，以使消息不会被其它规则意外忽略。例如，以下规则同步至少已有一天的消息。

```
auto=DATEADD( day, 1, ias_statustime ) < ias_currenttimestamp
```

以下是影响同步有效性的因素的列表，创建您自己的 [自定义] 策略规则时应考虑到这些因素。

- 消息大小
- 同步频率
- 带宽和网络可靠性
- 消息传递优先级
- 数据传输开销

自定义策略由一组传输规则定义。

每个规则的形式如下：

schedule = *condition*

其中 *schedule* 定义何时评估 *condition*。有关详细信息，请参见“规则语法”一节第 762 页。

所有满足 *condition* 的消息都将被传输出去。特别是，如果 *schedule* 为自动，则在发生以下任何一种情况时都将对条件进行评估：

- 调用了 PutMessage()。请参见：
  - .NET： “PutMessage 方法”一节第 283 页
  - C++： “putMessage 函数”一节第 444 页
  - Java： “putMessage 方法”一节第 560 页
  - SQL： “ml_qa_putmessage”一节第 681 页
- 消息状态发生了更改。当某条消息得到应用程序的确认时通常会发生这种情况。请参见：
  - .NET： “AcknowledgementMode 枚举”一节第 212 页
  - C++： “AcknowledgementMode 类”一节第 386 页
  - Java： “AcknowledgementMode 接口”一节第 498 页
  - SQL：使用 SQL API 进行的所有消息传递均是事务性的
- 接收到推式通知。

  请参见“使用推式通知”一节第 34 页。
- 接收到网络状态更改通知。

- 调用了 TriggerSendReceive()。请参见：
  - .NET： “TriggerSendReceive 方法”一节第 297 页
  - C++： “triggerSendReceive 函数”一节第 453 页
  - Java： “triggerSendReceive 方法”一节第 570 页
  - SQL： “ml_qa_triggersendreceive”一节第 682 页

了解传输状态

确定消息传输状态最简单的方法是使用 Sybase Central。转到 [**消息属性**] 窗口的 [**常规**] 选项卡查看消息传输状态。这些可能的值为：

- **已传输** 消息已发送。
- **正在传输** 正在发送消息。
- **未传输** 消息未发送。
- **do_not_transmit** 不应发送消息。

了解消息状态

确定消息状态最简单的方法是使用 Sybase Central。转到 [**消息属性**] 窗口的 [**常规**] 选项卡查看消息状态。这些可能的值为：

- **待执行** 消息已发送但未被接收。
- **正在接收** 消息正处于接收过程中，或者已接收但未确认。
- **最终** 消息已达到最终状态。
- **已到期** 消息在达到到期时间之前未被接收。
- **已取消** 消息已取消。
- **无法收到** 消息格式错误或尝试发送时失败次数过多。
- **已接收** 消息已接收并得到确认。

传输规则

消息传输是在客户端消息存储库和服务器消息存储库之间移动消息的操作。

消息传输由 QAnywhere 代理和 MobiLink 服务器处理：

- QAnywhere 代理连接到客户端消息存储库。它将消息传入或传出 MobiLink 服务器。
- MobiLink 服务器连接到服务器消息存储库。它从 QAnywhere 代理接收传输消息并将它们传输到其它 QAnywhere 代理。

消息传输只能在客户端消息存储库和服务器消息存储库之间进行。只有当 QAnywhere 代理连接到 MobiLink 服务器时才能进行消息传输。

在 QAnywhere 中，规则是用于确定何时进行消息传输、传输哪些消息以及应在何时删除消息的逻辑。可以在客户端上和服务器上指定规则。

规则有两个部分：调度和条件。调度定义事件发生时间。条件定义哪些消息将成为事件的一部分。例如，如果事件为消息传输，则调度指示何时进行传输，而条件则定义传输中将包括哪些消息。如果事件为消息删除，则调度指示何时进行删除，而条件则指示删除哪些消息。

传输规则可用于指定何时进行消息传输以及要传输哪些消息。可以为客户端和服务器都指定传输规则。

有关如何指定传输规则的详细信息，请参见：

- “客户端传输规则”一节第 769 页
- “服务器传输规则”一节第 770 页
- “规则语法”一节第 762 页
- “规则变量”一节第 767 页

删除规则

如果不想使用缺省行为，还可以使用删除规则来指定应在何时从消息存储库中删除消息。可以为客户端和服务器都指定删除规则。

有关使用删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

启动 QAnywhere 代理

可在命令行上使用命令行选项运行此代理。启动此代理时至少需要使用以下选项：

- **连接参数** 连接到客户端消息存储库。

  在 [**代理配置文件属性**] 窗口中，这是 [**消息存储库**] 选项卡上的信息。

  在 qaagent 命令行中，这是使用 -c 选项指定的。

  请参见“-c 选项”一节第 706 页。

- **客户端消息存储库 ID** 标识客户端消息存储库。初始化客户端消息存储库后第一次运行 qaagent 时，可以选择使用此选项命名消息存储库；如果不使用此选项，缺省情况下将使用设备的名称。如果这样，每次启动 qaagent 时都必须使用 -id 选项指定唯一的客户端消息存储库 ID。

  在 [**代理配置文件属性**] 窗口中，这在 [**常规**] 选项卡上指定。

  在 qaagent 命令行中，这是使用 -id 选项指定的。

  请参见“-id 选项”一节第 709 页。

- **网络协议和协议选项** 连接到 MobiLink 服务器。这是必需的操作，除非 MobiLink 服务器和 QAnywhere 代理在同一台设备上运行并使用缺省通信参数。

  在 [**代理配置文件属性**] 窗口中，这是 [**服务器**] 选项卡上的服务器信息。

  在 qaagent 命令行中，这是 -x 选项。

  请参见“-x 选项”一节第 723 页。

有关所有 QAnywhere 代理选项的完整列表，请参见“qaagent 实用程序”一节第 704 页。

在 Windows Mobile 上启动 qaagent

在 Windows Mobile 上，通过指定 -qi 选项可在安静模式下启动 QAnywhere 代理。

请参见“-qi 选项”一节第 719 页。

运行 QAnywhere 代理的多个实例

可在一台设备上运行 qaagent 的多个实例。但是，启动第二个实例时：

- 必须使用不同的数据库文件启动 QAnywhere 代理的第二个实例。
- 必须使用 -id 选项指定唯一的消息存储库 ID。

  请参见“-id 选项”一节第 709 页。

停止 QAnywhere 代理

要停止 QAnywhere 代理，单击 QAnywhere 代理消息窗口中的 [**关闭**]。

在安静模式下启动 QAnywhere 代理时，只能通过运行 **qastop** 停止它。

另请参见

- “qastop 实用程序”一节第 743 页
- “-qi 选项”一节第 719 页

通过 QAnywhere 代理启动的进程

QAnywhere 代理启动其它进程以处理各种消息传递任务。这些进程都由 QAnywhere 代理管理，不需要另外管理。QAnywhere 代理启动时，将生成以下进程：

- **dbmlsync** dbmlsync 可执行文件是 MobiLink 同步客户端。dbmlsync 可执行文件用于发送和接收消息。

  小心
  不要在独立于 qaagent 的 QAnywhere 消息存储库上运行 dbmlsync。

- **dblsn** dblsn 可执行文件是监听器实用程序。它接收推式通知。如果不使用推式通知，在部署应用程序时就不需要提供 dblsn 可执行程序，必须使用 -push **none** 运行 qaagent。

  请参见“-push 选项”一节第 717 页。

- **数据库服务器** 客户端消息存储库是 SQL Anywhere 或 UltraLite 数据库。QAnywhere 代理要求数据库服务器运行此数据库。对于 Windows Mobile，数据库服务器是 *dbsrv11.exe*。对于 Windows，数据库服务器是个人数据库服务器 *dbeng11.exe*。

  根据在 qaagent -c 选项中指定的通信参数，QAnywhere 代理可以生成数据库服务器或连接到正在运行的服务器。

  请参见“-c 选项”一节第 706 页。

部署 QAnywhere 代理

有关部署信息，请参见“部署 QAnywhere 应用程序”一节第 126 页。

确定在客户端进行消息传输的时间

在客户端，通过指定**策略**来确定消息传输的时间。策略会告诉 QAnywhere 代理应将消息从客户端消息存储库移动到服务器消息存储库的时间。如果不指定策略，缺省情况下，当消息排队等待传送到服务器时自动传输消息。共有一个自定义策略和三个预定义策略：[调度]、[自动] 和 [要求时]。

可通过以下方法指定策略：

- 使用 Sybase Central 的 QAnywhere 插件，选择 [**工具**] » [**QAnywhere 11**] » [**SQL Anywhere 的新代理配置文件**]。策略在 [**代理配置文件属性**] 窗口中的 [**常规**] 选项卡上指定。此任务创建一个扩展名为 *.qaa* 的文件，这是 Sybase Central 约定。

  要使用 Sybase Central 的 QAnywhere 插件指定自定义属性，选择 [**工具**] » [**QAnywhere 11**] » [**新代理规则文件**]。此任务创建一个扩展名为 *.qar* 的文件，这是 Sybase Central 约定。

- 在命令行上使用 -policy 选项运行 qaagent。对于自定义策略，创建一个规则文件并指定它。

另请参见

- “消息传输策略”一节第 42 页
- “传输规则”一节第 46 页
- “-policy 选项”一节第 716 页

处理不可靠网络

没有增量上载和增量下载时，消息以个体发送，所以，如果正在上载或下载消息时网络连接丢失，则消息传输将会失败。使用增量上载和增量下载时，大的消息将分为多个小的消息块。通过将消息以多个小块的形式发送，可单独发送各消息块，这使消息可在多次同步中逐步上载或下载。所有消息块都到达目标后，完整的消息也就到达了目标。

有关如何实现增量上载的信息，请参见“-iu 选项”一节第 710 页。

有关如何实现增量下载的信息，请参见“-idl 选项”一节第 709 页。

编写 QAnywhere 客户端应用程序

目录

QAnywhere 接口简介

QAnywhere 客户端应用程序管理 QAnywhere 消息的接收和发送。可以使用以下几种 QAnywhere API 中的一种来编写此类应用程序：

- QAnywhere .NET API
- QAnywhere C++ API
- QAnywhere Java API
- QAnywhere SQL API

在您的 QAnywhere 系统中，可将几种客户端类型组合使用。例如，使用 QAnywhere SQL 生成的消息也可由使用 .NET API、C++ API 或 Java API 创建的客户端接收。如果已在服务器上配置了一个 JMS 连接器，则消息也可由 JMS 客户端接收。同样，QAnywhere SQL 可用来接收由 QAnywhere .NET、C++、Java 或 JMS 客户端生成的消息。

QAnywhere .NET API

QAnywhere .NET API 是部署到使用 Microsoft .NET Framework 的 Windows 计算机和运行 Microsoft .NET Compact Framework 的手持式设备的编程接口。QAnywhere .NET API 是以 iAnywhere.QAnywhere.Client 命名空间的形式提供的。

QAnywhere 支持 Microsoft Visual Studio。

注意
虽然在本文档中 .NET API 代码示例使用的是 C# 编程语言，但是可以使用 Microsoft .NET 支持的任何编程语言来访问此 API。

几个版本的 TestMessage 示例应用程序分别使用了 Java、C# 和 Visual Basic.NET 来编写。同时还有一个 .NET compact framework 示例。

有关 .NET 版本的 TestMessage 示例应用程序的详细信息，请参见“第 4 课：探讨 TestMessage 客户端源代码”一节第 204 页。

请参见“用于客户端的 QAnywhere .NET (.NET 2.0)”一节第 212 页。

QAnywhere C++ API

QAnywhere C++ API 支持 Microsoft Visual Studio。

QAnywhere C++ API 由以下文件组成：

- 一组头文件（主要文件是 *qa.hpp*），位于 *install-dir\sdk\include* 下。
- 一个导入库 (*qany11.lib*)，位于 *install-dir\sdk\lib\x86* 和 *install-dir\sdk\lib\ce\arm.50* 下。
- 运行时 DLL (*qany11.dll*)，位于 *install-dir\bin32* 和 *install-dir\ce\arm.50* 下。

要访问 API，源代码文件必须包含头文件。导入库用于将应用程序链接到运行时 DLL。运行库 DLL 必须与应用程序一起部署。

一个用 C++ 编写的 TestMessage 示例应用程序，位于 *samples-dir\QAnywhere\Desktop\MFC*。

请参见“QAnywhere C++ API 参考”第 385 页。

QAnywhere Java API

QAnywhere Java API 支持 JRE 1.4.2 及更高版本。移动 Web 服务 WSDL 编译器生成的 Java 类与 JDK 1.5.0 及更高版本兼容。

QAnywhere Java API 由以下文件组成：

- API 参考资料，它在本书中提供，或在 SQL Anywhere 11 安装目录的 *documentation\zh\javadocs\QAnywhere* 的子目录中以 Javadoc 格式提供。
- 用于 UltraLite 消息存储库的运行库 DLL (*qadbiuljni.dll*)，位于 SQL Anywhere 11 安装目录的 *bin32* 子目录中。
- 类文件的档案 (*qaclient.jar*)，位于 SQL Anywhere 11 安装目录的 *java* 子目录中。

编译应用程序时，必须在路径中包括此类文件档案。运行库 DLL 必须与应用程序一起部署。

一个用 Java 编写的 TestMessage 示例应用程序，位于 *samples-dir\QAnywhere\Desktop\MFC*。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）

请参见“QAnywhere Java API 参考”第 497 页。

QAnywhere SQL API

QAnywhere SQL API 是在 SQL 中实现消息传递 API 的一组存储过程。使用 QAnywhere SQL API，您可以创建消息、设置或获取消息属性和内容、发送和接收消息、触发消息同步，以及设置和获取消息存储库属性。

请参见“QAnywhere SQL API 参考”第 645 页。

JMS 连接器

QAnywhere 包括一个可在 QAnywhere 和 JMS 应用程序之间提供连接的 JMS 连接器。请参见：

- “使用外部消息系统进行消息传递的方案”一节第 7 页
- “JMS 连接器”一节第 154 页
- “教程：使用 JMS 连接器”一节第 166 页

移动 Web 服务连接器

QAnywhere 包括一个在 QAnywhere 和 Web 服务之间传递消息时使用的移动 Web 服务连接器。

请参见“移动 Web 服务”第 101 页。

编写客户端应用程序快速入门

♦ **设置客户端应用程序概述**

1. 初始化适当的 QAnywhere API。请参见：
   - “设置 .NET 应用程序”一节第 57 页
   - “设置 C++ 应用程序”一节第 59 页
   - “设置 Java 应用程序”一节第 60 页
   - “设置 SQL 应用程序”一节第 61 页
2. 设置 QAnywhere Manager 配置属性。请参见“QAnywhere Manager 配置属性”一节第 90 页。
3. 编写应用程序代码并进行编译。请参见：
   - “QAnywhere 消息简介”一节第 14 页
   - “客户端消息存储库属性”一节第 26 页
   - “发送 QAnywhere 消息”一节第 66 页
   - “接收 QAnywhere 消息”一节第 74 页
   - “读取非常大的消息”一节第 79 页
   - “实现事务性消息传递”一节第 67 页
   - “关闭 QAnywhere”一节第 88 页
4. 将应用程序部署到目标设备。

   请参见“部署 QAnywhere 应用程序”一节第 126 页。

其它入门资源

- “教程：探讨 TestMessage”第 197 页
- 示例应用程序安装在 *samples-dir\QAnywhere* 中。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）

初始化 QAnywhere API

使用 QAnywhere 发送或接收消息之前，必须完成下面的初始化任务。

设置 .NET 应用程序

使用 QAnywhere .NET 客户端发送或接收消息之前，必须完成以下初始化任务。

Visual Studio 项目必须进行两项更改后才能使用：

- 添加对 QAnywhere .NET DLL 的引用。添加引用可通知 Visual Studio .NET 包含哪个 DLL，以找到 QAnywhere .NET API 的代码。
- 在源代码中添加一行以引用 QAnywhere .NET API 类。要使用 QAnywhere .NET API，您必须在源代码中添加一行以引用数据提供程序。必须为 C# 添加与 Visual Basic .NET 不同的语句行。

此外，您还必须初始化 QAnywhere .NET API。

♦ 在 Visual Studio 项目中添加对 QAnywhere .NET API 的引用

1. 启动 Visual Studio 并打开项目。
2. 在 [**Solution Explorer**] 窗口中，右击 [**References**] 文件夹，然后选择 [**Add Reference**]。
3. 在 [**.NET**] 选项卡上，单击 [**Browse**] 来查找 *iAnywhere.QAnywhere.Client.dll*。缺省位置为：
   - .NET Framework 2.0：*install-dir\Assembly\v2*
   - .NET Compact Framework 2.0：*install-dir\ce\Assembly\v2*

   选择 DLL 然后单击 [**Open**]。
4. 可以验证 DLL 是否已添加到项目中。打开 [**Add Reference**] 窗口然后单击 [**.NET**] 选项卡。随即 *iAnywhere.QAnywhere.Client.dll* 出现在 [**Selected Components**] 列表中。单击 [**OK**] 关闭窗口。

在源代码中引用数据提供程序类

♦ 在代码中引用 QAnywhere .NET API 类

1. 启动 Visual Studio 并打开项目。
2. 如果使用 C#，请将以下语句行添加到文件开始处的 using 指令列表中：

   ```
   using iAnywhere.QAnywhere.Client;
   ```
3. 如果使用 Visual Basic，请将以下语句行添加到文件开始处的导入列表中：

   ```
   Imports iAnywhere.QAnywhere.Client
   ```

   并不严格要求使用该语句行。但是，它使您可以使用 QAnywhere 类的简写形式。即使没有该语句，您仍然可以在代码中使用完全限定的类名。例如：

   ```
   iAnywhere.QAnywhere.Client.QAManager
   mgr =
   ```

```
 new iAnywhere.QAnywhere.Client.QAManagerFactory.Instance.CreateQAManager(
"qa_manager.props" );
```

可代替

```
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(
 "qa_manager.props" );
```

♦ 初始化 QAnywhere .NET API

1. 包括 iAnywhere.QAnywhere.Client 命名空间，如上面的过程中所述。

   ```
   using iAnywhere.QAnywhere.Client;
   ```

2. 创建 QAManager 对象。

   例如，要创建缺省的 QAManager 对象，可调用 CreateQAManager（以 null 作为其参数）：

   ```
   QAManager mgr;
   mgr = QAManagerFactory.Instance.CreateQAManager( null );
   ```

   提示
   为了提供最大并发数，多线程应用程序应为每个线程创建一个 QAManager。请参见“多线程注意事项”一节第 89 页。

   有关 QAManagerFactory 的详细信息，请参见“QAManagerFactory 类”一节第 298 页。

   也可以创建使用属性文件自定义的 QAManager 对象。属性文件在 CreateQAManager 方法中指定：

   ```
   mgr = QAManagerFactory.Instance.CreateQAManager(
     "qa_mgr.props" );
   ```

   其中，*qa_mgr.props* 是远程设备上属性文件的名称。

3. 初始化 QAManager 对象。例如：

   ```
   mgr.Open(
      AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
   ```

   open 方法的参数是确认模式，它指示确认消息的方式。它必须为 IMPLICIT_ACKNOWLEDGEMENT 或 EXPLICIT_ACKNOWLEDGEMENT。使用隐式确认时，客户端在收到消息时会进行确认。使用显式确认时，必须在 QAManager 上调用 Acknowledge 方法对消息进行确认。

   有关确认模式的详细信息，请参见“AcknowledgementMode 枚举”一节第 212 页。

现在，您就可以发送消息了。

您可以创建 QATransactionalManager 而非 QAManager。请参见“为 .NET 客户端实现事务性消息传递”一节第 68 页。

另请参见

- “用于客户端的 QAnywhere .NET (.NET 2.0)”一节第 212 页

设置 C++ 应用程序

使用 QAnywhere C++ 客户端发送或接收消息之前，必须完成以下初始化任务。

♦ 初始化 QAnywhere C++ API

1. 包括 QAnywhere 头文件。

   ```
   #include <qa.hpp>
   ```

   qa.hpp 定义 QAnywhere 类。

2. 初始化 QAnywhere。

   为此，初始化用于创建 QAManager 对象的工厂。

   ```
   QAManagerFactory * factory;

   factory = QAnywhereFactory_init();
   if( factory == NULL ) {
         // Fatal error.
   }
   ```

   有关 QAManagerFactory 的详细信息，请参见“QAManagerFactory 类”一节第 454 页。

3. 创建 QAManager 实例。

   您可以按以下方法创建缺省 QAManager 对象：

   ```
   QAManager *  mgr;

   // Create a manager
   mgr = factory->createQAManager( NULL );
   if( mgr == NULL ) {
     // fatal error
   }
   ```

   请参见“QAManager 类”一节第 420 页。

   提示
   为了提供最大并发数，多线程应用程序应为每个线程创建一个 QAManager。请参见“多线程注意事项”一节第 89 页。

   可以以编程方式或使用属性文件自定义 QAManager 对象。

   - 要以编程方式自定义 QAManager，请使用 setProperty()。

     请参见“以编程方式设置 QAnywhere Manager 配置属性”一节第 92 页。

   - 要使用属性文件，请在 createQAManager() 中指定属性文件：

     ```
     mgr = factory->createQAManager( "qa_mgr.props" );
     ```

其中，*qa_mgr.props* 是远程设备上属性文件的名称。

请参见“在文件中设置 QAnywhere Manager 配置属性”一节第 91 页。

4. 初始化 QAManager 对象。

```
qa_bool  rc;
rc=mgr->open(
     AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT );
```

open 方法的参数是确认模式，它指示确认消息的方式。它必须是 **IMPLICIT_ACKNOWLEDGEMENT** 或 **EXPLICIT_ACKNOWLEDGEMENT** 两种方式中的某一种。使用隐式确认时，客户端在收到消息时会进行确认。使用显式确认时，必须在 QAManager 上调用一种确认方法对消息进行确认。

有关确认模式的详细信息，请参见“AcknowledgementMode 类”一节第 386 页。

> 您可以创建 QATransactionalManager 而非 QAManager。请参见“为 C++ 客户端实现事务性消息传递”一节第 69 页。

现在，即可发送消息。

另请参见

- “QAnywhere C++ API 参考”第 385 页

设置 Java 应用程序

使用 QAnywhere Java 客户端发送或接收消息之前，必须完成以下初始化任务。

♦ 初始化 QAnywhere Java API

1. 将 *qaclient.jar* 的位置添加到类路径中。缺省情况下，此文件位于 *install-dir\Java* 下。
2. 导入 ianywhere.qanywhere.client 包。

```
import ianywhere.qanywhere.client.*;
```

3. 创建 QAManager 对象。

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager(null);
```

还可以为 createQAManager 方法指定属性文件来自定义 QAManager 对象：

```
mgr = QAManagerFactory.getInstance().createQAManager("qa_mgr.props.");
```

> **提示**
> 为了提供最大并发数，多线程应用程序应为每个线程创建一个 QAManager。请参见“多线程注意事项”一节第 89 页。

4. 初始化 QAManager 对象。


```
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

open 方法的参数是确认模式，它指示确认消息的方式。它必须为 IMPLICIT_ACKNOWLEDGEMENT 或 EXPLICIT_ACKNOWLEDGEMENT。使用隐式确认时，客户端在收到消息时会进行确认。使用显式确认时，必须在 QAManager 上调用一种确认方法对消息进行确认。

有关确认模式的详细信息，请参见“AcknowledgementMode 接口”一节第 498 页。

> 您可以创建 QATransactionalManager 而非 QAManager。请参见“为 Java 客户端实现事务性消息传递”一节第 70 页。

现在，即可发送消息。

另请参见

- “QAnywhere Java API 参考”第 497 页

设置 SQL 应用程序

在 SQL 中，QAnywhere SQL 允许您执行 QAnywhere .NET API、C++ API 和 Java API 的许多消息传递功能。此功能包括创建消息、设置或获取消息属性和内容、发送和接收消息、触发消息同步，以及设置和获取消息存储库属性。

使用 QAnywhere SQL 生成的消息也可以由使用编程 API 创建的客户端接收。如果已在服务器上配置了一个 JMS 连接器，则消息也可由 JMS 客户端接收。同样，QAnywhere SQL 可用来接收由 QAnywhere .NET、C++、Java API 或 JMS 客户端生成的消息。

QAnywhere SQL 消息传递与用户事务共存。这表示，在提交事务时将提交该连接上的所有 QAnywhere 操作。

请参见“编写 QAnywhere 客户端应用程序”第 53 页。

权限

只有具备 DBA 特权的用户才能自动具备执行 QAnywhere 存储过程的权限。要对某个用户授予权限，具有 DBA 特权的用户必须调用过程 ml_qa_grant_messaging_permissions。

请参见“ml_qa_grant_messaging_permissions”一节第 680 页。

确认模式

QAnywhere SQL API 不支持 IMPLICIT_ACKNOWLEDGEMENT 或 EXPLICIT_ACKNOWLEDGEMENT 模式。通过 SQL API 进行的所有消息传递均是事务性的。

示例

以下示例在库存表中创建一个触发器。当一个项的库存低于特定的阀值时，此触发器将发送一条消息。在提交调用此触发器的事务后，将会发送消息。如果事务被回退，则不发送消息。

```
CREATE TRIGGER inventory_trigger AFTER UPDATE ON inventory
REFERENCING old AS oldinv new AS newinv
```

```
FOR EACH ROW
begin
   DECLARE msgid VARCHAR(128);
   IF oldinv.quantity > newinv.quantity AND newinv.quantity < 10 THEN
      -- Create the message
      SET msgid = ml_qa_createmessage();
      -- Set the message content
      CALL ml_qa_settextcontent( msgid,
         'Inventory of item ' || newinv.itemname
            || ' has fallen to only ' || newinv.quantity );
      -- Make the message high priority
      CALL ml_qa_setpriority( msgid, 9 );
      -- Set a message subject
      CALL ml_qa_setstringproperty( msgid,
         'tm_Subject', 'Inventory low!' );
      -- Send the message to the inventoryManager queue
      CALL ml_qa_putmessage( msgid,
         'inventoryManager' );
   end if;
end
```

另请参见

- “QAnywhere SQL API 参考”第 645 页

QAnywhere 消息地址

一个 QAnywhere 消息地址包括两个部分，即客户端消息存储库 ID 和应用程序队列名：

```
id\queue-name
```

队列名在应用程序内指定，且必须将它告知其它设备上的发送应用程序的实例。有关客户端消息存储库 ID 的信息，请参见“设置客户端消息存储库”一节第 23 页。

每个地址一次最多允许一个应用程序与它关联。多个应用程序运行时关联同一个地址会导致消息检索期间产生未定义行为。

在应用程序中以字符串形式构造地址时，如有必要，请确保对反斜线字符进行转义。请遵循所用的编程语言的字符串转义规则。如果 JMS 目标包含一个反斜线，则必须用另一个反斜线来转义该反斜线。

地址长度不得超过 255 个字符。

系统队列

通知和网络状态更改都作为**系统消息**发送到 QAnywhere 应用程序。系统消息与其它消息相同，只是在名为 **system** 的单独队列中接收。

请参见“系统队列”一节第 64 页。

向 JMS 连接器发送消息

QAnywhere 到 JMS 目标地址有两个部分：

- 连接器地址。这是 ianywhere.connector.address 属性的值。

  请参见“配置 JMS 连接器属性”一节第 157 页。
- JMS 队列名。这是使用 JMS 管理工具创建的队列。

目标地址的形式为：

```
connector-address\JMS-queue-name
```

有关在 JMS 应用程序中指定消息地址的详细信息，请参见：

- “向 JMS 连接器发送 QAnywhere 消息”一节第 158 页
- “为传递到 QAnywhere 的 JMS 消息指定地址”一节第 160 页
- “连接器”第 153 页

系统队列

存在一个名为 **system** 的特殊队列，该队列可接收 QAnywhere 系统消息。有两类消息可发送至系统队列：

- “网络状态通知”一节第 64 页
- “有关推式通知的通知”一节第 65 页

示例

以下 C# 代码可处理系统消息和正常消息，这在您使用按需策略时会非常有用。假定您已经定义了 onMessage() 和 onSystemMessage() 消息处理方法（它们实现处理消息的应用程序逻辑）。

```
// Declare the message listener and system listener.
private QAManager.MessageListener _receiveListener;
private QAManager.MessageListener _systemListener;
...

// Create a MessageListener that uses the appropriate message handlers.
_receiveListener = new QAManager.MessageListener( onMessage );
_systemListener = new QAManager.MessageListener( onSystemMessage );
...

// Register the message handler.
mgr.SetMessageListener( queue-name, _receiveListener );
mgr.SetMessageListener( "system", _systemListener );
```

系统消息处理程序可以查询消息属性以确定消息包含的信息。消息类型属性指示消息是否包含网络状态通知。例如，对于消息 `msg`，您可以执行以下处理过程：

```
msg_type = (MessageType)msg.GetIntProperty( MessageProperties.MSG_TYPE );
if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
  // Process a network status change.
  mgr.TriggerSendReceive( );
} else if ( msg_type == MessageType.PUSH_NOTIFICATION ) {
  // Process a push notification.
  mgr.TriggerSendReceive( );
} else if ( msg_type == MessageType.REGULAR ) {
  // This message type should not be received on the
  // system queue. Take appropriate action here.
}
```

网络状态通知

网络状态发生变化时，NETWORK_STATUS_NOTIFICATION 类型的消息将被发送到系统队列。其到期时间为一分钟。无法更改此到期时间。

当设备进入网络覆盖范围内或处于网络覆盖范围之外时，会有一条消息发送到系统队列，其中包含以下信息：

- **ias_Adapters** 字符串。可用于连接到 MobiLink 服务器的网络适配器列表。此列表用竖线分隔。此属性可以读取但不得进行设置。请参见：
  - .NET： “ADAPTER 字段”一节第 216 页
  - C++： “ADAPTER 变量”一节第 389 页
  - Java： “ADAPTERS 变量”一节第 501 页
- **ias_RASNames** 字符串。可用于连接到 MobiLink 服务器的网络名列表。此列表用竖线分隔。请参见：
  - .NET： “RASNAMES 字段”一节第 221 页
  - C++： “RASNAMES 变量”一节第 393 页
  - Java： “RASNAMES 变量”一节第 504 页
- **ias_NetworkStatus** 整型。网络连接的状态。如果已连接，则值为 1；否则，值为 0。请参见：
  - .NET： “NETWORK_STATUS 字段”一节第 219 页
  - C++： “NETWORK_STATUS 变量”一节第 391 页
  - Java： “NETWORK_STATUS 变量”一节第 503 页

监控网络可用性

可使用网络状态通知来监控网络可用性，并在某个设备进入覆盖范围时执行操作。例如，使用按需策略，并在收到 NETWORK_STATUS_NOTIFICATION 类型的系统队列消息 (ias_NetworkStatus=1) 时，调用 QAManagerBase triggerSendReceive。

另请参见

- “预定义的消息属性”一节第 687 页中的 ias_MessageType
- “系统队列”一节第 63 页

有关推式通知的通知

当收到来自服务器的推式通知时，PUSH_NOTIFICATION 类型的消息将被发送到系统队列。此消息将通知：服务器上有一些排队消息。其到期时间为一分钟。无法更改此到期时间。

在您使用按需策略时，此类系统消息将非常有用。例如，您可以在收到 PUSH_NOTIFICATION 类型的系统队列消息时，调用 QAManagerBase triggerSendReceive。

另请参见

- “使用推式通知进行消息传递的方案”一节第 6 页
- “使用推式通知”一节第 34 页
- “系统队列”一节第 63 页
- “异步接收消息”一节第 75 页
- “预定义的消息属性”一节第 687 页中的 ias_MessageType
- .NET： “MessageProperties 类”一节第 215 页
- C++： “MessageProperties 类”一节第 388 页
- Java： “MessageProperties 接口”一节第 499 页

发送 QAnywhere 消息

以下过程介绍如何从 QAnywhere 应用程序发送消息。这些过程假定您已创建并打开了一个 QAManager 对象。

从您的应用程序发送一条消息，并不能确保该消息是从您的设备传送的。这一操作只是把消息放在一个队列上等待传送。QAnywhere 代理执行将消息发送到 MobiLink 服务器的任务，接着 MobiLink 服务器再将消息传送到消息所要到达的目标。

有关消息传输的发生时间的详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

♦ 发送消息 (.NET)

1. 创建一条新消息。

   您可以分别使用 CreateTextMessage() 或 CreateBinaryMessage()，来创建文本消息或二进制消息。

   ```
   QATextMessage     msg;
   msg = mgr.CreateTextMessage();
   ```

2. 设置消息属性。

   使用 QATextMessage 或 QABinaryMessage 类的方法来设置属性。

   请参见“QAnywhere 消息简介”一节第 14 页。

3. 将消息放入队列，准备发送。

   ```
   mgr.PutMessage( "store-id\\queue-name", msg );
   ```

   其中，*store-id* 和 *queue-name* 是组合构成目标地址的字符串。

   请参见“PutMessage 方法”一节第 283 页和“确定在客户端进行消息传输的时间”一节第 51 页。

♦ 发送消息 (C++)

1. 创建一条新消息。

   您可以分别使用 createTextMessage() 或 createBinaryMessage()，来创建文本消息或二进制消息。

   ```
   QATextMessage *   msg;
   msg = mgr->createTextMessage();
   ```

2. 设置消息属性。

   使用 QATextMessage 或 QABinaryMessage 类的方法来设置消息属性。

   请参见“QAnywhere 消息简介”一节第 14 页。

3. 将消息放入队列，准备发送。

   ```
   if( msg != NULL ) {
     if( !mgr->putMessage( "store-id\\queue-name", msg ) ) {
       // Display error using mgr->getLastErrorMsg().
       }
       mgr->deleteMessage( msg );
   }
   ```


其中，*store-id* 和 *queue-name* 是组合构成目标地址的字符串。

请参见“putMessage 函数”一节第 444 页和“确定在客户端进行消息传输的时间”一节第 51 页。

- **发送消息 (Java)**

1. 创建一条新消息。

   您可以分别使用 QAManagerBase.createTextMessage() 或 QAManagerBase.createBinaryMessage()，来创建文本消息或创建二进制消息。

   ```
   QATextMessage msg;
   msg = mgr.createTextMessage();
   ```

2. 设置消息属性。

   使用 QATextMessage 或 QABinaryMessage 方法来设置消息属性。

   请参见“QAnywhere 消息简介”一节第 14 页。

3. 将消息放入队列。

   ```
   mgr.putMessage("store-id\\queue-name", msg);
   ```

请参见“putMessage 方法”一节第 560 页和“确定在客户端进行消息传输的时间”一节第 51 页。

- **发送消息 (SQL)**

1. 声明保存消息 ID 的变量。

   ```
   begin
       declare @msgid varchar(128);
   ```

2. 创建一条新消息。

   ```
       set @msgid = ml_qa_createmessage();
   ```

3. 设置消息属性。

   有关详细信息，请参见“消息属性”一节第 655 页。

4. 将消息放入队列。

   ```
       call ml_qa_putmessage( @msgid, 'clientid\queuename' );
       commit;
   end
   ```

请参见“ml_qa_putmessage”一节第 681 页和“确定在客户端进行消息传输的时间”一节第 51 页。

实现事务性消息传递

事务性消息传递通过保证组中的所有消息要么全部传送，要么都不传送的方式，来提供对消息进行分组的功能。这通常称为单个**事务**。

实现事务性消息传递时，可创建一个名为 QATransactionalManager 的特殊 QAManagerBase 对象。

有关详细信息，请参见：

- .NET 客户端： “QATransactionalManager 接口”一节第 328 页
- C++ 客户端： “QATransactionalManager 类”一节第 486 页
- Java 客户端： “QATransactionalManager 接口”一节第 601 页
- SQL 客户端：对于 SQL 客户端，所有消息传递均是事务性的，且不需要任何事务性管理器

为 .NET 客户端实现事务性消息传递

♦ 创建事务管理器

1. 初始化 QAnywhere。

   这一步骤与在非事务性消息传递中的步骤相同。

   ```
   using iAnywhere.QAnywhere.Client;
   ```

2. 创建 QATransactionalManager 对象。

   例如，要创建缺省的 QATransactionalManager 对象，可调用 CreateQATransactionalManager（以 null 作为其参数）：

   ```
   QAManager mgr;
   mgr =
     QAManagerFactory.Instance.CreateQATransactionalManager(
     null );
   ```

   请参见“QAManagerFactory 类”一节第 298 页。

   也可以创建使用属性文件自定义的 QATransactionalManager 对象。属性文件在 CreateQATransactionalManager 方法中指定：

   ```
   mgr =
     QAManagerFactory.Instance.CreateQATransactionalManager(
     "qa_mgr.props" );
   ```

   其中，*qa_mgr.props* 是远程设备上属性文件的名称。

3. 初始化 QAManager 对象。

   ```
   mgr.Open();
   ```

现在，即可发送消息。以下过程在一个事务中发送两条消息。

♦ 在一个事务中发送多条消息

1. 初始化消息对象。

   ```
   QATextMessage msg_1;
   QATextMessage msg_2;
   ```

2. 发送消息。

   以下代码在一个事务中发送两条消息：


```
msg_1 = mgr.CreateTextMessage();
if( msg_1 != null ) {
  msg_2 = mgr.CreateTextMessage();
  if( msg_2 != null ) {
    if( !mgr.PutMessage( "jms_1\\queue_name", msg_1 ) ) {
      // Display message using mgr.GetLastErrorMsg().
    } else {
      if( !mgr.PutMessage( "jms_1\\queue_name", msg_2 ) ) {
        // Display message using mgr.GetLastErrorMsg().
      } else {
        mgr.Commit();
      }
    }
  }
}
```

Commit 方法可提交当前事务并开始一个新事务。此方法提交全部 PutMessage() 方法和 GetMessage() 方法调用。

注意
第一个事务以对 open 方法的调用开始。

另请参见

- “QATransactionalManager 接口”一节第 328 页

为 C++ 客户端实现事务性消息传递

♦ **创建事务管理器**

1. 初始化 QAnywhere。

   这一步骤与在非事务性消息传递中的步骤相同。

   ```
   #include <qa.hpp>
   QAManagerFactory *   factory;

   factory = QAnywhereFactory_init();
   if( factory == NULL ) {
       // Fatal error.
   }
   ```

2. 创建事务性管理器。

   ```
   QATransactionalManager *  mgr;
   mgr = factory->createQATransactionalManager( NULL );
   if( mgr == NULL ) {
       // Fatal error.
   }
   ```

   与非事务管理器相同，可以指定属性文件来自定义 QAnywhere 行为。在本例中，没有使用任何属性文件。

3. 初始化管理器。

```
if( !mgr->open() ) {
   // Display message using mgr->getLastErrorMsg().
}
```

现在，即可发送消息。以下过程在一个事务中发送两条消息。

♦ 在一个事务中发送多条消息

1. 初始化消息对象。

   ```
   QATextMessage *  msg_1;
   QATextMessage *  msg_2;
   ```

2. 发送消息。

   以下代码在一个事务中发送两条消息：

   ```
   msg_1 = mgr->createTextMessage();
   if( msg_1 != NULL ) {
     msg_2 = mgr->createTextMessage();
     if( msg_2 != NULL ) {
       if( !mgr->putMessage( "jms_1\\queue_name", msg_1 ) ) {
         // Display message using mgr->getLastErrorMsg().
       } else {
         if( !mgr->putMessage( "jms_1\\queue_name", msg_2 ) ) {
           // Display message using mgr->getLastErrorMsg().
         } else {
           mgr->commit();
         }
       }
       mgr->deleteMessage( msg_2 );
     }
     mgr->deleteMessage( msg_1 );
   }
   ```

   commit 方法可提交当前事务并开始一个新事务。此方法提交全部 putMessage() 方法和 getMessage() 方法调用。

注意
第一个事务以对 open 方法的调用开始。

另请参见

- C++： “QATransactionalManager 类”一节第 486 页
- Java： “QATransactionalManager 接口”一节第 601 页

为 Java 客户端实现事务性消息传递

♦ 创建事务管理器

1. 初始化 QAnywhere。

   这一步骤与在非事务性消息传递中的步骤相同。

   ```
   import ianywhere.qanywhere.client;
   QAManagerFactory factory = new QAManagerFactory();
   ```


请参见“QAManagerFactory 类”一节第 298 页。

2. 创建 QATransactionalManager 对象。

   例如，要创建缺省的 QATransactionalManager 对象，可调用 createQATransactionalManager（以 null 作为其参数）：

```
QAManager mgr;
mgr = factory.createQATransactionalManager( null );
```

   也可以创建使用属性文件自定义的 QATransactionalManager 对象。属性文件在 createQATransactionalManager 方法中指定：

```
mgr = factory.createQATransactionalManager( "qa_mgr.props" );
```

   其中，*qa_mgr.props* 是远程设备上属性文件的名称。

3. 初始化 QAManager 对象。

```
mgr.open();
```

现在，即可发送消息。以下过程在一个事务中发送两条消息。

♦ 在一个事务中发送多条消息

1. 初始化消息对象。

```
QATextMessage msg_1;
QATextMessage msg_2;
```

2. 发送消息。

   以下代码在一个事务中发送两条消息：

```
msg_1 = mgr.createTextMessage();
if( msg_1 != null ) {
  msg_2 = mgr.createTextMessage();
  if( msg_2 != null ) {
    if( !mgr.putMessage( "jms_1\\queue_name", msg_1 ) ) {
      // Display message using mgr.getLastErrorMsg().
    } else {
      if( !mgr.putMessage( "jms_1\\queue_name", msg_2 ) ) {
        // Display message using mgr.getLastErrorMsg().
      } else {
        mgr.commit();
      }
    }
  }
}
```

   commit 方法可提交当前事务并开始一个新事务。此方法提交全部 putMessage() 方法和 getMessage() 方法调用。

> **注意**
> 第一个事务以对 open 方法的调用开始。

取消 QAnywhere 消息

如果要取消一条 QAnywhere 消息，可在传输前将其置于已取消状态。当使用 QAnywhere 代理的缺省删除规则时，已取消的消息最终会从消息存储库中删除。如果 QAnywhere 消息已经处于最终状态，或已经传输到中央消息传递服务器，则无法取消该消息。

以下过程介绍如何取消 QAnywhere 消息。

注意
无法使用 QAnywhere SQL API 取消消息。

♦ 取消消息 (.NET)

1. 获取要取消的消息的 ID。

```
// msg is a QAMessage instance that has not been
// transmitted.
string msgID = msg.getMessageID();
```

2. 使用要取消的消息的 ID 来调用 CancelMessage。

```
mgr.CancelMessage(msgID);
```

请参见“CancelMessage 方法”一节第 265 页。

♦ 取消消息 (C++)

1. 获取要取消的消息的 ID。

```
// msg is a QAMessage instance that has not been
// transmitted.
qa_string msgID = msg->getMessageID();
```

2. 使用要取消的消息的 ID 来调用 cancelMessage。

```
bool result = mgr->cancelMessage(msgID);
```

请参见“cancelMessage 函数”一节第 431 页。

♦ 取消消息 (Java)

1. 获取要取消的消息的 ID。

```
// msg is a QAMessage instance that has not been
// transmitted.
String msgID = msg.getMessageID();
```

2. 使用要取消的消息的 ID 来调用 cancelMessage。

```
boolean result = mgr.cancelMessage(msgID);
```

请参见“cancelMessage 方法”一节第 545 页。

接收 QAnywhere 消息

以下主题介绍如何接收 QAnywhere 消息。

同步接收消息

为同步接收消息，应用程序需显式轮询队列以检查是否存在消息。可定期轮询队列，也可在用户启动某个特定操作（例如单击 [刷新] 按钮）时轮询队列。

♦ 同步接收消息 (.NET)

1. 声明消息对象以保存进来的消息。

```
QAMessage msg;
QATextMessage text_msg;
```

2. 轮询消息队列，收集消息：

```
for(;;) {
    msg = mgr.GetMessageNoWait("queue-name");
    if( msg == null ) {
      break;
    }
    addMessage( msg );
  }
```

请参见“GetMessageNoWait 方法”一节第 275 页。

♦ 同步接收消息 (C++)

1. 声明消息对象以保存进来的消息。

```
QAMessage * msg;
QATextMessage * text_msg;
```

2. 轮询消息队列，收集消息：

```
for( ;; ) {
    msg = mgr->getMessageNoWait( "queue-name" );
    if( msg == NULL ) {
      break;
    }
    addMessage(msg);
  }
```

请参见“getMessageNoWait 函数”一节第 441 页。

♦ 同步接收消息 (Java)

1. 声明消息对象以保存进来的消息。

```
QAMessage msg;
QATextMessage text_message;
```

2. 轮询消息队列，收集消息：

```
if(mgr.start()) {
  for ( ;; ) {
    msg = mgr.getMessageNoWait("queue-name");
    if ( msg == null ) {
      break;
    }
    addMessage(msg);
  }
  mgr.stop();
}
```

请参见“getMessageNoWait 方法”一节第 554 页。

♦ 同步接收消息 (SQL)

1. 声明保存消息 ID 的对象。

```
begin
    declare @msgid varchar(128);
```

2. 轮询消息队列，收集消息。

```
    loop
        set @msgid = ml_qa_getmessagenowait( 'myaddress' );
        if @msgid is null then leave end if;
       message 'a message with content ' || ml_qa_gettextcontent( @msgid )
|| ' has been received';
    end loop;
    commit;
end
```

请参见：

- “ml_qa_getmessagenowait”一节第 677 页
- “ml_qa_getmessagetimeout”一节第 678 页
- “ml_qa_getmessage”一节第 676 页

异步接收消息

要使用 .NET、C++ 和 Java API 异步接收消息，可编写并注册一个消息监听器函数，当队列中出现消息时，由 QAnywhere 调用此函数。消息监听器将进来的消息作为一个参数。消息监听器执行的任务取决于您的应用程序。例如，在 TestMessage 示例应用程序中，消息监听器将消息添加到 TestMessage 主窗口的消息列表中。

.NET、C++ 和 Java 的开发提示

为防止在处理已接收的消息及正在确认的消息的过程中中途发生应用程序错误，在 EXPLICIT_ACKNOWLEDGEMENT 模式中使用 QAManagers 会更加安全一些。

如果 QAManager 是在 EXPLICIT_ACKNOWLEDGEMENT 模式中打开的，则只有在消息被成功处理后，才能采用 onMessage 方法对其进行确认。这样，如果在处理消息时出现错误，则会因为消息无法确认而再次对其进行接收。

如果 QAManager 是在 IMPLICIT_ACKNOWLEDGEMENT 模式中打开的，则传递到 onMessage 的消息会在 onMessage 返回时被隐式确认。如果在处理消息时，用户应用程序遇到错误，则消息将会被确认，但决不会被再次接收。

♦ 异步接收消息 (.NET)

1. 实现消息处理程序方法。

```
private void onMessage(QAMessage msg) {
    // Process message.
}
```

2. 注册消息处理程序。

   要注册消息处理程序，请创建一个 QAManager.MessageListener 对象，该对象将消息处理程序函数作为其参数。然后使用 QAManager.SetMessageListener 函数在特定队列中注册该 MessageListener。在以下示例中，*queue-name* 是一个字符串，该字符串是 QAManager 对象监听的队列的名称。

```
MessageListener listener;
listener = new MessageListener( onMessage );
mgr.SetMessageListener( "queue-name", listener );
```

   请参见“MessageListener 委派”一节第 214 页和“SetMessageListener 方法”一节第 289 页。

♦ 异步接收消息 (C++)

1. 创建一个实现 QAMessageListener 接口的类。

```
class MyClass: public QAMessageListener {
   public:
   void onMessage( QAMessage * Msg);
};
```

   请参见“QAMessageListener 类”一节第 480 页。

2. 实现 onMessage 方法。

   QAMessageListener 接口包含 onMessage 方法。每当有消息到达队列时，QAnywhere 库便会将新消息作为一个参数传递给此方法对其进行调用。

```
void MyClass::onMessage(QAMessage * msg) {
     // Process message.
}
```

3. 注册消息监听器。

```
my_listener = new MyClass();
mgr->setMessageListener( "queue-name", my_listener );
```

   请参见“setMessageListener 函数”一节第 449 页。

♦ 异步接收消息 (Java)

1. 实现消息处理程序方法和异常处理程序方法。

```
class MyClass implements QAMessageListener {
  public void onMessage(QAMessage message) {
```



```
      // Process the message.
    }
    public void onException(
      QAException exception, QAMessage message) {
      // Handle the exception.
    }
  }
```

2. 注册消息处理程序。

```
MyClass listener = new MyClass();
mgr.setMessageListener("queue-name", listener);
```

请参见“QAMessageListener 接口”一节第 593 页和“setMessageListener 方法”一节第 564 页。

♦ **异步接收消息 (SQL)**

- 创建一个名称为 **ml_qa_listener_*queue*** 的存储过程，其中 *queue* 是消息队列名。

  只要消息在给定队列中排队，就会调用此过程。

  请参见“ml_qa_listener_queue”一节第 680 页。

使用选择程序接收消息

可以使用**消息选择程序**来选择要接收的消息。消息选择程序是一个类似于 SQL 的表达式，可指定接收操作所要使用的消息子集的选择条件。

消息选择程序的语法和语义与传输规则的条件部分完全相同。

请参见“条件语法”一节第 763 页。

示例

以下 C# 示例从 receiveQueue 获取下一条消息（具有一个值为 1 的消息属性 intprop）。

```
msg = receiver.GetMessageBySelectorNoWait(
   receiveQueue, "intprop=1" );
```

以下 C++ 示例从 receiveQueue 获取下一条消息（具有一个值为 1 的消息属性 intprop）。

```
msg = receiver->getMessageBySelectorNoWait(
  receiveQueue, "intprop=1" );
```

以下 Java 示例从 receiveQueue 获取下一条消息（具有一个值为 1 的消息属性 intprop）。

```
msg = receiver.getMessageBySelectorNoWait(
  receiveQueue, "intprop=1");
```

另请参见

- .NET：“GetMessageBySelector 方法”一节第 272 页 和 “GetMessageBySelectorNoWait 方法”一节第 273 页
- C++：“getMessageBySelector 函数”一节第 439 页 和 “getMessageBySelectorNoWait 函数”一节第 440 页
- Java：“getMessageBySelector 方法”一节第 552 页 和 “getMessageBySelectorNoWait 方法”一节第 552 页
- SQL：SQL API 不支持使用选择程序接收消息

读取非常大的消息

有时消息过大，超出 QAManager 属性 MAX_IN_MEMORY_MESSAGE_SIZE 设置的限制，或超出其缺省值（在 Windows 上为 1MB，在 Windows Mobile 上为 64KB）的限制。在这种情况下，内存中的消息对象无法包含此消息的完整内容，因此一些依赖于载入内存中的消息的完整内容的方法（如 readInt() 和 readString()）便无法使用。但是，您可以直接从消息存储库中逐段读取非常大的消息。为此，请循环使用 QATextMessage.readText() 或 QABinaryMessage.readBinary()。

有关详细信息，请参见：

- .NET：“ReadBinary 方法”一节第 229 页 和 “ReadText 方法”一节第 327 页
- C++：“readBinary 函数”一节第 400 页 和 “readText 函数”一节第 484 页
- Java：“readBinary 方法”一节第 513 页 和 “readText 方法”一节第 598 页
- SQL：SQL API 不支持接收非常大的消息

这样做时，不能使用以 IMPLICIT_ACKNOWLEDGEMENT 打开的 QAManager。必须使用以 EXPLICIT_ACKNOWLEDGEMENT 打开的 QAManager，并且必须在确认该消息之前完成全部 readText() 或 readBinary() 调用。

请参见“确认模式”一节第 61 页。

浏览 QAnywhere 消息

可浏览进来队列和外发队列中的消息。浏览操作不会影响消息的状态。

有关消息状态的详细信息，请参见“预定义的消息属性”一节第 687 页中的 ias_Status。

以下主题介绍如何浏览 QAnywhere 消息。

浏览所有消息

可通过调用适当的 browseMessages() 方法，来浏览所有队列中的消息。

以下 .NET 示例使用 QAManager.BrowseMessages() 方法来浏览所有队列：

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessages();
while( msgs.MoveNext() ) {
      msg = (QAMessage)msgs.Current;
      // Process message.
}
```

以下 C++ 示例使用 QAManager browseMessages 函数来浏览所有队列：

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessages();
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
  mgr->browseClose( bh );
}
```

以下 Java 示例使用 QAManager.browseMessages 方法来浏览所有队列：

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessages();
while( msgs.hasMoreElements() ) {
    msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

另请参见

- .NET： “BrowseMessages 方法”一节第 262 页
- C++： “browseMessages 函数”一节第 428 页
- Java： “browseMessages 方法”一节第 543 页
- SQL：SQL API 不支持浏览消息

浏览队列中的消息

可通过向适当的 browseMessagesByQueue() 方法提供队列名，浏览给定队列中的消息。

以下 .NET 示例使用 QAManager.BrowseMessagesByQueue 方法来浏览队列：


```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByQueue( "q1" );
while( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

以下 C++ 示例使用 QAManager browseMessagesByQueue 函数来浏览队列：

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesByQueue( _T("q1") );
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

以下 Java 示例使用 QAManager.browseMessagesByQueue 方法来浏览队列：

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesByQueue( "q1" );
while( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

另请参见

- .NET： “BrowseMessagesByQueue 方法”一节第 264 页
- C++： “browseMessagesByQueue 函数”一节第 429 页
- Java： “browseMessagesByQueue 方法”一节第 544 页
- SQL：SQL API 不支持浏览消息

按 ID 浏览消息

可通过为 browseMessagesbyID() 方法指定某特定消息的 ID，来浏览该消息。

以下 .NET 示例使用 QAManager.BrowseMessageByID 方法来浏览消息：

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByID( "ID:123" );
if( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

以下 C++ 示例使用 QAManager browseMessageByID 函数来浏览消息：

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesByID( _T( "ID:123" ) );
msg = mgr->browseNextMessage( bh );
if( msg != qa_null ) {
  // Process message.
}
mgr->browseClose( bh );
```


以下 Java 示例使用 QAManager.browseMessageByID 方法来浏览消息：

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesByID( "ID:123" );
if( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

另请参见

- .NET： “BrowseMessagesByID 方法”一节第 263 页
- C++： “browseMessagesByID 函数”一节第 428 页
- Java： “browseMessagesByID 方法”一节第 543 页
- SQL：SQL API 不支持浏览消息

使用选择程序浏览消息

可以使用**消息选择程序**来选择要浏览的消息。消息选择程序是一个类似于 SQL 的表达式，可指定浏览操作所要使用的消息子集的选择条件。

消息选择程序的语法和语义与传输规则的条件部分完全相同。

请参见“条件语法”一节第 763 页。

以下 .NET 示例浏览消息存储库中具有值为 1 的属性 intprop 的所有消息。

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesBySelector( "intprop = 1" );
while( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

以下 C++ 示例浏览消息存储库中具有值为 1 的属性 intprop 的所有消息。

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesBySelector( _T("intprop = 1") );
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

以下 Java 示例浏览消息存储库中具有值为 1 的属性 intprop 的所有消息。

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesBySelector( "intprop = 1" );
while( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```


另请参见

- .NET： “BrowseMessagesBySelector 方法”一节第 265 页
- C++： “browseMessagesBySelector 函数”一节第 430 页
- Java： “browseMessagesBySelector 方法”一节第 545 页
- SQL：SQL API 不支持浏览消息

处理 QAnywhere 异常

QAnywhere C++ API、Java API 和 .NET API 包括一些用来处理异常的特殊对象和属性。

.NET 异常

QAException 类可封装 QAnywhere 客户端应用程序异常。在捕获一个 QAnywhere 异常后，可使用 QAException ErrorCode 和 Message 属性来确定错误代码和错误消息。

请注意，如果在消息监听器委派内抛出一个 QAException，并且该 QAException 未在消息监听器中被捕获，则会将其记录到 QAManager 日志文件中。因为仅记录未捕获的 QAException，因此建议在消息监听器委派内处理所有异常，或由异常监听器委派处理所有异常，以便能够对这些异常进行适当的处理。

有关消息监听器委派和异常监听器委派的详细信息，请参见：

- “MessageListener 委派”一节第 214 页
- “MessageListener2 委派”一节第 214 页
- “ExceptionListener 委派”一节第 213 页
- “ExceptionListener2 委派”一节第 213 页

有关日志文件的详细信息，请参见“QAnywhere Manager 配置属性”一节第 90 页。

抛出一个 QAException 时，当前事务将被回退。当具有 QATransactionalManager 的消息监听器中发生这种情况时，抛出 QAException 时正在处理的消息会被放回接收队列，以使该消息可被重新接收。可使用消息存储库属性 ias_MaxDeliveryAttempts 来防止发生无限循环。

当属性 ias_MaxDeliveryAttempts 被 QAnywhere 应用程序设置为正整数 *n* 时，如同在 `mgr.SetIntStoreProperty( "ias_MaxDeliveryAttempts", 5 )` 中一样，QAnywhere 客户端将会在将此消息的状态设置为无法接收之前，最多尝试接收这条未确认的消息 *n* 次。如果未设置属性 ias_MaxDeliveryAttempts 或该属性为负值，则 QAnywhere 客户端尝试接收消息的次数将不受限制。

有关详细信息，请参见：

- “QAException 类”一节第 243 页
- “ErrorCode 属性”一节第 252 页
- “预定义的客户端消息存储库属性”一节第 746 页

C++ 异常

对于 C++，QAError 类可封装 QAnywhere 客户端应用程序异常。可使用 QAManagerBase::getLastError() 方法或 QAManagerFactory::getLastError() 方法，来确定与上一个执行方法相关的错误代码。可使用相应的 getLastErrorMessage() 方法来获得错误文本。

有关错误代码的列表及详细信息，请参见“QAError 类”一节第 411 页。

有关 getLastError 和 getLastErrorMessage 的详细信息，请参见：

- QAManagerBase: “getLastError 函数”一节第 437 页和 “getLastErrorMsg 函数”一节第 437 页
- QAManagerFactory: “getLastError 函数”一节第 456 页和 “getLastErrorMsg 函数”一节第 457 页

Java 异常

QAException 类可封装 QAnywhere 客户端应用程序异常。在捕获一个 QAnywhere 异常后，可使用 QAException ErrorCode 和 Message 属性来确定错误代码和错误消息。

如果在消息监听器内抛出一个 QAException，并且该 QAException 未在消息监听器中被捕获，则会将其记录到 QAManager 日志文件中。因为仅记录未捕获的 QAException，因此建议在消息监听器内处理所有异常，或由异常监听器处理所有异常，以便能够对这些异常进行适当的处理。

有关消息监听器和异常监听器的详细信息，请参见：

- “QAMessageListener 接口”一节第 593 页
- “QAMessageListener2 接口”一节第 594 页
- “QAException 类”一节第 525 页

有关日志文件的详细信息，请参见“QAnywhere Manager 配置属性”一节第 90 页。

抛出一个 QAException 时，当前事务将被回退。当具有 QATransactionalManager 的消息监听器中发生这种情况时，抛出 QAException 时正在处理的消息会被放回接收队列，以使该消息可被重新接收。可使用消息存储库属性 ias_MaxDeliveryAttempts 来防止发生无限循环。

当属性 ias_MaxDeliveryAttempts 被 QAnywhere 应用程序设置为正整数 *n* 时，如同在 `mgr.SetIntStoreProperty( "ias_MaxDeliveryAttempts", 5 )` 中一样，QAnywhere 客户端将会在将此消息的状态设置为无法接收之前，最多尝试接收这条未确认的消息 *n* 次。如果未设置属性 ias_MaxDeliveryAttempts 或该属性为负值，则 QAnywhere 客户端尝试接收消息的次数将不受限制。

有关详细信息，请参见：

- “ErrorCode 属性”一节第 252 页
- “预定义的客户端消息存储库属性”一节第 746 页

错误代码

下表列出 QAnywhere 错误代码值：

错误值	说明
0	无错误。
1000	初始化错误。
1001	终止错误。
1002	无法访问客户端属性文件。

错误值	说明
1003	没有目标。
1004	未实现函数。
1005	不能写入到消息，因为它处于只读模式。
1006	在客户端消息存储库中存储消息时出错。
1007	从客户端消息存储库中检索消息时出错。
1008	初始化后台线程时出错。
1009	打开消息存储库连接时出错。
1010	客户端属性文件中有一个无效的属性。
1011	打开日志文件时出错。
1012	遇到了意外的消息结尾。
1013	消息存储库过大，设备上可用磁盘空间不足。
1014	还没有为进行消息传递而初始化消息存储库。
1015	取得队列深度时出错。
1016	在未设置消息存储库 ID 时，无法使用 QAManagerBase.getQueueDepth。
1017	当过滤器为 ALL 时，无法在给定目标上使用 QAManagerBase.getQueueDepth。
1018	取消消息时出错。
1019	取消消息时出错。不能取消已发送的消息。
1020	确认消息时出错。
1021	QAManager 没有打开。
1022	QAManager 已打开。
1023	给定选择程序存在一个语法错误。
1024	时间戳在可接受范围以外。
1025	无法打开 QAManager，因为并发服务器请求的最大数不够大（参见数据库“-gn 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》）。
1026	从消息存储库中检索属性时出错。

错误值	说明
1027	将属性存储到消息存储库中时出错。

关闭 QAnywhere

完成发送和接收消息后，您可以通过完成以下过程之一来关闭 QAnywhere 消息传递系统。

♦ 关闭 QAnywhere (.NET)

- 停止并关闭 QAnywhere Manager。

```
mgr.Stop();
mgr.Close();
```

♦ 关闭 QAnywhere (C++)

1. 关闭 QAnywhere Manager。

```
mgr->stop();
mgr->close();
```

2. 终止工厂。

```
QAnywhereFactory_term();
```

此步骤将关闭应用程序的消息传递部分。

♦ 关闭 QAnywhere (Java)

- 停止并关闭 QAnywhere Manager。

```
mgr.stop();
mgr.close();
```

另请参见

- .NET： “Stop 方法”一节第 297 页
- C++： “stop 函数”一节第 452 页
- Java： “stop 方法”一节第 570 页
- SQL：SQL API 不支持关闭 QAnywhere

多线程注意事项

对 QAManager 的访问被序列化。当对单个 QAManager 进行多线程访问时，如果有一个线程在 QAManager 上执行方法调用，则线程将会阻塞。要实现最大化并发性，请对每个线程使用不同的 QAManager。一次仅允许一个线程对一个 QAManager 实例进行访问。其它线程将会阻塞，直到第一个线程所调用的 QAManager 方法返回时为止。

QAnywhere Manager 配置属性

您可以通过以下方式之一来设置 QAnywhere Manager 配置属性：

- 创建属性文本文件来定义 QAnywhere Manager 配置属性，以供一个 Manager 实例使用。

  请参见“在文件中设置 QAnywhere Manager 配置属性”一节第 91 页。
- 以编程方式设置 QAnywhere Manager 配置属性。

  请参见“以编程方式设置 QAnywhere Manager 配置属性”一节第 92 页。

以下是 QAnywhere Manager 配置属性：

- **COMPRESSION_LEVEL=n** 设置压缩级别。

  n 是压缩因子，以 0 到 9 之间的整数表示，其中 0 表示无压缩，9 表示最大压缩。
- **CONNECT_PARAMS=connect-string** 指定 QAnywhere Manager 用来连接到消息存储库数据库的连接字符串。以 *keyword=value* 形式指定每个连接选项，用分号分隔多个选项。

  独立客户端不支持此属性。

  缺省值为 `"eng=qanywhere;uid=ml_qa_user;pwd=qanywhere"`

  有关选项的列表，请参见“连接参数”一节 《SQL Anywhere 服务器 - 数据库管理》。

  有关管理数据库用户和口令的信息，请参见“编写安全的消息传递应用程序”第 131 页。
- **DATABASE_TYPE=string** 指定 QAnywhere Manager 要连接到的数据库的类型。对于 SQL 数据库使用 **sqlanywhere**，而对于 UltraLite 数据库使用 **ultralite**。缺省情况下，Manager 使用 **sqlanywhere**。
- **LOG_FILE=filename** 指定用来写入记录消息的文件的名称。隐式指定此选项可启用记录。
- **MAX_IN_MEMORY_MESSAGE_SIZE=n** 读取一条消息时，*n* 是将为其分配缓冲区的最大消息的大小（以字节为单位）。大于 *n* 个字节的消息必须使用流式操作读取。Windows 上的缺省值为 1MB，Windows Mobile 上为 64KB。

以下属性是专为独立客户端制定的属性：

- **ML_PROTOCOL_TYPE** 指定协议类型。有效选项是 **tcpip**、**tls**、**http** 或 **https**。
- **ML_PROTOCOL_PARAMS** 指定 MobiLink 连接参数。
- **ML_PROTOCOL_USENAME** 执行效果与 QAnywhere 代理中的 -mu 选项相同。
- **ML_PROTOCOL_PASSWORD** 执行效果与 QAnywhere 代理中的 -mn 选项相同。
- **INC_UPLOAD** 执行效果与 QAnywhere 代理中的 -iu 选项相同。
- **INC_DOWNLOAD** 执行效果与 QAnywhere 代理中的 -idl 选项相同。
- **STORE_ID** 执行效果与 QAnywhere 代理中的 -id 选项相同。
- **STORE_ENCRYPTION_KEY** 指定用于加密 MessageStore 的加密密钥。

- **POLICY** 执行效果与 QAnywhere 代理中的 -policy 选项相同。
- **DELETE_PERIOD** 指定执行删除规则之间所间隔的秒数。如果指定的量是负数，则禁止执行删除规则。
- **PUSH** 执行效果与 QAnywhere 代理中的 -push 选项相同。

在文件中设置 QAnywhere Manager 配置属性

> **注意**
> 在 Sybase Central 中，可以创建或打开 QAnywhere Manager 配置文件。从 QAnywhere 插件任务窗格中，选择 [**创建代理配置文件**]。选择文件名和位置之后，将打开配置文件的 [**属性**] 窗口，在该对话框中可设置属性。

QAManager Manager 属性文件中的信息是特定于 QAManager 的一个实例的。

必须为具有应用程序的每个部署副本的远程设备配置并安装了属性文件，才能使用该属性文件。

有关指定属性文件名的信息，请参见：

- .NET API： “CreateQAManager 方法”一节第 300 页
- C++ API： “createQAManager 函数”一节第 454 页
- Java API： “createQAManager 方法”一节第 572 页
- SQL API：无法使用 QAnywhere SQL API 在文件中设置属性。请参见“以编程方式设置 QAnywhere Manager 配置属性”一节第 92 页。

如果该属性文件与客户端可执行文件不在同一目录，则还必须指定其绝对路径。如果要使用这些属性的缺省设置，请使用空值而非文件名。

文件中所设置的值允许您启用或禁用一些 QAnywhere 功能，例如自动消息压缩和记录。

QAnywhere Manager 配置属性文件中的条目采用 *name=value* 的形式。有关属性名的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。如果 *value* 有空格，则用双引号将其引起来。注释行以 # 开头。例如：

```
# contents of QAnywhere manager configuration properties file
LOG_FILE=.\sender.ini.txt
# A comment
CONNECT_PARAMS=eng=qanywhere;uid=ml_qa_user;pwd=qanywhere
DATABASE_TYPE=sqlanywhere
MAX_IN_MEMORY_MESSAGE_SIZE=2048
COMPRESSION_LEVEL=0
```

引用配置文件

假定有一个名为 *mymanager.props* 的 QAnywhere Manager 配置属性文件，内容如下：

```
COMPRESSION_LEVEL=9
CONNECT_PARMS=DBF=mystore.db
```

创建 QAManager 时，通过名称引用该文件。

下面是使用 C# 编写的一个示例：

```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( "mymanager.props" );
mgr.Open( AcknowledgeMode.EXPLICIT_ACKNOWLEDGEMENT );
```

有关 .NET API 的信息，请参见“QAManager 接口”一节第 253 页和“QAManagerFactory 类”一节第 298 页。

下面是使用 C++ 编写的一个示例：

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( "mymanager.props" );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

有关 C++ API 的信息，请参见“QAManager 类”一节第 420 页和“QAManagerFactory 类”一节第 454 页。

下面是使用 Java 编写的一个示例：

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager("mymanager.props");
mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
```

有关 Java API 的信息，请参见“QAManagerFactory 类”一节第 571 页和“QAManager 接口”一节第 535 页。

以编程方式设置 QAnywhere Manager 配置属性

在 QAnywhere API 中，可以使用 QAManagerBase setproperty 方法以编程方式设置属性。以编程方式设置 QAnywhere Manager 配置属性，必须在调用 QAManager 实例的 open 方法之前完成。

有关 QAManager 属性的详细信息，请参见“QAnywhere Manager 配置属性”一节第 90 页。

示例

以下 C# 示例以编程方式设置属性。创建 QAManager 时，指定这些属性设置。

```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( null );
mgr.SetProperty( "COMPRESSION_LEVEL", "9" );
mgr.SetProperty( "CONNECT_PARAMS", "DBF=mystore.db" );
mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.Open( AcknowledgeMode.EXPLICIT_ACKNOWLEDGEMENT );
```

有关 .NET API 的信息，请参见“QAManager 接口”一节第 253 页和“QAManagerFactory 类”一节第 298 页。

以下 C++ 示例以编程方式设置属性。创建 QAManager 时，指定这些属性设置。

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( NULL );
mgr->setProperty( "COMPRESSION_LEVEL", "9" );
mgr->setProperty( "CONNECT_PARAMS", "DBF=mystore.db" );
mgr->setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```


有关 C++ API 的信息，请参见“QAManager 类”一节第 420 页和“QAManagerFactory 类”一节第 454 页。

以下 Java 示例以编程方式设置属性。创建 QAManager 时，指定这些属性设置。

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager(null);
mgr.setProperty("COMPRESSION_LEVEL", 9);
mgr.setStringProperty("CONNECT_PARMS", "DBF=mystore.db");
mgr.setStringProperty("DATABASE_TYPE", "sqlanywhere");
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

有关 Java API 的信息，请参见“QAManagerFactory 类”一节第 571 页和“QAManager 接口”一节第 535 页。

QAnywhere 独立客户端

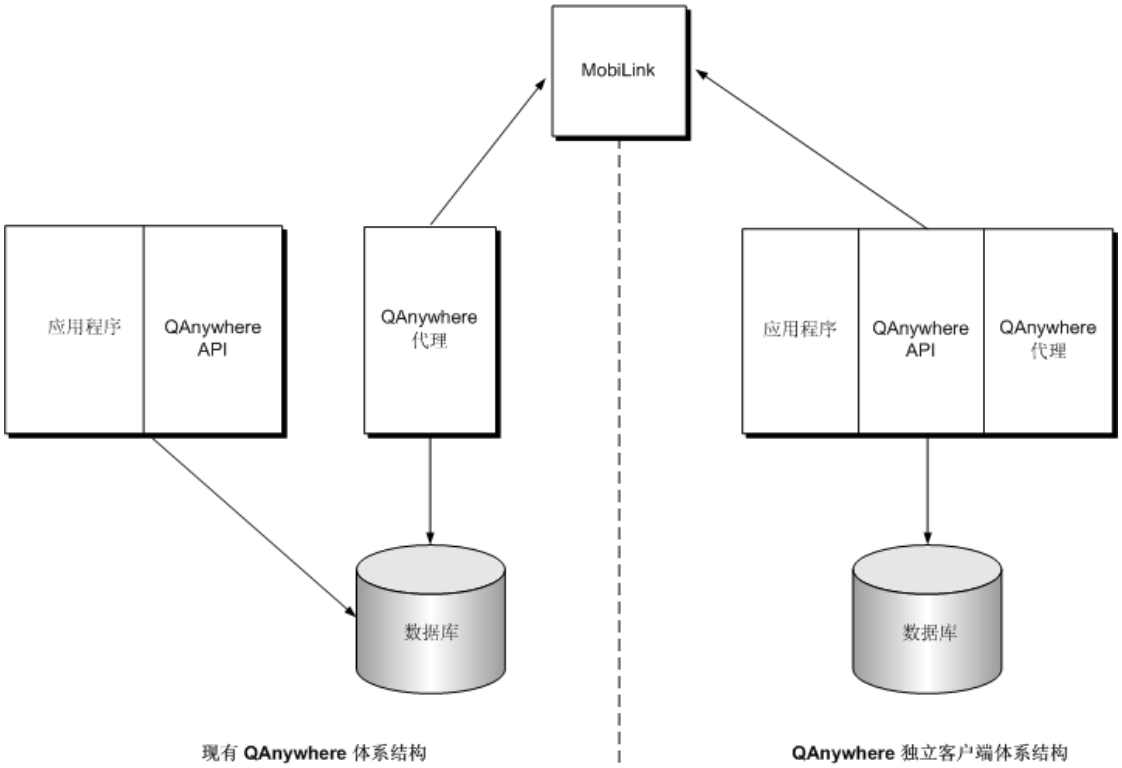
目录

QAnywhere 独立客户端简介

QAnywhere 独立客户端提供了一个精简的客户端，该客户端允许您设置消息传递系统而不必为运行 QAnywhere 代理或管理数据库担扰。独立客户端将客户端存储库管理和 QAnywhere 代理功能并入到访问客户端 API 的同一个进程中，这样用户就无须创建或维护客户端消息存储库且不需要以单独的进程运行 QAnywhere 代理。

下图显示了独立客户端体系结构和现有 QAnywhere 体系结构。

现有 **QAnywhere** 体系结构与 **QAnywhere** 独立客户端体系结构的比较



理解独立客户端消息存储库

消息存储库是绑定到存储库 ID 的 UltraLite 数据库，由独立客户端自动创建和维护。QAnywhere API 使用进程中 UltraLite 运行时访问消息存储库，而不使用 UltraLite 引擎。

独立客户端消息存储库与现有 QAnywhere 消息存储库的不同之处在于，一次只能有一个用户应用程序访问消息存储库。即，同一设备上的多个独立客户端应用程序需要单独的消息存储库和唯一的存储库 ID。因此，独立客户端没有 "本地消息传递" 的概念，由此，消息可以发送到同一消息存储库中的不同队列。通过独立客户端发送的所有消息都假定用于不同的消息存储库。

由于即使在客户端应用程序未运行时消息也可能存在于消息存储库中，所以可以在初次使用时通过提供 STORE_ENCRYPTION_KEY 值来加密存储库。以后每次用户尝试使用该消息存储库时都必须提供相同的密钥。该加密密钥也能加密磁盘上的数据。请参见“QAnywhere Manager 配置属性”一节第 90 页。

部署独立客户端

QAnywhere 独立客户端分布在单独的 dll 文件（用于 .NET）和单独的 JAR 文件（用于 Java）中：

- .NET 实现分布在 dll *iAnywhere.QAnywhere.StandAloneClient.dll* 中并使用命名空间 iAnywhere.QAnywhere.StandAloneClient。请参见“用于客户端的 QAnywhere .NET (.NET 2.0)”一节第 212 页。
- Java 实现分布在 JAR 文件 *qastandaloneclient.jar* 中并使用包名称 ianywhere.qanywhere.standaloneclient。请参见“用于客户端的 QAnywhere Java API”一节第 498 页。

独立客户端 API

独立客户端的 Java 和 .NET 实现都保存与现有客户端相同的 API，但存在以下例外。

以下 QAManager 配置属性专用于独立客户端：

- **ML_PROTOCOL_TYPE** 指定协议类型。有效选项为 tcpip、tls、http 或 https。
- **ML_PROTOCOL_PARAMS** 指定 MobiLink 连接参数。
- **ML_PROTOCOL_USENAME** 指定 MobiLink 用户名。此选项与 QAnywhere 代理 -mu 选项相同。请参见“-mu 选项”一节第 712 页。
- **ML_PROTOCOL_PASSWORD** 为 MobiLink 用户指定新口令。此选项与 QAnywhere 代理 -mn 选项相同。请参见“-mn 选项”一节第 711 页。
- **INC_UPLOAD** 指定增量上载大小。此选项与 QAnywhere 代理 -iu 选项相同。请参见“-iu 选项”一节第 710 页。
- **INC_DOWNLOAD** 指定增量下载大小。此选项与 QAnywhere 代理 -idl 选项相同。请参见“-idl 选项”一节第 709 页。
- **STORE_ID** 指定独立客户端要连接到的客户端消息存储库 ID。此选项与 QAnywhere 代理 -id 选项相同。请参见“-id 选项”一节第 709 页。
- **STORE_ENCRYPTION_KEY** 指定用于加密消息存储库的加密密钥。
- **POLICY** 指定确定何时发生消息传输的策略。此选项与 QAnywhere 代理 -policy 选项相同。请参见“-policy 选项”一节第 716 页。
- **DELETE_PERIOD** 指定对已达到最终状态的消息执行删除所间隔的秒数。
- **PUSH** 指定如何传送推式通知。此选项与 QAnywhere 代理 -push 选项相同。请参见“-push 选项”一节第 717 页。

QAnywhere 独立客户端不支持以下 QAManager 配置属性：

- CONNECT_PARAMS
- DATABASE_TYPE

另请参见

- “QAnywhere Manager 配置属性”一节第 90 页
- “qaagent 实用程序”一节第 704 页

移动 Web 服务

目录

移动 Web 服务介绍

Web 服务已成为显示应用程序功能的常用方式，并能在各种企业资源之间获得更好的互操作性。其拓展了移动应用程序的功能，并简化了开发过程。

由于可能无法获得连接（也可能是被中断），以及无线环境和设备的其它一些限制，要在移动环境中实现 Web 服务会很困难。例如，使用移动应用程序的用户可能希望在脱机时对某个 Web 服务发出请求，并能在用户联机时获得响应，或者一个 IT 管理员可能希望根据移动应用程序使用的网络连接类型（如 GPRS、802.11 或安装到底座上的）来指定可限制 Web 服务响应大小的规则。

QAnywhere 通过优化的移动异步 Web 服务来解决这些问题，这些服务可充分利用 QAnywhere 存储并转发消息传递结构体系。通过使用 QAnywhere 移动 Web 服务，您的移动应用程序可发出 Web 服务请求（即使在应用程序离线的情况下），并让这些请求排队等待以后进行传输。这些请求作为 QAnywhere 消息传送，然后服务器端的 Web 服务连接器会发出请求，从 Web 服务获得响应，然后以消息的形式将该响应返回给客户端。QAnywhere 传输规则可根据多种多样的参数（使用的网络、请求和响应的大小、位置、时间等）来控制对哪些请求和响应进行传输。结果是一个复杂且灵活的结构体系，利用这个体系，移动应用程序可通过使用已证明的技术和简单的编程模型来利用大量的 Web 服务功能。

从开发的角度来看，您可以使用 Web 服务代理类，就像在连接环境中一样，并且 QAnywhere 将处理传输、验证、序列化等等。提供了一个 WSDL 编译器来获取 WSDL 文档并生成特殊的代理类（.NET 或 Java），移动应用程序可使用这些类来调用 Web 服务。这些类使用基础的 QAnywhere 基础结构来发送请求和接收响应。调用了某个对象方法后，一个 SOAP 请求便自动创建，并作为消息发送至服务器，在这里连接器会发出 Web 服务请求并以消息的形式返回结果。

另请参见

- “移动 Web 服务”一节 《SQL Anywhere 11 - 简介》

设置移动 Web 服务

以下步骤概述了设置移动 Web 服务所需的步骤。

♦ 设置移动 Web 服务概述

1. 如果尚不具备服务器消息存储库，请先行设置。

   请参见“设置服务器消息存储库”一节第 21 页。

2. 使用 -m 选项和到服务器消息存储库的连接启动 MobiLink 服务器。

   请参见“启动启用了 MobiLink 的 QAnywhere”一节第 30 页。

3. 如果尚不具备客户端消息存储库，请先行设置。这些是用来临时存储消息的 SQL Anywhere 数据库。

   请参见“设置客户端消息存储库”一节第 23 页。

4. 运行 iAnywhere WSDL 编译器，创建可在应用程序中使用的类。

   请参见“运行 iAnywhere WSDL 编译器”一节第 104 页。

5. 为每个客户端，编写一个使用 WSDL 编译器所生成的类的 Web 服务客户端应用程序。

   请参见“编写移动 Web 服务应用程序”一节第 106 页。

6. 创建 Web 服务连接器。

   请参见“Web 服务连接器”一节第 163 页。

7. 使用到本地客户端消息存储库的连接为每个客户端启动 QAnywhere 代理 (qaagent)。

   请参见“启动 QAnywhere 代理”一节第 48 页。

其它入门资源

- “移动 Web 服务示例”一节第 115 页中介绍了如何用 Java 设置 Weather Web 服务的示例。
- 使用货币交换 Web 服务的移动 Web 服务示例应用程序安装在 *samples-dir\QAnywhere \MobileWebServices* 中。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）此示例提供 Java 和 C# 两种版本。
- 可在 QAnywhere 新闻组上发布问题：ianywhere.public.sqlanywhere.qanywhere

移动 Web 服务开发提示

- 对于移动 Web 服务 .NET 应用程序，WSDL 编译器生成的代理类必须编译到应用程序可执行文件中。而不能编译到其自己的程序集中。
- 对于移动 Web 服务 Java 应用程序，必须使用 JDK 1.5.x。
- iAnywhere WSDL 编译器不支持 CHAR 数据类型。建议您用 STRING 数据类型替代 CHAR 数据类型。

运行 iAnywhere WSDL 编译器

假定有一个介绍 Web 服务的 WSDL 源，则 iAnywhere WSDL 编译器将生成一组包含在应用程序中的 Java 代理类、C# 代理类或 SQL Anywhere 的 SQL SOAP 客户端过程。

由 WSDL 编译器生成的 Java 或 C# 代理类用于与 QAnywhere 一起使用。这些类将 Web 服务操作显示为方法调用。所生成的类包括：

- 主要服务绑定类（此类继承自移动 Web 服务运行库中的 ianywhere.qanywhere.ws.WSBase）。
- WSDL 文件中所指定的每个复杂类型的代理类。

有关生成的代理类的信息，请参见：

- .NET： “用于 Web 服务的 QAnywhere .NET (.NET 2.0)”一节第 333 页
- Java： “用于 Web 服务的 QAnywhere Java API”一节第 610 页

WSDL 编译器支持 WSDL 1.1 和 SOAP 1.1（通过 HTTP 和 HTTPS）。

语法

wsdlc [*options*] *wsdl-uri*

wsdl-uri:

它是对 WSDL（Web Services Description Language，简称 Web 服务描述语言）源（URL 或文件）的说明。

选项：

- **-h** 显示帮助文本。
- **-v** 显示详细信息。
- **-o *output-directory*** 指定生成文件的输出目录。
- **-l *language*** 指定生成文件的语言。该语言是 **java**、**cs** 或 **sql** 中的一种。这些选项必须以小写字母指定。
- **-d** 当联系 iAnywhere 客户支持时，显示可能有用的调试信息。

特定于 Java 的选项：

- **-p *package*** 指定一个程序包名。这使您可以覆盖缺省的程序包名。

特定于 C# 的选项：

- **-n *namespace*** 指定命名空间。使用此选项可在所选命名空间中封装生成的类。

特定于 SQL 的选项：

- **-f *filename*** （必需）指定写入 SQL 语句的 SQL 输出文件的名称。此操作将覆盖所有现有同名文件。
- **-p=*prefix*** 为已生成的函数或过程名指定前缀。缺省的前缀是后跟一个句点的服务名（例如，"WSDish."）。

- **-x** 生成过程定义而不是函数定义。

编写移动 Web 服务应用程序

您的应用程序会向 QAnywhere 发送一个 Web 服务请求，QAnywhere 将该请求发送到 MobiLink 服务器中的移动 Web 服务连接器。连接器会将该请求发送到 Web 服务，或对该请求进行排队，直到 Web 服务可用为止。QAnywhere 收到响应时，它会通知应用程序，或将该响应排队，直到应用程序可用为止。

设置 .NET 移动 Web 服务应用程序

在将 .NET 同 QAnywhere 一起使用前，必须对 Visual Studio 项目作如下更改：

- 添加对 QAnywhere .NET DLL 和移动 Web 服务 .NET DLL 的引用。这可通知 Visual Studio 包含哪个 DLL 以找到 QAnywhere .NET API 的代码和移动 Web 服务 .NET API 的代码。
- 向您的源代码中添加一些行以引用 QAnywhere .NET API 类和移动 Web 服务 .NET API 类。要使用 QAnywhere .NET API，您必须在源代码中添加一行以引用数据提供程序。必须为 C# 添加与 Visual Basic 不同的语句行。

完整的说明如下。

♦ 在 Visual Studio 项目中添加对 QAnywhere .NET API 和对移动 Web 服务 API 的引用

1. 启动 Visual Studio 并打开项目。
2. 在 [Solution Explorer] 窗口中，右击 [**References**] 文件夹，然后从弹出式菜单中选择 [**Add Reference**]。
3. 在 [**浏览**] 选项卡上，在以下目录中定位 *iAnywhere.QAnywhere.Client.dll* 和 *iAnywhere.QAnywhere.WS.dll*：
   - .NET Framework 2.0：*install-dir\Assembly\v2*
   - .NET Compact Framework 2.0：*install-dir\ce\Assembly\v2*

   在您的环境的适当目录中，选择每个 DLL 并单击 [**Open**]。
4. 要验证 DLL 是否已添加到项目，请展开 Solution Explorer 中的 [**References**] 树。*iAnywhere.QAnywhere.Client.dll* 和 *iAnywhere.QAnywhere.WS.dll* 应出现在列表中。

在源代码中引用数据提供程序类

♦ 在代码中引用 QAnywhere .NET API 类和移动 Web 服务 API 类

1. 启动 Visual Studio 并打开项目。
2. 如果使用 C#，请将以下语句行添加到文件开始处的 using 指令列表中：

```
using iAnywhere.QAnywhere.Client;
using iAnywhere.QAnywhere.WS;
```

3. 如果使用 Visual Basic，请将以下语句行添加到文件开始处的导入列表中：

```
Imports iAnywhere.QAnywhere.Client
Imports iAnywhere.QAnywhere.WS
```

并不严格要求使用导入语句行。但是，它使您可以使用 QAnywhere 和移动 Web 服务类的简写形式。如果没有它们，您仍然可以在代码中使用完全限定类名。例如，以下代码使用长格式：

```
iAnywhere.QAnywhere.Client.QAManager
mgr =
 new iAnywhere.QAnywhere.Client.QAManagerFactory.Instance.CreateQAManager(
"qa_manager.props" );
```

以下代码使用简写形式：

```
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(
 "qa_manager.props" );
```

♦ 对 .NET 初始化 QAnywhere 和移动 Web 服务

1. 包括 *iAnywhere.QAnywhere.Client* 和 *iAnywhere.QAnywhere.WS* 命名空间，如上面的过程中所述。

   ```
   using iAnywhere.QAnywhere.Client;
   using iAnywhere.QAnywhere.WS;
   ```

2. 创建 QAManager 对象。

   例如，要创建缺省的 QAManager 对象，可调用 CreateQAManager（以 null 作为其参数）：

   ```
   QAManager mgr;
   mgr = QAManagerFactory.Instance.CreateQAManager( null );
   ```

   提示
   为了提供最大并发数，多线程应用程序应为每个线程创建一个 QAManager。请参见“多线程注意事项”一节第 89 页。

   有关 QAManagerFactory 的详细信息，请参见“QAManagerFactory 类”一节第 298 页。

   您也可以创建使用属性文件自定义的 QAManager 对象。属性文件在 CreateQAManager 方法中指定：

   ```
   mgr = QAManagerFactory.Instance.CreateQAManager(
     "qa_mgr.props" );
   ```

   其中，*qa_mgr.props* 是远程设备上属性文件的名称。

3. 初始化 QAManager 对象。例如：

   ```
   mgr.Open(
      AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
   ```

   open 方法的参数是确认模式，它指示确认消息的方式。它必须为 IMPLICIT_ACKNOWLEDGEMENT 或 EXPLICIT_ACKNOWLEDGEMENT。

   移动 Web 服务所使用的 QAnywhere 消息无法访问移动 Web 服务应用程序。当在 EXPLICIT_ACKNOWLEDGEMENT 模式下使用 QAManager 时，可使用 WSResult 的 Acknowledge 方法来确认包含 Web 服务请求结果的 QAnywhere 消息。此方法指示应用程序是否已经成功地处理了该响应。

有关确认模式的详细信息，请参见：

- WSBase“SetQAManager 方法”一节第 338 页
- WSResult“Acknowledge 方法”一节第 351 页

> 您可以创建 QATransactionalManager 而非 QAManager。请参见“为 .NET 客户端实现事务性消息传递”一节第 68 页。

4. 创建一个服务绑定类的实例。

   移动 Web 服务 WSDL 编译器可从定义 Web 服务的 WSDL 文档生成服务绑定类。

   QAManager 由 Web 服务绑定类的实例使用，以在发出 Web 服务请求的过程中执行消息传送操作。通过设置服务绑定类的属性 WS_CONNECTOR_ADDRESS，您可指定要用来发送 Web 服务请求（通过 QAnywhere）的连接器地址。您可使用要连接的 Web 服务 URL 来配置每个 QAnywhere Web 服务连接器，并且如果某个应用程序需要位于多个 URL 的 Web 服务，则对每个 URL 配置该连接器。

   例如：

```
CurrencyConverterSoap service = new CurrencyConverterSoap( )
service.SetQAManager(mgr);
service.setProperty(
   "WS_CONNECTOR_ADDRESS",
   "ianywhere.connector.currencyconvertor\\");
```

   注意地址的结尾必须包含 \\。

另请参见

- “用于 Web 服务的 QAnywhere .NET (.NET 2.0)”一节第 333 页
- “用于客户端的 QAnywhere .NET (.NET 2.0)”一节第 212 页

示例

要初始化移动 Web 服务，就必须创建一个 QAManager 和一个服务绑定类的实例。例如：

```
// QAnywhere initialization
  QAManager mgr = QAManagerFactory.Instance.CreateQAManager( null );
  mgr.SetProperty( "CONNECT_PARAMS",
"eng=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
  mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
  mgr.Start();

  // Instantiate the web service proxy
  CurrencyConvertorSoap service = new CurrencyConvertorSoap();
  service.SetQAManager( mgr );
  service.SetProperty( "WS_CONNECTOR_ADDRESS",
"ianywhere.connector.currencyconvertor\\" );
```

CurrencyConvertor 示例的响应时间取决于 Web 服务的可用性。当移动 Web 服务应用程序并不总是可用时，异步 Web 服务请求会很有用。使用此方法，通过调用服务绑定类中的方法来发出 Web 服务请求，以将请求放置在外发队列中。该方法返回 WSResult，该结果可在以后（甚至是应用程序被重新启动之后）用来查询响应的状态。请参见“异步 Web 服务请求”一节第 112 页。

设置 Java 移动 Web 服务应用程序

要在 Java 中创建移动 Web 服务应用程序，您就必须完成以下初始化任务。

♦ 对 Java 初始化 QAnywhere 和移动 Web 服务

1. 将以下文件的位置添加到类路径中。缺省情况下，这些文件位于 *install-dir\Java* 下：
   - *qaclient.jar*
   - *iawsrt.jar*
   - *jaxrpc.jar*
2. 导入 ianywhere.qanywhere.client 和 ianywhere.qanywhere.ws 包：

   ```
   import ianywhere.qanywhere.client.*;
   import ianywhere.qanywhere.ws.*;
   ```

3. 创建 QAManager 对象。

   ```
   QAManager mgr;
   mgr = QAManagerFactory.getInstance().createQAManager(null);
   ```

   还可以为 createQAManager 方法指定属性文件来自定义 QAManager 对象：

   ```
   mgr = QAManagerFactory.getInstance().createQAManager("qa_mgr.props.");
   ```

   提示
   为了提供最大并发数，多线程应用程序应为每个线程创建一个 QAManager。请参见“多线程注意事项”一节第 89 页。

4. 初始化 QAManager 对象。

   ```
   mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
   ```

   open 方法的参数是确认模式，它指示确认消息的方式。它必须为 IMPLICIT_ACKNOWLEDGEMENT 或 EXPLICIT_ACKNOWLEDGEMENT。

   移动 Web 服务所使用的 QAnywhere 消息无法访问移动 Web 服务应用程序。当在 EXPLICIT_ACKNOWLEDGEMENT 模式下使用 QAManager 时，可使用 WSResult 的 Acknowledge 方法来确认包含 Web 服务请求结果的 QAnywhere 消息。此方法指示应用程序是否已经成功地处理了该响应。

   有关确认模式的详细信息，请参见：

   - WSBase“setQAManager 方法”一节第 614 页
   - WSResult“acknowledge 方法”一节第 621 页

   您可以创建 QATransactionalManager 而非 QAManager。请参见“为 Java 客户端实现事务性消息传递”一节第 70 页。

5. 创建一个服务绑定类的实例。

   移动 Web 服务 WSDL 编译器可从定义 Web 服务的 WSDL 文档生成服务绑定类。

在发出 Web 服务请求的过程中，QAManager 由 Web 服务绑定类的实例使用来执行消息传送操作。通过设置服务绑定类的 WS_CONNECTOR_ADDRESS 属性，您可指定要用来发送 Web 服务请求（通过 QAnywhere）的连接器地址。使用要连接的 Web 服务 URL 来对每个 QAnywhere Web 服务连接器进行配置。这表示，如果某个应用程序需要位于多个 URL 的 Web 服务，则必须对每个服务 URL 配置 QAnywhere 连接器。

例如：

```
CurrencyConverterSoap service = new CurrencyConverterSoap( );
service.setQAManager(mgr);
service.setProperty("WS_CONNECTOR_ADDRESS",
"ianywhere.connector.currencyconvertor\\");
```

注意地址的结尾必须包含 \\。

另请参见

- “用于 Web 服务的 QAnywhere Java API”一节第 610 页
- “用于客户端的 QAnywhere Java API”一节第 498 页

示例

要初始化移动 Web 服务，就必须创建一个 QAManager 和一个服务绑定类的实例。例如：

```
// QAnywhere initialization
  Properties props = new Properties();
  props.put( "CONNECT_PARAMS",
"eng=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
  QAManager mgr = QAManagerFactory.getInstance().createQAManager( props );
  mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
  mgr.start();

  // Instantiate the web service proxy
  CurrencyConvertorSoap service = new CurrencyConvertorSoap();
  service.setQAManager( mgr );
  service.setProperty( "WS_CONNECTOR_ADDRESS",
"ianywhere.connector.currencyconvertor\\" );
```

服务绑定类的多个实例

应对每个 QAManager 创建一个服务绑定类的实例。如果移动 Web 服务应用程序具有多个服务绑定类实例，则使用 SetServiceID 方法来设置服务 ID 非常重要。例如：

```
service1.SetServiceID("1")
service2.SetServiceID("2")
```

服务 ID 同服务名称相结合，构成接收 Web 服务响应的队列名。重要的是每个给定服务的实例具有唯一的服务 ID，使得给定的实例不会获得由其它服务的实例所发出的请求的响应。如果未设置服务 ID，则缺省为 ""。服务 ID 对于避免使用相同服务的多个应用程序相互发生冲突也非常重要，因为队列名将消息保存在瞬时性应用程序间的消息存储库中。

编译并运行移动 Web 服务应用程序

运行时库

Java 的运行时库为 *iawsrt.jar*，位于 *install-dir\Java* 中。

C# 的运行时库为 *iAnywhere.QAnywhere.WS.dll*，位于以下目录中：

- .NET Framework 2.0：*install-dir\Assembly\v2*
- .NET Compact Framework 2.0：*install-dir\ce\Assembly\v2*

下面的部分介绍需要用来编译和运行移动 Web 服务应用程序的文件。

所需的运行时库 (Java)

包括类路径中的以下文件。这些文件位于 *install-dir\Java* 下：

- *jaxrpc.jar*
- *qaclient.jar*
- *iawsrt.jar*

所需的运行时库 (.NET)

SQL Anywhere 11 安装会自动将以下文件包含在您的全局程序集高速缓存中：

- *iAnywhere.QAnywhere.Client.dll*
- *iAnywhere.QAnywhere.WS.dll*

关闭移动 Web 服务

移动 Web 服务应用程序通过关闭 QAManager 来依次执行关闭。例如：

```
// QAnywhere finalization in C#:
mgr.Stop();
mgr.Close();

// QAnywhere finalization in Java:
mgr.stop();
mgr.close();
```


发出 Web 服务请求

在移动 Web 服务应用程序中，有以下两种基本方法可用来发出 Web 服务请求：

- **同步** 请参见“同步 Web 服务请求”一节第 112 页。
- **异步** 请参见“异步 Web 服务请求”一节第 112 页。

同步 Web 服务请求

在应用程序被连接至网络时使用同步 Web 服务请求。使用此方法，通过调用服务绑定类中的方法来发出 Web 服务请求，且仅当已接收到来自服务器的 Web 服务响应时才返回结果。

示例

下面的示例发出请求以获得美元对加元的汇率：

```
//C#
double r = service.ConversionRate( Currency.USD, Currency.CAD );

// Java
double r = service.conversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );
```

异步 Web 服务请求

仅在移动 Web 服务应用程序不定时地连接至网络时，异步 Web 服务请求才有用。使用此方法，通过调用服务绑定类中的方法来发出 Web 服务请求，以将请求放置在外发队列中。该方法返回 WSResult，该结果可在以后（甚至是应用程序被重新启动之后）用来查询响应的状态。

下面的示例发出一个异步请求以获得美元对加元的汇率：

```
// C#
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
string reqID = r.GetRequestID();

// Later: get the response for the specified request ID
WSResult r = service.GetResult( reqID );
if( r.GetStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  Console.WriteLine( "The conversion rate is " +
r.GetDoubleValue( "ConversionRateResult" ) );
} else {
  Console.WriteLine( "Response not available" );
}
// Java
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
String reqID = r.getRequestID();

// Later: get the response for the specified request ID
```



```
WSResult r = service.getResult( reqID );
if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
    System.out.println( "The conversion rate is " +
r.getDoubleValue( "ConversionRateResult" ) );
} else {
    System.out.println( "Response not available" );
}
```

当可获得对 Web 服务请求的响应时，还可使用 WSListener 获取异步回调。例如：

```
// C#
// Make a request to get the USD to CAD exchange rate
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Register a listener for the result
service.SetListener( r.GetRequestID(), new CurrencyConvertorListener() );

// Java
// Make a request to get the USD to CAD exchange rate
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Register a listener for the result
service.setListener( r.getRequestID(), new CurrencyConvertorListener() );
```

WSListener 接口定义两种方法来处理异步事件：

- **OnResult** 执行 OnResult 方法来处理对 Web 服务请求的响应。这样会传递表示 Web 服务请求的结果的 WSResult 对象。
- **OnException** 执行 OnException 方法来处理在对 Web 服务请求的响应进行处理期间所产生的错误。这样会传递 WSException 对象和 WSResult 对象。WSException 对象包含有关发生的错误的信息，而 WSResult 对象可用来获得响应所对应的请求 ID。

```
// C#
class CurrencyConvertorListener : WSListener
{
   public CurrencyConvertorListener() {
   }

   public void OnResult( WSResult r ) {
    try {
         USDToCAD._statusMessage = "USD to CAD currency exchange rate: " +
r.GetDoubleValue( "ConversionRateResult" );
    } catch( Exception exc ) {
        USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: "
+ exc.Message;
    }
   }

   public void OnException( WSException exc, WSResult r ) {
    USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: " +
exc.Message;
   }
}
// Java
private class CurrencyConvertorListener implements WSListener
{
  public CurrencyConvertorListener() {
  }

  public void onResult( WSResult r ) {
```

```
   try {
        USDToCAD._statusMessage = "USD to CAD currency exchange rate: " +
r.getDoubleValue( "ConversionRateResult" );
   } catch( Exception exc ) {
        USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed: "
+ exc.getMessage();
   }
   }

   public void onException( WSException exc, WSResult r ) {
        USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed: "
+ exc.getMessage();
   }
}
```


移动 Web 服务示例

此示例向您演示创建移动 Web 服务应用程序的方法。创建此应用程序只需花费几分钟，它使用 QAnywhere 的存储并转发功能，使您即使在脱机状态也可以发出气象报告请求，然后在该报告可用时查看该报告。

Global Weather Web 服务

以下代码描述一个名为 Global Weather 的 Web 服务。（这是一个从公共气象 Web 服务复制而来的 wsdl 文件。）将代码复制到一个文件，将该文件命名为 *globalweather.wsdl*：

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://
www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:tns="http://www.webserviceX.NET" xmlns:tm="http://
microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://
schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://www.webserviceX.NET"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://
www.webserviceX.NET">
      <s:element name="GetWeather">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CityName"
type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="CountryName"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetWeatherResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCitiesByCountry">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CountryName"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCitiesByCountryResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetCitiesByCountryResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetWeatherSoapIn">
```

```
      <wsdl:part name="parameters" element="tns:GetWeather" />
    </wsdl:message>
    <wsdl:message name="GetWeatherSoapOut">
      <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountrySoapIn">
      <wsdl:part name="parameters" element="tns:GetCitiesByCountry" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountrySoapOut">
      <wsdl:part name="parameters" element="tns:GetCitiesByCountryResponse" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpGetIn">
      <wsdl:part name="CityName" type="s:string" />
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpGetOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpGetIn">
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpGetOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpPostIn">
      <wsdl:part name="CityName" type="s:string" />
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpPostOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpPostIn">
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpPostOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:portType name="GlobalWeatherSoap">
      <wsdl:operation name="GetWeather">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather
report for all major cities around the world.</documentation>
        <wsdl:input message="tns:GetWeatherSoapIn" />
        <wsdl:output message="tns:GetWeatherSoapOut" />
      </wsdl:operation>
      <wsdl:operation name="GetCitiesByCountry">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major
cities by country name(full / part).</documentation>
        <wsdl:input message="tns:GetCitiesByCountrySoapIn" />
        <wsdl:output message="tns:GetCitiesByCountrySoapOut" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:portType name="GlobalWeatherHttpGet">
      <wsdl:operation name="GetWeather">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather
report for all major cities around the world.</documentation>
        <wsdl:input message="tns:GetWeatherHttpGetIn" />
        <wsdl:output message="tns:GetWeatherHttpGetOut" />
      </wsdl:operation>
      <wsdl:operation name="GetCitiesByCountry">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major
cities by country name(full / part).</documentation>
        <wsdl:input message="tns:GetCitiesByCountryHttpGetIn" />
        <wsdl:output message="tns:GetCitiesByCountryHttpGetOut" />
      </wsdl:operation>
```



```
  </wsdl:portType>
  <wsdl:portType name="GlobalWeatherHttpPost">
    <wsdl:operation name="GetWeather">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather
report for all major cities around the world.</documentation>
      <wsdl:input message="tns:GetWeatherHttpPostIn" />
      <wsdl:output message="tns:GetWeatherHttpPostOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major
cities by country name(full / part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountryHttpPostIn" />
      <wsdl:output message="tns:GetCitiesByCountryHttpPostOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <wsdl:operation name="GetWeather">
      <soap:operation soapAction="http://www.webserviceX.NET/GetWeather"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <soap:operation soapAction="http://www.webserviceX.NET/
GetCitiesByCountry" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpGet" type="tns:GlobalWeatherHttpGet">
    <http:binding verb="GET" />
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpPost"
type="tns:GlobalWeatherHttpPost">
    <http:binding verb="POST" />
    <wsdl:operation name="GetWeather">
```

```
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="GlobalWeather">
    <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
      <soap:address location="http://www.webservicex.net/globalweather.asmx" /
>
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpGet"
binding="tns:GlobalWeatherHttpGet">
      <http:address location="http://www.webservicex.net/globalweather.asmx" /
>
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpPost"
binding="tns:GlobalWeatherHttpPost">
      <http:address location="http://www.webservicex.net/globalweather.asmx" /
>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

生成代理类

要创建一个移动应用程序来访问 Global Weather Web 服务，首先要运行 QAnywhere WSDL 编译器。它将生成一个代理类，这个代理类可用于应用程序中来发出此全球天气服务的请求。在本例中，应用程序是使用 Java 编写的。

```
wsdlc -l java globalweather.wsdl
```

此命令生成名为 *GlobalWeatherSoap.java* 的代理类，它位于当前目录的 *NET\webserviceX* 子目录中。此代理类是您的应用程序的服务绑定类。下面是 *GlobalWeatherSoap.java* 的内容：

```
/*
 * GlobalWeatherSoap.java
 *
 * Generated by the iAnywhere WSDL Compiler Version 10.0.1.3415
 * Do not edit this file.
 */

package NET.webserviceX;

import ianywhere.qanywhere.ws.*;
import ianywhere.qanywhere.client.QABinaryMessage;
import ianywhere.qanywhere.client.QAException;

import java.io.*;
```



```
import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.stream.*;

public class GlobalWeatherSoap extends ianywhere.qanywhere.ws.WSBase
{

    public GlobalWeatherSoap(String iniFile) throws WSException
    {
        super(iniFile);
        init();
    }

    public GlobalWeatherSoap() throws WSException
    {
        init();
    }

    public void init()
    {

        setServiceName("GlobalWeather");
    }


    public java.lang.String getWeather(java.lang.String cityName,
                        java.lang.String countryName) throws QAException,
WSException, WSFaultException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );
            String        responsePartName = "GetWeatherResult";
            java.lang.String    returnValue;

            writeSOAPHeader( hd, "GetWeather", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CityName",cityName,"string","http://
www.w3.org/2001/XMLSchema",true,true);

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetWeather", "GetWeatherResponse" );

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

            return returnValue;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }

    public WSResult asyncGetWeather(java.lang.String cityName,
```

```
                        java.lang.String countryName) throws QAException,
WSException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );

            writeSOAPHeader( hd, "GetWeather", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CityName",cityName,"string","http://
www.w3.org/2001/XMLSchema",true,true);

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetWeather", "GetWeatherResponse" );

            WSResult wsResult = invoke( qaRequestMsg );

            return wsResult;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }


    public java.lang.String getCitiesByCountry(java.lang.String countryName)
throws QAException, WSException, WSFaultException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );
            String        responsePartName = "GetCitiesByCountryResult";
            java.lang.String    returnValue;

            writeSOAPHeader( hd, "GetCitiesByCountry", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetCitiesByCountry" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetCitiesByCountry", "GetCitiesByCountryResponse" );

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

            return returnValue;
        } catch( TransformerConfigurationException e ) {
```



```
            throw new WSException( e );
        }
    }

    public WSResult asyncGetCitiesByCountry(java.lang.String countryName)
throws QAException, WSException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );

            writeSOAPHeader( hd, "GetCitiesByCountry", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetCitiesByCountry" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetCitiesByCountry", "GetCitiesByCountryResponse" );

            WSResult wsResult = invoke( qaRequestMsg );

            return wsResult;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }
}
```

编写移动 Web 服务应用程序

下一步，编写使用服务绑定类来发出 Web 服务的请求并处理结果的应用程序。下面是两个应用程序，二者均脱机发出 Web 服务请求，并在可以连接时处理结果。

第一个应用程序（名为 RequestWeather）发出全球天气服务的请求并显示请求的 ID。将以下代码复制到名为 *RequestWeather.java* 的文件中：

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import com.myweather.GlobalWeatherSoap;

class RequestWeather
{
    public static void main( String [] args ) {
    try {
        // QAnywhere initialization
        QAManager mgr = QAManagerFactory.getInstance().createQAManager();
        mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
        mgr.start();

        // Instantiate the web service proxy
        GlobalWeatherSoap service = new GlobalWeatherSoap();
        service.setQAManager( mgr );
        service.setProperty( "WS_CONNECTOR_ADDRESS",
"ianywhere.connector.globalweather\\" );

        // Make a request to get weather for Beijing
```

```
            WSResult r = service.asyncGetWeather( "Beijing", "China" );

            // Display the request ID so that it can be used by ShowWeather
            System.out.println( "Request ID: " + r.getRequestID() );

            // QAnywhere finalization
            mgr.stop();
            mgr.close();

        } catch( Exception exc ) {
            System.out.println( exc.getMessage() );
        }
        }
}
```

第二个应用程序称为 ShowWeather，它显示给定请求 ID 的气象条件。将以下代码复制到名为 *ShowWeather.java* 的文件中：

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import com.myweather.GlobalWeatherSoap;

class ShowWeather
{
   public static void main( String [] args ) {
   try {
     // QAnywhere initialization
     QAManager mgr = QAManagerFactory.getInstance().createQAManager();
     mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
     mgr.start();

     // Instantiate the web service proxy
     GlobalWeatherSoap service = new GlobalWeatherSoap();
     service.setQAManager( mgr );

     // Get the response for the specified request ID
     WSResult r = service.getResult( args[0] );
     if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  System.out.println( "The weather is " +
r.getStringValue( "GetWeatherResult" ) );
  r.acknowledge();
     } else {
  System.out.println( "Response not available" );
     }

     // QAnywhere finalization
     mgr.stop();
     mgr.close();

 } catch( Exception exc ) {
     System.out.println( exc.getMessage() );
 }
    }
}
```

编译应用程序和服务绑定类：

```
javac -classpath ".;%sqlany11%\java\iawsrt.jar;%sqlany11%\java\qaclient.jar"
com\myweather\GlobalWeatherSoap.java RequestWeather.java
javac -classpath ".;%sqlany11%\java\iawsrt.jar;%sqlany11%\java\qaclient.jar"
com\myweather\GlobalWeatherSoap.java ShowWeather.java
```


创建 QAnywhere 消息存储库并启动 QAnywhere 代理

您的移动 Web 服务应用程序在每个移动设备上都需要一个客户端消息存储库。它还需要服务器消息存储库，但本例使用 QAnywhere 示例服务器消息存储库。

要创建一个客户端消息存储库，请使用 dbinit 实用程序创建一个 SQL Anywhere 数据库，然后运行 QAnywhere 代理以将其设置为客户端消息存储库：

```
dbinit -i qanywhere.db
qaagent -q -si -c "dbf=qanywhere.db"
```

启动 QAnywhere 代理，连接到客户端消息存储库：

```
qaagent -c "dbf=qanywhere.db;eng=qanywhere;uid=ml_qa_user;pwd=qanywhere"
```

启动 QAnywhere 服务器：

```
mlsrv11 -m -zu+ -c "dsn=QAnywhere 11
Demo;uid=ml_server;pwd=sql;start=dbsrv11" -v+ -ot qanyserv.mls
```

♦ 创建 Web 服务连接器

创建一个 Web 服务连接器，该连接器可监听发送至 GetWeather Web 服务的 QAnywhere 消息，可在消息到达时调用 Web 服务，并可将响应发送回源客户端。

1. 打开 Sybase Central，然后单击 [**连接**] » [**使用 QAnywhere 11 连接**]。
2. 在 [**用户 ID**] 字段中，键入 **ml_server**。
3. 在 [**口令**] 字段中键入 **sql**。
4. 单击 [**ODBC 数据源名称**]，然后浏览到 QAnywhere 11 Demo 的位置。
5. 单击 [**OK**]。
6. 选择 [**文件**] » [**新建**] » [**连接器**]。
7. 单击 [**Web 服务**]。单击 [**下一步**]。
8. 在 [**连接器名称**] 字段中，键入 **ianywhere.connector.globalweather**。单击 [**下一步**]。
9. 在 [**URL**] 字段中，键入 **http://www.webservicex.net/globalweather.asmx**。单击 [**完成**]。

使用 Web 服务

要对 Web 服务中获取气象报告的请求排队，请键入：

```
java -classpath ".;%sqlany11%\java\iawsrt.jar;%sqlany11%\java\qaclient.jar"
RequestWeather
```

返回请求 ID。

要查看气象报告，请键入以下内容。结尾应为请求 ID，本例中为 REQ123123123。

```
java -classpath ".;%sqlany11%\java\iawsrt.jar;%sqlany11%\java\qaclient.jar"
ShowWeather REQ123123123
```

返回详细的气象报告。

部署 QAnywhere

目录

部署 QAnywhere 应用程序

QAnywhere 为 SQL Anywhere 消息存储库提供了 C++、Java 和 .NET API 支持。Java 和 .NET API 也支持 UltraLite 消息存储库。部署 QAnywhere 应用程序所需的文件要根据您的 Windows 环境、消息存储库类型以及所选择的 API 来确定。如果是部署移动 Web 服务应用程序，则还需要一些附加的文件。

除了下面列出的文件外，QAnywhere 应用程序需要：

- 在“部署 SQL Anywhere MobiLink 客户端”一节 《MobiLink - 服务器管理》的 "MobiLink 同步客户端"、"监听器" 和可选的 "安全" 等小节中列出的所有文件。只有使用推式通知时才需要监听器文件,这是缺省情况。
- “部署数据库服务器”一节 《SQL Anywhere 服务器 - 编程》中的 dbeng11 或 dbsrv11 文件。

要部署 Sybase Central，请参见“部署管理工具”一节 《SQL Anywhere 服务器 - 编程》。

Windows 应用程序

所有目录都相对于 *install-dir* 目录。

有关 Windows Mobile 环境文件结构的详细信息，请参见“Windows Mobile 应用程序”一节 《MobiLink - 服务器管理》。

以下是安装 SQL Anywhere 消息存储库所需文件的列表。

客户端 API	Windows 文件
C++	● *bin32\qany11.dll* ● *bin32\qaagent.exe* ● *bin32\qastop.exe*
Java	● *bin32\qaagent.exe* ● *bin32\qastop.exe* ● *java\qaclient.jar* ● *java\jodbc.jar* 对于移动 Web 服务应用程序，还需要以下文件： ● *java\iawsrt.jar* ● *java\jaxrpc.jar*

客户端 API	Windows 文件
.NET	● *bin32\qazlib.dll* ● *bin32\qaagent.exe* ● *bin32\qastop.exe* ● *assembly\v2\iAnywhere.QAnywhere.Client.dll* ● *assembly\v2\iAnywhere.QAnywhere.Resources.dll* ● *assembly\v2\iAnywhere.Data.SQLAnywhere.dll* 对于移动 Web 服务应用程序，还需要以下文件： ● *Assembly\v2\iAnywhere.QAnywhere.WS.dll*

以下是安装 UltraLite 消息存储库所需文件的列表。

客户端 API	Windows 文件
Java	● *bin32\qauagent.exe* ● *bin32\qastop.exe* ● *bin32\qadbiuljni.dll* ● *java\qaclient.jar* 对于移动 Web 服务应用程序，还需要以下文件： ● *java\iawsrt.jar* ● *java\jaxrpc.jar*
.NET	● *bin32\qazlib.dll* ● *bin32\qauagent.exe* ● *bin32\qastop.exe* ● *assembly\v2\iAnywhere.QAnywhere.Client.dll* ● *assembly\v2\iAnywhere.QAnywhere.Resources.dll* ● *ultralite\ultralite.NET\assembly\v2\iAnywhere.Data.UltraLite.dll* 对于移动 Web 服务应用程序，还需要以下文件： ● *Assembly\v2\iAnywhere.QAnywhere.WS.dll*

创建 UltraLite 消息存储库时，必须使用 UltraLite 创建数据库实用程序创建一个 udb 数据库文件，然后再使用 QAnywhere UltraLite 代理的 -si 选项初始化此数据库。请参见“UltraLite 创建数据库实用程序 (ulcreate)”一节 《UltraLite - 数据库管理和参考》和“qauagent 实用程序”一节第 725 页。

Windows Mobile 应用程序

所有目录都相对于 *install-dir* 目录。

有关 Windows 环境文件结构的详细信息，请参见“Windows 应用程序”一节 《MobiLink - 服务器管理》。

以下是安装 SQL Anywhere 消息存储库所需文件的列表。

客户端 API	Windows Mobile 文件
C++	● *ce\arm.50\qany11.dll* ● *ce\arm.50\qaagent.exe* ● *ce\arm.50\qastop.exe*
Java	● *ce\arm.50\qaagent.exe* ● *ce\arm.50\qastop.exe* ● *java\qaclient.jar* ● *java\jodbc.jar* 对于移动 Web 服务应用程序，还需要以下文件： ● *java\iawsrt.jar* ● *java\jaxrpc.jar*
.NET	● *ce\arm.50\qazlib.dll* ● *ce\arm.50\qaagent.exe* ● *ce\arm.50\qastop.exe* ● *ce\assembly\v2\iAnywhere.QAnywhere.Client.dll* ● *ce\assembly\v2\iAnywhere.QAnywhere.Resources.dll* ● *ce\assembly\v2\iAnywhere.Data.SQLAnywhere.dll* 对于移动 Web 服务应用程序，还需要以下文件： ● *ce\Assembly\v2\iAnywhere.QAnywhere.WS.dll*

以下是安装 UltraLite 消息存储库所需文件的列表。

客户端 API	Windows Mobile 文件
Java	● *ce\arm.50\qauagent.exe* ● *ce\arm.50\qastop.exe* ● *ce\arm.50\qadbiuljni.dll* ● *java\qaclient.jar* 对于移动 Web 服务应用程序，还需要以下文件： ● *java\iawsrt.jar* ● *java\jaxrpc.jar*

客户端 API	Windows Mobile 文件
.NET	● *ce\arm.50\qazlib.dll* ● *ce\arm.50\qauagent.exe* ● *ce\arm.50\qastop.exe* ● *ce\assembly\v2\iAnywhere.QAnywhere.Client.dll* ● *ce\assembly\v2\iAnywhere.QAnywhere.Resources.dll* ● *ultralite\ultralite.NET\ce\assembly\v2\iAnywhere.Data.UltraLite.dll* 对于移动 Web 服务应用程序，还需要以下文件： ● *ce\Assembly\v2\iAnywhere.QAnywhere.WS.dll*

创建 UltraLite 消息存储库时，必须使用 UltraLite 创建数据库实用程序创建一个 udb 数据库文件，然后再使用 QAnywhere UltraLite 代理的 -si 选项初始化此数据库。请参见“UltraLite 创建数据库实用程序 (ulcreate)”一节 《UltraLite - 数据库管理和参考》和“qauagent 实用程序”一节第 725 页。

注册 QAnywhere .NET API DLL

QAnywhere .NET API DLL（*Assembly\v2\iAnywhere.QAnywhere.Client.dll*）需要在 Windows（Windows Mobile 除外）的全局程序集高速缓存中注册。全局程序集高速缓存列出了计算机上所有已注册的程序。在安装 SQL Anywhere 时，安装程序会对它进行注册。在 Windows Mobile 上无需注册 DLL。

如果要部署 QAnywhere，必须使用包含在 .NET Framework 中的 gacutil 实用程序注册 QAnywhere .NET API DLL（*Assembly\v2\iAnywhere.QAnywhere.Client.dll*）。

编写安全的消息传递应用程序

目录

创建安全的客户端消息存储库

要保护客户端消息存储库的安全，可进行以下操作：

- 更改缺省口令。

  请参见“管理客户端消息存储库口令”一节第 132 页。
- 加密消息存储库的内容。

  请参见“加密客户端消息存储库”一节第 133 页。

示例

首先，创建一个带加密密钥的 SQL Anywhere 数据库：

```
dbinit mystore.db -i -s -ek some_phrase
```

选项 -i 和 -s 最适用于小型设备。-ek 选项指定高度加密的加密密钥。请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

然后，将该数据库初始化为客户端消息存储库：

```
qaagent -id mystore -si -c "dbf=mystore.db;dbkey=some_phrase"
```

接下来，新建一个具有 DBA 权限的远程用户，并为该用户创建一个口令。撤消缺省 QAnywhere 用户并更改缺省 DBA 用户的口令。以 DBA 为用户名并以 sql 为口令登录，然后执行以下 SQL 语句：

```
CREATE USER secure_user IDENTIFIED BY secure_password
GRANT MEMBERSHIP IN GROUP ml_qa_user_group TO secure_user
GRANT REMOTE DBA TO secure_user
REVOKE CONNECT FROM ml_qa_user
ALTER USER DBA IDENTIFIED BY new_dba_password
COMMIT
```

注意
所有 QAnywhere 用户都必须属于 ml_qa_user_group 并具有远程 DBA 权限。

接着，以安全的 DBA 用户的身份启动 QAnywhere 代理：

```
qaagent -id mystore -c
"dbf=mystore.db;dbkey=some_phrase;uid=secure_user;pwd=secure_password"
```

管理客户端消息存储库口令

应该更改为消息存储库创建的缺省用户 ID 的口令。系统会为每个 SQL Anywhere 数据库创建缺省用户 ID DBA，口令为 SQL。此外，qaagent -si 选项还会创建缺省用户 ID ml_qa_user，以及缺省口令 qanywhere。要更改这些口令，请使用 GRANT 语句。

请参见“更改口令”一节 《SQL Anywhere 服务器 - 数据库管理》。

加密客户端消息存储库

在创建客户端消息存储库时，可使用下面的命令进行加密。

```
dbinit -i -s -ek encryption-key database-file
```

（使用选项 -i 和 -s 在小型设备上创建数据库是很好的做法。）如果已使用加密密钥初始化某个消息存储库，则必须具有该密钥才能启动加密消息存储库上的数据库服务器。

请使用以下命令指定加密密钥，以启动包含加密消息存储库的 QAnywhere 代理。QAnywhere 代理将使用提供的加密密钥，自动启动加密消息存储库上的数据库服务器。

```
qaagent -c "DBF=database-file;DBKEY=encryption-key"
```

现在，任何应用程序都可以通过 QAnywhere API 访问加密消息存储库。请注意，由于用于管理消息存储库的数据库服务器已在运行，所以应用程序不需要提供加密密钥。

如果 QAnywhere 代理未运行而应用程序需要访问加密消息存储库，则 QAnywhere API 将使用 QAnywhere Manager 初始化文件中指定的连接参数自动启动数据库服务器。要启动加密消息存储库上的数据库服务器，必须按以下形式在数据库连接参数中指定加密密钥。

```
CONNECT_PARAMS=DBF=database-file;DBKEY=encryption-key
```

另请参见

- “加密和解密数据库”一节 《SQL Anywhere 服务器 - 数据库管理》
- “初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》
- QAnywhere 代理“-c 选项”一节第 706 页

加密通信流

可以使用 qaagent -x 选项指定一个 QAnywhere 代理可用于与 MobiLink 服务器进行通信的安全通信流。该选项使您可以使用服务器端证书进行服务器验证，还使您可以使用高度加密来加密通信流。

请参见“-x 选项”一节第 723 页。

您还必须为 MobiLink 服务器设置传送层安全性。有关创建数字证书和设置 MobiLink 服务器的信息，请参见“加密 MobiLink 客户端/服务器通信”一节 《SQL Anywhere 服务器 - 数据库管理》。

需要单独授予许可的组成部分
ECC 加密和 FIPS 认证的加密需要单独的许可。所有高度加密技术受出口法规约束。

请参见“单独授权的组件”一节 《SQL Anywhere 11 - 简介》。

示例

以下示例显示如何在 QAnywhere 代理和 MobiLink 服务器之间建立安全通信流。它们使用在安装 SQL Anywhere 安全性组件时安装的示例标识文件。

使用 RSA 的安全的 TCP/IP：

```
mlsrv11 -x tls(tls_type=rsa;identity=rsaserver.id;identity_password=test)
qaagent -x tls(tls_type=rsa;trusted_certificates=rsaroot.crt)
```

使用 ECC 保障 TCP/IP 的安全：

```
mlsrv11 -x tls(tls_type=ecc;identity=eccserver.id;identity_password=test)
qaagent -x tls(tls_type=ecc;trusted_certificates=eccroot.crt)
```

使用 HTTPS 保障 HTTP 安全（只针对 HTTPS 支持 RSA 证书）：

```
mlsrv11 -x https(identity=rsaserver.id;identity_password=test)
qaagent -x https(trusted_certificates=rsaroot.crt)
```


对 MobiLink 进行口令验证

在远程设备和服务器之间建立了安全的通信流后，您可能还需要对用户进行验证，以确保他们能够与服务器进行通信。

可通过为客户端消息存储库创建 MobiLink 用户名并将其在服务器消息存储库中注册来完成该操作。

另请参见

- “-mu 选项”一节第 712 页
- “-mp 选项”一节第 712 页
- “MobiLink 用户”《MobiLink - 客户端管理》

保护服务器管理请求的安全

可使用口令保护服务器管理请求的安全。消息字符串属性 ias_ServerPassword 用于指定服务器口令。服务器口令通过 ianywhere.qa.server.password.e 属性进行设置。如果未设置该属性，则口令为 QAnywhere。

服务器口令以文本形式传输。请使用加密通信流发送需要服务器口令的服务器管理请求。

有关 ianywhere.qa.server.password.e 属性的详细信息，请参见“服务器属性”一节第 753 页。

使用 MobiLink 用户验证实用程序添加用户

为确保安全性，应使用 MobiLink 用户验证实用程序 (mluser) 添加用户。mluser 实用程序允许您通过统一数据库上的服务器消息存储库注册 MobiLink 用户名。然后，用户名和口令参数被 QAnywhere 代理用于在消息传输过程中验证用户。请参见“MobiLink 用户验证实用程序 (mluser)”一节 《MobiLink - 服务器管理》。

如果您使用推式通知，则还有必要为监听器 (dblsn) 添加 MobiLink 用户。对于各个添加的用户，监听器还必须添加用户名 ias_[user]_lsn。

还可以通过 mlsrv11 使用 -zu+ 选项添加新用户，但是，如果安全保障有问题，则不应使用此选项。使用带 -zu+ 选项的 mlsrv11，可在所有新用户第一次进行同步时将其添加到统一数据库。这意味着无法识别的用户未通过验证就被添加。请参见“-zu 选项”一节 《MobiLink - 服务器管理》。

中继服务器的安全性

使用中继服务器时，设置中继服务器配置文件中的选项以控制所需的安全级别。请参见“中继服务器配置文件”一节 《MobiLink - 服务器管理》。

建立与 Web 服务器之间的安全通信流（例如，带受信任证书的 HTTPS）以确保 QAnywhere 代理和 Web 服务器之间通信的安全性。请参见 qaagent“-x 选项”一节第 723 页。

参考中继服务器文档来获得有关建立中继服务器与 MobiLink 之间的安全通信流的信息。请参见“中继服务器” 《MobiLink - 服务器管理》。

管理服务器消息存储库

目录

传输规则

传输规则可用于指定何时进行消息传输以及要传输哪些消息。可以为客户端和服务器都指定传输规则。

管理服务器传输规则是管理服务器消息存储库的重要部分。

服务器传输规则控制从服务器到客户端的消息的行为。服务器传输规则由 MobiLink 服务器处理。它们适用于使用推式通知和不使用通知时。

可通过多种方法来设置服务器传输规则：

- 编写服务器管理请求来设置传输规则。

  请参见“使用服务器管理请求指定传输规则”一节第 189 页。

- 使用 Sybase Central 设置规则。

  请参见“使用 Sybase Central 指定服务器传输规则”一节第 140 页。

- 创建服务器传输规则文件并在启动 MobiLink 服务器时指定它。不建议使用此方法。

  请参见“使用传输规则文件指定服务器传输规则（不建议使用）”一节第 770 页。

服务器传输规则

服务器传输规则控制从服务器到客户端的消息的行为。服务器传输规则由 MobiLink 服务器处理。它们应用于使用推式通知和不使用通知时。

可通过多种方法来设置服务器传输规则：

- 编写服务器管理请求来设置传输规则。

  请参见“使用服务器管理请求指定传输规则”一节第 189 页。

- 使用 Sybase Central 设置规则。

  请参见“使用 Sybase Central 指定服务器传输规则”一节第 140 页。

- 创建服务器传输规则文件并在启动 MobiLink 服务器时指定它。不建议使用此方法。

  请参见“使用传输规则文件指定服务器传输规则（不建议使用）”一节第 770 页。

使用 Sybase Central 指定服务器传输规则

可以在 Sybase Central 中创建和编辑传输规则。

♦ 指定缺省服务器传输规则

1. 启动 Sybase Central：
   - 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
   - 从 [**连接**] 上，选择 [**使用 QAnywhere 11 连接**]。

- 指定一个 [**ODBC 数据源名称**] 或 [**ODBC 数据源文件**]、[**用户 ID**] 和 [**口令**]（如果需要）。单击 [**OK**]。

2. 在 [**服务器消息存储库**] 下，选择数据源名称。
3. 选择 [**文件**] » [**属性**]。
4. 打开 [**传输规则**] 选项卡并选择 [**自定义缺省传输规则**]。
5. 单击 [**新建**] 添加规则。
6. 可通过将条件键入文本字段或从下拉列表中选择 [**消息变量**] 或 [**常量**] 来添加条件。
7. 单击 [**确定**] 退出。

使用服务器管理请求指定服务器传输规则

可以使用服务器管理请求来指定应用于所有用户的缺省服务器传输规则，也可以为每个客户端指定传输规则。

要为服务器指定缺省传输规则，请为客户端 ianywhere.server.defaultClient 设置 ianywhere.qa.server.rules 属性。对于客户端，可使用 ianywhere.qa.server.rules 属性来指定服务器传输规则。

有关使用服务器管理请求指定传输规则的详细信息，请参见“使用服务器管理请求指定传输规则”一节第 189 页。

管理消息归档

档案消息存储库是与服务器消息存储库共存的一组表，它存储所有等待被删除的消息。定期执行的系统进程在消息存储库之间传送消息的方式为：通过删除服务器消息存储库中达到最终状态的所有消息，然后将它们插入到档案消息存储库中。

消息一直保留在档案消息存储库中，直到服务器删除规则将其删除为止。使用档案消息存储库可以最小化同步期间需要过滤的消息数量，从而能够提高服务器消息存储库的性能。请参见“删除规则”一节第 47 页。

使用服务器管理请求

服务器管理请求是从客户端发送到服务器的特殊消息。服务器管理请求包含内容、格式化的 XML，用于指示服务器执行各种功能。

使用以下功能和服务器管理请求来管理服务器消息存储库：

- “刷新客户端传输规则”一节第 176 页
- “取消消息”一节第 176 页
- “删除消息”一节第 177 页

有关使用服务器管理请求的详细信息，请参见：

- “使用服务器管理请求管理服务器消息存储库”一节第 176 页
- “编写服务器管理请求”一节第 174 页
- “服务器管理请求参考”第 693 页

管理客户端消息存储库

目录

监控 QAnywhere 客户端

可以使用服务器管理请求或 Sybase Central 来监控 QAnywhere 客户端。

服务器管理请求可用于获取当前服务器上的客户端列表。此列表包含在服务器上注册的客户端，包括远程客户端、打开的连接器和目标别名。请参见“监控 QAnywhere 客户端”一节第 194 页。

在 Sybase Central 中，可以使用服务器消息存储库的 [**客户端**] 窗格查看当前服务器上的客户端列表。

监控客户端属性

可以使用服务器管理请求或 Sybase Central 来监控 QAnywhere 客户端属性。

服务器管理请求可用于查看为客户端设置了哪些属性。响应仅列出已为客户端设置的属性（非缺省属性）。请参见“监控属性”一节第 195 页。

在 Sybase Central 中，可以使用 QAnywhere [**客户端属性**] 窗口来查看和更改客户端属性。

管理客户端消息存储库属性

客户端消息存储库属性可在客户端应用程序中为每个客户端消息存储库设置。

请参见“在应用程序中管理客户端消息存储库属性”一节第 750 页。

客户端消息存储库属性可在传输规则中使用（以过滤传送到客户端的消息），也可在删除规则中使用（以确定要添加的消息）。

请参见“QAnywhere 传输和删除规则”第 761 页。

客户端消息存储库属性还可在服务器管理消息中指定，并存储在服务器消息存储库中。

请参见“服务器管理请求简介”一节第 172 页。

目标别名

目录

目标别名

目标别名是消息地址和其它目标别名的列表。如果将一条消息发送至某个目标别名，则该消息将被发送至列表中的所有成员。

目标别名的成员可以具有一个与之关联的传送条件。只有匹配该条件的消息才会转发到相应的成员。

示例

定义一个名为 all_clients 的目标别名，其成员为 client1 和 client2。

为 client1 定义以下传送条件：

```
ias_Priority=1
```

为 client2 定义以下传送条件：

```
ias_Priority=9
```

只有优先级为 1 的消息能够发送至 client1，而那些优先级为 9 的消息将被发送至 client2。

创建目标别名

可使用以下方法创建和管理目标别名：

- 服务器管理请求

  请参见“使用服务器管理请求创建目标别名”一节第 190 页。
- Sybase Central

  请参见“使用 Sybase Central”一节第 150 页。

使用 Sybase Central

您可以使用 Sybase Central 来创建或修改目标别名。

♦ 使用 Sybase Central 创建目标别名

1. 启动 Sybase Central：
   - 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
   - 选择 [**连接**] » [**使用 QAnywhere 11 连接**]。
   - 指定一个 [**ODBC 数据源名称**] 或 [**ODBC 数据源文件**]、[**用户 ID**] 和 [**口令**]（如果需要）。
   - 单击 [**确定**]。
2. 选择 [**文件**] » [**新建**] » [**目标别名**]。
3. 在 [**别名**] 字段中，键入别名的名称。
4. 在 [**目标**] 字段中，于每个目标自己的行中键入其名称。

5. 单击 [**确定**]。

连接器

目录

JMS 连接器

Java 消息服务 (JMS) API 可向 Java 应用程序提供消息传递功能。除在 QAnywhere 客户端应用程序间交换消息外，您还可以与支持 JMS 接口的外部消息传递系统交换消息。这可以通过使用一种称为连接器的经过特殊配置的客户端来实现。在 QAnywhere 部署中，外部消息传递系统被设置为类似 QAnywhere 客户端的角色。它有自己的地址和配置。

有关此方法的体系结构的详细信息，请参见“使用外部消息系统进行消息传递的方案”一节 第 7 页。

在服务器群环境中运行带 QAnywhere 消息传递的 MobiLink 时，仅主服务器中的 QAnywhere 连接器启动。如果主服务器失败，QAnywhere 连接器会自动在新的主服务器中启动，这样在外部消息传递系统（如 JMS）中交换数据时，消息排序可以保留下来。有关详细信息，请参见“在服务器群中运行 MobiLink 服务器”一节 《MobiLink - 服务器管理》。

设置 JMS 连接器

以下步骤概述设置带有 JMS 连接器的 QAnywhere 所需的任务（假设已设置了 QAnywhere）。

♦ QAnywhere 应用程序与外部 JMS 系统集成的概述

1. 使用 JMS 管理工具为您的 JMS 系统创建 JMS 队列。QAnywhere 连接器会监听一个 JMS 队列以获得 JMS 消息。如果此队列尚不存在，就必须创建。

   有关如何创建队列的信息，请参见 JMS 产品的文档。

2. 打开 Sybase Central，连接到您的服务器消息存储库。
3. 选择 [**文件**] » [**新建**] » [**连接器**]。
4. 单击 [**JMS**]，然后在 [**您使用的是哪一种 JMS 系统**] 列表中选择要使用的 Web 服务器的类型。单击 [**下一步**]。
5. 在 [**连接器名称**] 页中：
   - 在 [**连接器名称**] 字段中，键入 QAnywhere 客户端用来寻址连接器的连接器地址。请参见“向 JMS 连接器发送 QAnywhere 消息”一节第 158 页。
   - 在 [**接收器目标**] 字段中，键入连接器用来监听从 JMS 到 QAnywhere 客户端的消息的队列名。
   - 单击 [**下一步**]。
6. 在 [**JNDI 设置**] 页中：
   - 在 [**JNDI 工厂**] 字段中，键入用于访问外部 JMS JNDI 名称服务的工厂名称。
   - 在 [**名称服务 URL**] 字段中，键入用于访问 JMS JNDI 名称服务的 URL。
   - 在 [**用户名**] 字段中，键入用于连接到外部 JMS JNDI 名称服务的验证名称。
   - 在 [**口令**] 字段中，键入用于连接到外部 JMS JNDI 名称服务的验证口令。
   - 单击 [**下一步**]。
7. 在 [**JMS 队列设置**] 页中：
   - 在 [**队列工厂**] 字段中，键入外部 JMS 提供程序队列的工厂名称。
   - 在 [**用户名**] 字段中，键入用于连接到外部 JMS 队列连接的用户 ID。
   - 在 [**口令**] 字段中，键入用于连接到外部 JMS 队列连接的口令。
   - 单击 [**下一步**]。
8. 在 [**JMS 主题设置**] 页中：
   - 在 [**主题工厂**] 字段中，键入外部 JMS 提供程序的主题工厂名称。
   - 在 [**用户名**] 字段中，键入用于连接到外部 JMS 主题连接的用户 ID。
   - 在 [**口令**] 字段中，键入用于连接到外部 JMS 主题连接的口令。
   - 单击 [**完成**]。
9. 单击 [**确定**]。

10. 使用到服务器消息存储库的连接及 -sl java 选项启动 MobiLink 服务器。请参见“启动 MobiLink 服务器以进行 JMS 集成”一节第 156 页。
11. 要在 JMS 连接器上设置其它选项，请右击刚刚创建的连接器并选择 [属性]；也可使用服务器管理请求。

    有关可用属性的列表，请参见“配置 JMS 连接器属性”一节第 157 页。

    有关如何使用服务器管理请求设置连接器属性的信息，请参见“使用服务器管理请求管理连接器”一节第 179 页。

♦ **发送消息**

1. 要从您的 QAnywhere 系统中的应用程序向外部消息传递系统发送消息，请创建一条 QAnywhere 消息并将其发送到 ***connector-address\JMS-queue-name***。

   请参见“向 JMS 连接器发送 QAnywhere 消息”一节第 158 页。
2. 从外部消息传递系统向您的 QAnywhere 系统中的应用程序发送消息：
   - 创建 JMS 消息。
   - 将 ias_ToAddress 属性设置为 QAnywhere ***id\queue***（其中 *id* 是客户端消息存储库的 ID，*queue* 是应用程序队列名称）。
   - 将消息放入 JMS 队列。

   请参见“为传递到 QAnywhere 的 JMS 消息指定地址”一节第 160 页。

其它入门资源

- QAnywhere JMS 示例安装在 *samples-dir\QAnywhere\jms* 中。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）
- 可在 QAnywhere 新闻组上发布问题：ianywhere.public.sqlanywhere.qanywhere
- 有关在服务器群环境中设置消息传递的详细信息，请参见“在服务器群中运行 MobiLink 服务器”一节 《MobiLink - 服务器管理》。

启动 MobiLink 服务器以进行 JMS 集成

要与支持 JMS 接口的外部消息传递系统交换消息，必须使用以下选项启动 MobiLink 服务器 (mlsrv11)：

- **-c *connection-string*** 连接到服务器消息存储库。

  请参见“-c 选项”一节 《MobiLink - 服务器管理》。
- **-m** 启用 QAnywhere 消息传递。
- **-sl java (-cp "*jarfile*.jar")** 添加使用外部 JMS 提供程序所需的客户端 jar 文件。

  请参见“-sl java 选项”一节 《MobiLink - 服务器管理》。

示例

下面的示例使用名为 *jmsclient.jar* 的 JMS 客户端库（在当前工作目录中）和用作消息存储库的 QAnywhere 示例数据库来启动 MobiLink 服务器。该命令应全部在一行上输入。

```
mlsrv11 -sl java(-cp
"jmsclient.jar") -m -c "dsn=QAnywhere 11.0 Demo" ...
```

配置 JMS 连接器属性

可使用 JMS 连接器属性来指定同 JMS 系统连接的信息。这些属性将连接器配置到第三方 JMS 消息传递系统（如 BEA WebLogic 或 Sybase EAServer）。

可在以下几个位置设置和/或查看属性：

- Sybase Central [**连接器向导**]。

  请参见“设置 JMS 连接器”一节第 155 页。
- Sybase Central [**连接器属性**] 窗口。
- 服务器管理请求。

  请参见“创建和配置连接器”一节第 179 页。
- ml_qa_global_props MobiLink 系统表。

  请参见“ml_qa_global_props”一节 《MobiLink - 服务器管理》。

有关所有 JMS 连接器属性的列表，请参见“JMS 连接器属性”一节第 756 页。

配置多个连接器

通过为每个 JMS 系统定义一个 JMS 连接器，QAnywhere 可以连接到多个 JMS 消息系统。在配置的连接器中只有 ianywhere.connector.address 属性的值必须唯一。

ianywhere.connector.address 属性是 QAnywhere 客户端必须指定的地址前缀，用于为传递到 JMS 系统的消息指定地址。

另请参见

- “向 JMS 连接器发送 QAnywhere 消息”一节第 158 页
- “配置 JMS 连接器属性”一节第 157 页
- “创建和配置连接器”一节第 179 页

向 JMS 连接器发送 QAnywhere 消息

通过将地址设置为下列值，QAnywhere 客户端可向 JMS 系统发送消息：

connector-address\JMS-queue-name

connector-address 是连接器属性 ianywhere.connector.address 的值，而 *JMS-queue-name* 是通过 Java 命名和目录接口来查找 JMS 队列或主题的名称。

如果 *JMS-queue-name* 包含反斜线，则必须使用另一个反斜线来转义此反斜线。例如，上下文 ss 中名为 qq 的队列应指定为 `ss\\qq`。

```
// C# example
QAMessage msg; QAManager mgr;
... mgr.PutMessage( @"ianywhere.connector.wsmqfs\ss\\qq",msg
);

// C++ example
QAManagerBase *mgr; QATextMessage *msg; ... mgr->putMessage(
"ianywhere.connector.easerver\\ss\\\\qq", msg );
```

示例

例如，如果将 ianywhere.connector.address 设置为 ianywhere.connector.easerver，将 JMS 队列名称设置为 myqueue，则设置该地址的代码将如下所示：

```
// C# example
QAManagerBase mgr; QAMessage msg; // Initialize the manager. ... msg =
mgr.CreateTextMessage(); // Set the message content. ...
mgr.PutMessage(@"ianywhere.connector.easerver\myqueue", msg );

// C++ example
QAManagerBase *mgr; QATextMessage *msg; // Initialize the manager. ... msg =
mgr.createTextMessage(); // Set the message content. ... mgr->putMessage(
"ianywhere.connector.easerver\\myqueue", msg );
```

另请参见

- “QAnywhere 消息地址”一节第 63 页
- “配置 JMS 连接器属性”一节第 157 页

从 JMS 连接器向 QAnywhere 客户端发送消息

QAnywhere 消息会自然地映射到 JMS 消息。

QAnywhere 消息内容

QAnywhere	JMS	注释
QATextMessage	javax.jms.TextMessage	以 Unicode 格式复制消息文本
QABinaryMessage	javax.jms.BytesMessage	精确复制消息字节

QAnywhere 内置标头

下表介绍了内置标头的映射。在 C++ 和 JMS 中，这些是方法名；例如，Address 在 QAnywhere 中称为 getAddress() 或 setAddress()，在 JMS 中则称为 getJMSDestination() 或 setJMSDestination()。在 .NET 中，这些是属性，其名称则如下表所示；例如，Address 为 Address。

QAnywhere	JMS	注释
Address	JMSDestination 和 JMS 属性 ias_ToAddress	如果目标包含反斜线，则必须使用另一个反斜线来转义此反斜线。 只有地址的 JMS 部分映射到 Destination。当消息循环回 QAnywhere 时，很少有附加的 QAnywhere 地址后缀。此项会放入 ias_ToAddress。
Expiration	JMSExpiration	
InReplyToID	N/A	不映射。
MessageID	N/A	不映射。
Priority	JMSPriority	
Redelivered	N/A	不映射。
ReplyToAddress	JMS 属性 ias_ReplyToAddress	映射到 JMS 属性。
连接器的 xjms.receiveDestination 属性值	JMSReplyTo	ReplyTo 设置为由连接器用来接收 JMS 消息的 Destination。
Timestamp	N/A	不映射。

QAnywhere	JMS	注释
N/A	JMSTimestamp	将 JMS 消息映射到 QAnywhere 消息时，QAnywhere 消息的 JMSTimestamp 属性设置为 JMS 消息的 JMSTimestamp。
Timestamp	N/A	将 QAnywhere 消息映射到 JMS 消息时，JMS 消息的 JMSTimestamp 设置为创建 JMS 消息的时间。

QAnywhere 属性

QAnywhere 属性全部自然地映射到 JMS 属性，在此过程中属性类型不变，但也有以下例外：如果 QAnywhere 消息有一个名为 JMSType 的属性，则映射到 JMS 标头属性 JMSType。

为传递到 QAnywhere 的 JMS 消息指定地址

JMS 客户端向 QAnywhere 客户端发送消息的方法是：将 JMS 消息属性 ias_ToAddress 设置为 QAnywhere 地址，然后将消息发送到与连接器属性 xjms.receiveDestination 对应的 JMS 目标。

请参见“QAnywhere 消息地址”一节第 63 页。

示例

例如，向 QAnywhere 地址 "qaddr" 发送消息（其中 xjms.receiveDestination 的连接器设置为 "qanywhere_receive"）：

```
import javax.jms.*;
try {
    QueueSession session;
    QueueSender sender;
    TextMessage mgr;
    Queue connectorQueue;

    // Initialize the session.
    connectorQueue = session.createQueue("qanywhere_receive");
    sender = session.createSender( connectorQueue );
    msg = session.createTextMessage();
    msg.setStringProperty("ias_ToAddress", "qaddr");

    // Set the message content.
    sender.send(msg);
}
catch( JMSException e ) {
    // Handle the exception.
}
```


将 JMS 消息映射到 QAnywhere 消息

JMS 消息会自然地映射到 QAnywhere 消息。

JMS 消息内容

JMS	QAnywhere	注释
javax.jms.TextMessage	QATextMessage	以 Unicode 格式复制消息文本
javax.jms.BytesMessage	QABinaryMessage	精确复制消息字节
javax.jms.StreamMessage	N/A	不支持
javax.jms.MapMessage	N/A	不支持
javax.jms.ObjectMessage	N/A	不支持

JMS 内置标头

下表介绍了内置标头的映射。在 C++ 和 JMS 中，这些是方法名；例如，Address 在 QAnywhere 中称为 getAddress() 或 setAddress()，在 JMS 中则称为 getJMSDestination() 或 setJMSDestination()。在 .NET 中，这些是属性，其名称则如下表所示；例如，Address 为 Address。

JMS	QAnywhere	注释
JMS Destination	N/A	JMS Destination 必须设置为在连接器属性 xjms.receiveDestination 中指定的队列。
JMS Expiration	Expiration	
JMS CorrelationID	InReplyToID	
JMS MessageID	N/A	不映射。
JMS Priority	Priority	
JMS Redelivered	N/A	不映射。
JMS ReplyTo 和连接器的 ianywhere.connector.address 属性值	ReplyToAddress	连接器地址与 JMS ReplyTo Destination 名称连接在一起，中间用 '\' 分隔。
JMS DeliveryMode	N/A	不映射。
JMS Type	QAnywhere 消息属性 JMSType	

JMS	QAnywhere	注释
JMS Timestamp	N/A	不映射。

JMS 属性

JMS 属性全部自然地映射到 QAnywhere 属性，在此过程中属性类型不变，但也有一些例外。QAnywhere Address 属性是根据 JMS 消息属性 ias_ToAddress 的值进行设置的。如果设置了 JMS 消息属性 ias_ReplyToAddress，则 QAnywhere ReplyToAddress 以此值作为后缀，中间用 '\' 分隔。

Web 服务连接器

Web 服务连接器监听发送至特定地址的 QAnywhere 消息，然后在消息到达时调用 Web 服务。Web 服务响应作为 QAnywhere 消息被发回至源客户端。所有发送至 Web 服务连接器的消息应使用由 QAnywhere WSDL 编译器所产生的代理类来创建。

在服务器群环境中运行带 QAnywhere 消息传递的 MobiLink 时，仅主服务器中的 QAnywhere 连接器启动。如果主服务器失败，QAnywhere 连接器会自动在新的主服务器中启动，这样在外部消息传递系统（如 JMS）中交换数据时，消息排序可以保留下来。有关详细信息，请参见“在服务器群中运行 MobiLink 服务器”一节 《MobiLink - 服务器管理》 。

设置 Web 服务连接器

♦ **创建 Web 服务连接器**

1. 打开 Sybase Central，连接到您的服务器消息存储库。
2. 选择 [**文件**] » [**新建**] » [**连接器**]。
3. 单击 [**Web 服务**]。单击 [**下一步**]。
4. 在 [**连接器名称**] 字段中，键入 QAnywhere 客户端用来寻址连接器的连接器地址。单击 [**下一步**]。
5. 在 [**URL**] 字段中，键入 Web 服务的 URL（例如，*http://localhost:8080/qanyserv/F2C*）。单击 [**下一步**]。

   您可选择指定超时期（以毫秒为单位），这样，如果 Web 服务在指定时间量内未响应，系统会取消请求。此处设置属性 webservice.socket.timeout。
6. 在 [**HTTP 参数**] 页中，单击 [**Web 服务必须通过代理进行访问**]，然后完成以下字段：
   - 在 [**HTTP 用户名**] 字段中，键入用户名。此处设置属性 webservice.http.authName。
   - 在 [**HTTP 口令**] 字段中，键入用户口令。此处设置属性 webservice.http.password.e。
   - 在 [**代理主机名**] 字段中，键入主机名。如果要指定此属性，则必须指定 webservice.http.proxy.port 属性。
   - 在 [**代理端口**] 字段中，键入要连接到的代理服务器端口。如果要指定此属性，则必须指定 webservice.http.proxy.host 属性。
   - 在 [**代理用户名**] 字段中，键入代理服务器要求验证时将使用的代理用户名。如果要指定此属性，则还必须指定 webservice.http.proxy.password.e 属性。
   - 单击 [**完成**]。
7. 要在 Web 服务连接器上设置其它选项，可右击刚刚创建的连接器并选择 [**属性**]；也可使用服务器管理请求。

   有关可用属性的列表，请参见“Web 服务连接器属性”一节第 164 页。

   有关使用服务器管理请求的信息，请参见“使用服务器管理请求管理连接器”一节第 179 页。

Web 服务连接器属性

使用 Web 服务连接器属性可指定 Web 服务的连接信息。可在 Sybase Central [连接器向导] 中设置这些属性。

请参见“Web 服务连接器”一节第 163 页。

可在 Sybase Central 的 [**连接器属性**] 窗口中，或在 ml_qa_global_props MobiLink 系统表中查看 Web 服务连接器属性。

要打开 [**连接器属性**] 窗口，请在 Sybase Central 中右击连接器，然后选择 [**属性**]。

♦ 查看 Web 服务属性

1. 打开 Sybase Central，连接到您的服务器消息存储库。
2. 在左窗格中的 [**服务器消息存储库**] 下，选择数据源的名称。
3. 在右窗格中，选择 [**连接器**] 选项卡，然后选择 Web 服务连接器的名称。
4. 选择 [**文件**] » [**属性**]

有关 ml_qa_global_props MobiLink 系统表的详细信息，请参见“ml_qa_global_props”一节《MobiLink - 服务器管理》。

Web 服务连接器属性

- **ianywhere.connector.nativeConnection** 实现连接器的 Java 类。它仅供 QAnywhere 内部使用，并且不应删除或修改。
- **ianywhere.connector.id（不建议使用）** 唯一标识连接器的标识符。缺省值为 ianywhere.connector.address。
- **ianywhere.connector.address** QAnywhere 客户端用来标识连接器位置的连接器地址。此地址还用作前缀，作用于显示在此连接器的 MobiLink 消息窗口中所有记录的错误、警告和信息性消息。

  在 Sybase Central 中，您可在 [**连接器向导**] 的 [**连接器名称**] 页的 [**连接器名称**] 字段中设置此属性。
- **ianywhere.connector.compressionLevel** 从 Web 服务接收的消息的缺省压缩因子。压缩用 0 到 9 之间的一个整数表示，0 表示无压缩，9 表示最大压缩。

  在 Sybase Central 中，您可在 [**连接器属性**] 窗口的 [**常规**] 选项卡上的 [**压缩级别**] 部分中设置此属性。
- **ianywhere.connector.logLevel** MobiLink 消息窗口和 MobiLink 服务器消息日志文件中显示的连接器信息的信息量。日志级别的值如下：
  - **1** 记录错误消息。
  - **2** 记录错误和警告消息。
  - **3** 记录错误、警告和信息性消息。
  - **4** 记录错误、警告、信息性消息和调试消息。

在 Sybase Central 中，您可在 [**连接器属性**] 窗口的 [**常规**] 选项卡上的 [**记录级别**] 部分设置此属性。

- **ianywhere.connector.outgoing.retry.max** 从 QAnywhere 到外部消息传递系统进行消息传递的缺省重试次数。缺省值为 5。指定 0 可使连接器始终进行重试。

  在 Sybase Central 中，可通过在 [**连接器属性**] 窗口的 [**属性**] 选项卡下单击 [**新建**] 设置此属性。
- **ianywhere.connector.startupType** 启动类型可以是自动、手工或禁止。
- **webservice.http.authName** 如果 Web 服务要求 HTTP 验证，则使用此属性来指定用户名。
- **webservice.http.password.e** 如果 Web 服务要求 HTTP 验证，则使用此属性来指定口令。
- **webservice.http.proxy.authName** 如果代理服务器要求验证，则使用此属性来设置代理用户名。如果要指定此属性，则还必须指定 webservice.http.proxy.password.e 属性。
- **webservice.http.proxy.host** 如果 Web 服务必须通过 HTTP 代理才能访问，则使用此属性指定主机名。如果要指定此属性，则必须指定 webservice.http.proxy.port 属性。
- **webservice.http.proxy.password.e** 如果代理服务器要求验证，则使用此属性来设置代理口令。如果要指定此属性，则还必须指定 webservice.http.proxy.authName 属性。
- **webservice.http.proxy.port** 连接至代理服务器的端口。如果要指定此属性，则必须指定 webservice.http.proxy.host 属性。

向 Web 服务连接器发送消息

通过移动 Web 服务 API 向 Web 服务连接器发送消息。可使用 ianywhere.qanywhere.ws.WSBase 类中的 setProperty 方法将 WS_CONNECTOR_ADDRESS 属性设置为 Web 服务连接器的 ID。请参见“WSBase 类”一节第 610 页。

例如，当指定了 CurrencyConvertor 示例中的以下代码行时，用于发出 Web 服务请求的 Web 服务 API 通过 Web 服务连接器将这些请求作为消息发送出去。

```
service.setProperty(
    "WS_CONNECTOR_ADDRESS",
    "ianywhere.connector.currencyconvertor\\" );
```


教程：使用 JMS 连接器

JMS 连接器提供 JMS 消息系统和 QAnywhere 之间的连接。在本教程中，您在 JMS 客户端应用程序和 QAnywhere 客户端应用程序间发送消息。

必需的软件

- SQL Anywhere 11
- Java 软件开发工具包
- JMS 连接器

能力和经验

您需要：

- 熟悉 Java
- 具备配置 JMS 连接器的基础知识

目标

您将掌握并熟悉以下内容：

- 配置 JMS 连接器以便与示例 QAnywhere 应用程序通信
- 在 JMS 消息系统和示例 QAnywhere 应用程序间发送消息

重要概念

本节通过以下步骤在 JMS 消息系统和使用 SQL Anywhere 示例数据库的 QAnywhere 之间建立连接：

- 准备 JMS 连接器以发送和接收消息
- 运行 QAnywhere 服务器和客户端组件，以及 JMS 客户端
- 从 QAnywhere 客户端发送消息到 JMS 客户端，或相反

建议阅读的背景知识

有关使用 JMS 连接器的详细信息，请参见“连接器”第 153 页。

第 1 课：设置客户端和服务器组件

♦ **准备 JMS 提供程序**

1. 请参考 JMS 服务器文档来启动服务器。
2. 在 JMS 服务器中创建以下队列：
   - **testmessage** TestMessage 示例使用此队列名来监听消息。
   - **qanywhere_receive** QAnywhere JMS 连接器使用此队列名来监听消息。

   您可能需要在创建队列后重新启动服务器。有关详细信息，请参见 JMS 服务器文档。

♦ 启动 QAnywhere 客户端和服务器组件

1. 使用 Sybase Central 为您的 JMS 系统创建 QAnywhere JMS 连接器。请参见“设置 JMS 连接器”一节第 155 页。
2. 在命令提示符处，运行以下命令：

   ```
   mlsrv11 -m -c "dsn=QAnywhere 11 Demo" -sl
   java(-cp JMS-client-jar-files) -vcrs
   -zu+
   ```

   其中 *JMS-client-jar-files* 是用分号分隔的、访问 JMS 服务器所需的 jar 文件的列表。有关详细信息，请参见 JMS 服务器文档。

   MobiLink 服务器启动以进行消息传递。
3. 从 [开始] 菜单，选择 [程序] » [**SQL Anywhere 11**] » [**QAnywhere**] » [**使用 SQL Anywhere 的教程**] » [**QAnywhere Agent For SQL Anywhere -- saclient1**]。

   QAnywhere Agent 装载。
4. 从 [开始] 菜单，选择 [程序] » [**SQL Anywhere 11**] » [**QAnywhere**] » [**使用 SQL Anywhere 的教程**] » [**TestMessage - saclient1**]。

   QAnywhere 示例应用程序装载。

♦ 启动 JMS 版本的 TestMessage 客户端

1. 在命令提示符处，运行以下命令：

   ```
   edit samples-dir/QAnywhere/JMS/TestMessage/build.bat
   ```
2. 检查 *build.bat* 文件中的代码并确保 JMS 服务器文件的路径正确。

   例如，如果使用 EAServer，则缺省设置在 **easerver** 标题下定义：

   ```
   :easerver
   REM For EAServer, compile with the following JAR files
   SET easerver_install=c:\program files\sybase\easerver6
   SET jmsjars=%easerver_install%\lib\eas-client-15.jar
   goto build_app
   ```

   如果 EAServer 不在 *c:\program files\sybase\easerver6* 目录中，则更新 **easerver_install** 变量以使其指向正确的安装目录。请确保 **jmsjars** 变量指向 JMS 服务器 jar 文件的正确位置。

   如果未列出 JMS 服务器，则使用在批处理文件的开头附近定义的 **custom** 标头设置来自定义 JMS 文件路径位置。

   完成后，保存更改并退出编辑器。
3. 在命令提示符处，运行以下命令可编译 JMS TestMessage 客户端：

   ```
   build.bat JMS-server-name
   ```

   其中 *JMS-server-name* 是 JMS 服务器的名称，在 *build.bat* 文件中以标头名称表示。可接受的值有 **easerver**、**fioranomq**、**jboss**、**tibco**、**weblogic** 和 **custom**。缺省情况下，*build.bat* 使用 **easerver**。

4. 在命令提示符处，运行以下命令：

   ```
   edit samples-dir/QAnywhere/JMS/TestMessage/run.bat
   ```

5. 检查 *run.bat* 文件中的代码并确保 JMS 服务器文件的路径正确。

   例如，如果使用 EAServer，则缺省设置在 **easerver** 标题下定义：

   ```
   :easerver
   REM For EAServer, compile with the following JAR files
   SET easerver_install=c:\program files\sybase\easerver6
   SET jmsjars=%easerver_install%\lib\eas-client-15.jar
   goto build_app
   ```

   如果 EAServer 不在 *c:\program files\sybase\easerver6* 目录中，则更新 **easerver_install** 变量以使其指向正确的安装目录。请确保 **jmsjars** 变量指向 JMS 服务器 jar 文件的正确位置。

   如果未列出 JMS 服务器，则使用在批处理文件的开头附近定义的 **custom** 标头设置来自定义 JMS 文件路径位置。

   完成后，保存更改并退出编辑器。

6. 在命令提示符处，执行以下命令可运行 JMS TestMessage 客户端：

   ```
   run.bat JMS-server-name
   ```

   其中 *JMS-server-name* 是 JMS 服务器的名称，在 *build.bat* 文件中以标头名称表示。可接受的值有 **easerver**、**fioranomq**、**jboss**、**tibco**、**weblogic** 和 **custom**。缺省情况下，*build.bat* 使用 **easerver**。

7. 将 JMS TestMessage 窗口移到您屏幕的右侧，置于现有的 [**TestMessage -- saclient1**] 窗口下面。

第 2 课：从 JMS 客户端向 QAnywhere 客户端发送消息

♦ **从 JMS 客户端向 QAnywhere 客户端发送消息**

1. 从 JMS TestMessage 的 [**Message**] 菜单中，选择 [**New**]。
2. 在 [**Destination ID**] 字段中，键入 **saclient1**。
3. 使用示例文本完成 [**Subject**] 和 [**Message**] 字段。
4. 单击 [**Send**]。

   会出现一个窗口，指示消息已收到。

第 3 课：从 QAnywhere 客户端向 JMS 客户端发送消息

♦ **查找 QAnywhere JMS 连接器的名称**

1. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 选择 [**连接**] » [**使用 QAnywhere 11 连接**]。
3. 单击 [**ODBC 数据源名称**]。

4. 单击 [**浏览**] 并选择 [**QAnywhere 11 Demo**]。
5. 单击 [**确定**]。
6. 单击 [**确定**]。
7. 选择 [**连接器**] 选项卡。

   右窗格显示所有活动 JMS 连接器的列表。
8. 检查名称字段。

   在列表中只应该存在一个活动的 QAnywhere JMS 连接器。该连接器的名称显示在名称字段下。

♦ 从 QAnywhere 客户端向 JMS 客户端发送消息

1. 在 [**saclient1 - TestMessage**] 窗口中，单击 [**Message**] » [**New**]。
2. 在 [**Destination ID**] 字段中，键入 JMS 系统的名称。
3. 在 [**Subject**] 和 [**Message**] 字段中，键入示例文本。
4. 单击 [**Send**]。

   会出现一个窗口，指示消息已收到。

教程清理

关闭 TestMessage 客户端、QAnywhere 代理和 MobiLink 服务器。

♦ 关闭所有应用程序

1. 要关闭 JMS TestMessage 客户端应用程序，请选择 [**File**] » [**Exit**]。
2. 要关闭 [**saclient1 - TestMessage Client**] 窗口，请选择 [**File**] » [**Exit**]。
3. 要关闭 [**saclient1 - QAnywhere Agent**] 窗口和 MobiLink 服务器，请在相应的窗口中选择 [**关闭**]。
4. 要断开与 JMS 连接器的连接，请参见 JMS 服务器文档。

服务器管理请求

目录

服务器管理请求简介

QAnywhere 客户端应用程序可以将称为**服务器管理请求**的特殊消息发送到服务器。这些消息包含格式化为 XML 并要发送到 QAnywhere 系统队列的内容。它们需要一个特殊的验证字符串。服务器管理请求可以执行许多功能，例如：

- 启动和停止连接器和 Web 服务。

  请参见“打开连接器”一节第 181 页和“关闭连接器”一节第 181 页。
- 监控连接器状态。

  请参见“监控连接器”一节第 182 页。
- 设置和刷新客户端传输规则。

  请参见“使用服务器管理请求指定传输规则”一节第 189 页。
- 监控消息状态。

  请参见“监控 QAnywhere”一节第 193 页。
- 在服务器上设置、更新、删除和查询客户端消息存储库属性。

  请参见“使用服务器管理请求设置服务器属性”一节第 187 页。
- 取消消息。

  请参见“取消消息”一节第 176 页。
- 查询活动客户端、消息存储库属性和消息。

指定服务器管理请求的地址

缺省情况下，服务器管理请求必须定址于 **ianywhere.server\system.**。要更改此地址的客户端 ID 部分，请设置 ianywhere.qa.server.id 属性并重新启动服务器。例如，如果 ianywhere.qa.server.id 属性设置为 myServer，则服务器管理请求定址为 myServer\system。

有关设置 ianywhere.qa.server.id 属性的详细信息，请参见“服务器属性”一节第 753 页。

有关指定 QAnywhere 消息的地址的详细信息，请参见“发送 QAnywhere 消息”一节第 66 页。

有关系统队列的详细信息，请参见“系统队列”一节第 64 页。

示例

以下是一个消息详细信息请求的示例。它生成一个报告，该报告显示当前在服务器上优先级为 9 的所有消息的消息 ID、状态和目标地址。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <MessageDetailsRequest>
      <request>
         <requestId>testRequest</requestId>
         <condition>
            <priority>9</priority>
         </condition>
         <status/>
         <address/>
```



```
      </request>
   </MessageDetailsRequest>
</actions>
```

以下示例采用 C# 编写。它为客户端设置这样的服务器端传输规则，即优先级大于 4 时，来自服务器的消息仅传送至称为 someClient 的客户端。

```
QAManager mgr = ...; // Initialize the QAManager
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "ias_ServerPassword", "QAnywhere" );

// Indenting and newlines are just for readability
msg.Text = "<?xml version="1.0" encoding="UTF-8"?>\n"
+ "<actions>\n"
+ " <SetProperty>\n"
+ "  <prop>\n"
+ "   <client>someClient</client>\n"
+ "   <name>ianywhere.qa.server.rules</name>\n"
+ "   <value>ias_Priority &gt; 4</value>\n"
+ "  </prop>\n"
+ " </SetProperty>\n"
+ " <RestartRules>\n"
+ "  <client>someClient</client>\n"
+ " </RestartRules>\n"
+ "</actions>\n";

mgr.PutMessage( @"ianywhere.server\system", msg );
```

验证服务器管理请求

ianywhere.qa.server.password.e 服务器属性用于指定一个口令，使用该口令来验证服务器管理请求。如果未设置该属性，则口令为 QAnywhere。请参见“服务器属性”一节第 753 页。

编写服务器管理请求

服务器管理请求包含格式化为 XML 的内容。

> **注意**
> 不能在服务器管理请求内容中使用如 > 或 < 这样的符号。而要改用 > 和 <。

每种类型的服务器管理请求都包含自己的 XML 标记。例如，要关闭连接器，可使用 <CloseConnector> 标记。

每个服务器管理请求都以 <actions> 标记开头。

actionsResponseId 标记

actionsResponseId 标记作为 <actions> 标记的子标记，可用于跟踪和报告 <actions> 标记中操作的进度。处理 <action> 标记时，服务器会创建一份报告。

报告包含 <actionsResponseId> 标记的 ID 和由请求生成的错误消息。报告创建之后，即被发送到服务器管理请求的回复地址。

下面是使用 actionsResponseId 标记的服务器管理请求示例。

```
<?xml version="1.0" encountered="UTF-8"?>
<actions>
  <actionsResponseId>myActionID</actionsResponseId>
  <MessageDetailsRequest>
    <request>
      <requestId>testRequest</requestId>
      <condition>
        <priority>9</priority>
      </condition>
      <status/>
      <address/>
    </request>
  </MessageDetailsRequest>
</actions>
```

下面是 actionsResponseId 报告示例，其中 myActionId 请求未生成错误。

```
<?xml version="1.0" encoding="UTF-8"?>
<ActionsResponse>
<actionsResponseId>myActionId</actionsResponseId>
<error/>
</ActionsResponse>
```

档案消息存储库请求

要查看档案消息存储库中消息的详细信息，请将 <archived> 标记用作 <condition> 标记的子标记。如果忽略此标记，报告只包含服务器消息存储库发出的消息。

要确定档案消息存储库中是否存在某条消息，请将 <archived> 标记用作 <request> 标记的子标记。

示例

如果 testRequest 在档案消息存储库中则以下请求返回真，如果 testRequest 在服务器消息存储库中则返回假。

```
<request>
   <requestID>testRequest</requestID>
   <status/>
   <archived/>
</request>
```

创建目标别名

可以使用服务器管理请求创建和修改目标别名。请参见“使用服务器管理请求创建目标别名”一节第 190 页。

有关目标别名的详细信息，请参见“目标别名”一节第 150 页。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

使用服务器管理请求管理服务器消息存储库

可以使用服务器管理请求管理服务器消息存储库。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

刷新客户端传输规则

当服务器端的客户端传输规则发生变化时，必须刷新对应的客户端规则。可以在服务器管理请求中通过设置 ianywhere.qa.server.rules 属性更改客户端传输规则。

RestartRules 标记包含单个客户端标记，此标记指定要刷新的客户端的名称。

<RestartRules> 子标记	说明
<client>	要刷新传输规则的客户端的名称。

示例

服务器 XML 需要指定新传输规则属性，然后使用 RestartRules 标记重新启动规则处理。例如，以下 XML 将客户端 myclient 的服务器端传输规则更改为 `auto = ias_Priority > 4`。请注意 XML 中 ">" 的正确编码。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
      <prop>
        <client>myclient</client>
        <name>ianywhere.qa.server.rules</name>
        <value>auto = ias_Priority &gt; 4</value>
      </prop>
    </SetProperty>
    <RestartRules>
        <client>myclient</client>
    </RestartRules>
</actions>
```

取消消息

可以创建取消服务器消息存储库中的消息的服务器管理请求。可以创建一次性取消请求，也可以将取消请求安排为自动发生。还可以选择生成一个报告，其中详细记录已被取消的消息。

只有在激活请求时消息处于非最终状态且尚未传送给接收者的情况下，才能取消消息。

<CancelMessageRequest> 子标记	说明
<request>	有关特定请求的分组信息。指定多个 <request> 标记等效于分别发送多个服务器管理请求。

<Request> 子标记	说明
<condition>	包括要取消的消息的分组条件。请参见“Condition 标记”一节第 694 页。
<persistent>	指定请求应在服务器数据库中持久保留（这样即使服务器重新启动也可取消消息）。仅与调度一同使用。
<requestId>	为包括在作为该请求的结果而生成的每个报告中的请求指定唯一标识符。对此字段使用不同值可以使多个请求同时处于活动状态。使用同一请求 ID 可以使客户端覆盖或删除活动请求。
<replyAddr>	作为该请求的结果而生成的每个报告的返回地址。如果省略此标记，则报告的缺省返回地址为源消息的返回地址。
<report>	使得每次激活请求时都发送报告。要在每次激活请求时都发送报告，请在 <request> 标记内放置空 <report> 标记。
<schedule>	指定应按调度生成报告。请参见“服务器管理请求父标记”一节第 694 页。

示例

此请求取消服务器上地址为 ianywhere.connector.myConnector\deadqueue 的消息：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <CancelMessageRequest>
  <request>
   <requestId>cancelRequest</requestId>
   <condition>
    <customRule>ias_Address='ianywhere.connector.myConnector\deadqueue'</
customRule>
   </condition>
  </request>
 </CancelMessageRequest>
</actions>
```

删除消息

要指定服务器上的清理策略，请将特殊客户端 ianywhere.server.deleteRules 的 ianywhere.qa.server.deleteRules 属性设置为可控制从该服务器中删除哪些消息的一个或多个规则。

以下示例更改消息清理策略以删除到期和取消的消息：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>ianywhere.server.deleteRules</client>
     <name>ianywhere.qa.server.deleteRules</name>
     <value>auto = ias_Status in ( ias_ExpiredStatus, ias_CancelledStatus )
and ias_TransmissionStatus = IAS_TRANSMITTED</value>
 </prop>
    </SetProperty>
    <RestartRules>
 <client>ianywhere.server.deleteRules</client>
    </RestartRules>
</actions>
```


使用服务器管理请求管理连接器

可以使用服务器管理请求创建、配置、删除、启动、停止和监控连接器。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

另请参见

- “连接器”第 153 页
- “Web 服务连接器”一节第 163 页

创建和配置连接器

要创建连接器，请使用 <SetProperty> 添加属性，然后使用 <OpenConnector>。

示例

在以下示例中，服务器管理请求首先设置多个相关属性并将它们与客户端 ianywhere.connector.jboss（它是新连接器的客户端 ID）相关联。JMS 特定的属性设置为可指示连接本地 JBOSS JMS 服务器的连接器。然后使用 OpenConnector 标记启动连接器。请注意，如果尚未使用 JMS 客户端的相关 jar 文件启动 MobiLink 服务器，则连接器不会启动。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.jms.NativeConnectionJMS</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>ianywhere.connector.address</name>
  <value>ianywhere.connector.jboss</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.jndi.factory</name>
  <value>org.jnp.interfaces.NamingContextFactory</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.jndi.url</name>
  <value>jnp://0.0.0.0:1099</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.topicFactory</name>
  <value>ConnectionFactory</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.queueFactory</name>
  <value>ConnectionFactory</value>
   </prop>
```

```
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.receiveDestination</name>
  <value>qanywhere_receive</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.deadMessageDestination</name>
  <value>qanywhere_deadMessage</value>
   </prop>
    </SetProperty>
    <OpenConnector>
        <client>ianywhere.connector.jboss</client>
    </OpenConnector>
</actions>
```

修改连接器

要修改连接器，请关闭连接器，使用 <SetProperty> 标记更改属性，然后打开连接器。

示例

在以下示例中，连接器的记录级别更改为 4。关闭 ID 为 ianywhere.connector.jboss 的连接器；连接器属性 logLevel 更改为 4，然后使用新日志级别重新打开连接器。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <CloseConnector>
        <client>ianywhere.connector.jboss</client>
    </CloseConnector>
    <SetProperty>
        <prop>
       <client>ianywhere.connector.jboss</client>
  <name>ianywhere.connector.logLevel</name>
  <value>4</value>
   </prop>
    </SetProperty>
    <OpenConnector>
        <client>ianywhere.connector.jboss</client>
    </OpenConnector>
</actions>
```

删除连接器

要删除连接器，请使用 <SetProperty> 删除客户端的所有属性。

示例

在以下示例中，关闭 ID 为 ianywhere.connector.jboss 的连接器。其所有属性均被 <SetProperty> 标记删除，省略名称和值标记。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
        <prop>
       <client>ianywhere.connector.jboss</client>
```



```
        <name>ianywhere.connector.nativeConnection</name>
      </prop>
    </SetProperty>
</actions>
```

打开连接器

要打开连接器，请使用 <OpenConnector>。

OpenConnector 标记包含指定要打开的连接器名称的单个客户端标记。

<OpenConnector> 子标记	说明
<client>	要打开的连接器的名称。

另请参见

- “连接器”第 153 页
- “Web 服务连接器”一节第 163 页

示例

以下示例打开 simpleGroup 连接器。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <OpenConnector>
     <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

关闭连接器

要关闭连接器，请使用 <CloseConnector>。CloseConnector 标记包含指定要关闭的连接器名称的单个客户端标记。

<CloseConnector> 子标记	说明
<client>	要关闭的连接器的名称。

另请参见

- “连接器”第 153 页
- “Web 服务连接器”一节第 163 页

示例

以下示例关闭 simpleGroup 连接器。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
```

```
    <CloseConnector>
     <client>simpleGroup</client>
    </CloseConnector>
</actions>
```

监控连接器

要获取关于连接器的信息，请编写一种称为客户端状态请求的特殊服务器管理请求。它包含一个 <ClientStatusRequest> 标记，该标记使用一个或多个 <request> 标记，而 <request> 标记包含注册请求必需的信息。

您的客户端状态请求可以通过以下几种方式获得有关连接器的报告：

- 进行一次性请求。
- 注册状态更改监听器以在连接器的状态发生变化时发送报告。
- 注册错误监听器以在连接器出错时发送报告。

此外，可以将报告安排为在特定时间或以特定时间间隔发送。

ClientStatusRequest 标记

要获得有关连接器的信息，请使用 <ClientStatusRequest>。

客户端状态请求由一个或多个 <request> 标记组成，<request> 标记包含注册该请求的所有必需信息。

<ClientStatusRequest> 子标记	说明
<request>	请求中的分组信息。

客户端状态请求的 request 标记

在 <request> 标记中，使用可选的 <replyAddr> 标记为作为该请求的结果而生成的每个报告指定返回地址。如果省略此标记，则报告的缺省返回地址为源消息的回复地址。

使用可选的 <requestId> 为包括在每个报告中的请求添加一个标签。注册多个请求时，或者删除或修改请求时，使用此 ID 可以区分哪些报告是从特定请求生成的。

要为请求指定连接器列表，请包括一个或多个 <client> 标记，每个标记带有一个连接器地址。如果是一次性请求，所有连接器都包括在报告中。如果是事件监听器请求，则服务器会对这些连接器中的每个连接器进行监听。

要指定事件详细信息应在任何服务器停机期间都持久保留，请指定 <persistent> 标记。此标记不需要任何数据，其形式可以为 <persistent/> 或 <persistent></persistent>。

通过包括一个或多个 <onEvent> 标记（每个标记具有一个事件类型），可以选择指定事件列表。如果省略这些标记，则客户端状态请求会生成一次性请求。否则，客户端状态请求为指定事件注册事件监听器。

客户端状态请求的 <request> 子标记	说明
<client>	可以包括一个或多个 <client> 标记，每个标记具有一个连接器地址。如果是一次性请求，所有列出的连接器都包括在报告中。如果是事件监听器请求，则服务器开始监听这些连接器中的各个连接器。
<onEvent>	指定发生时服务器应生成报告的事件。可以包括一个或多个 <onEvent> 标记，每个标记具有一个事件类型。如果省略这些标记，则客户端状态请求会生成一次性请求。否则，使用客户端状态请求为指定的事件注册事件监听器。
<persistent>	指定此客户端状态请求中的详细信息应在服务器数据库中持久保留。
<replyAddr>	为作为该请求的结果而生成的每个报告指定返回地址。如果省略此标记，则报告的缺省返回地址为源消息的回复地址。
<requestId>	报告的标签。该值用作请求的标签，并包括在作为该请求的结果而生成的每个报告中。这样做可以在已注册多个请求并要删除或修改未完成的请求时，区分哪些报告是从特定请求生成的。
<schedule>	请参见“服务器管理请求父标记”一节第 694 页。

一次性客户端状态请求

通过省略客户端状态请求的 <onEvent> 和 <schedule> 标记可以创建一次性请求。在这种情况下，生成单个报告，其中包含在客户端状态请求中指定的每个连接器的当前状态信息。

以下 XML 消息省略了 <onEvent> 和 <schedule> 标记，一次性请求示例也同样省略了这些标记。它生成单个报告，其中包含在 <ClientStatusRequest> 标记中指定的每个连接器的当前状态信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <requestId>myOneTimeRequest</requestId>
   <client>ianywhere.server</client>
   <client>ianywhere.connector.beajms</client>
  </request>
 </ClientStatusRequest>
</actions>
```

事件发生时的客户端状态请求

要指定希望 QAnywhere 服务器为其生成状态报告的事件，请在您的客户端状态请求中包括一个或多个 <onEvent> 标记。与一次性请求不同，服务器不会立即对请求做出响应，而是开始监听要发生的事件。每次触发这些事件中的一个事件时，都发送一个报告，其中包含有关引发该事件的连接器的信息。

事件发生时的请求支持以下事件：

事件	发生时间
open	已关闭的连接器打开。
close	先前打开或暂停的连接器关闭。
statusChange	连接器的状态从一种状态变为另一种状态。可能的状态为打开和关闭。
error	连接器抛出意外错误。
fatalError	连接器抛出未处理的致命错误。
none	这永远不会发生。这会从连接器有效地删除所有以前的事件监视项目。

在以下示例中，每次服务器连接器更改其状态或生成错误时，都向地址为 ianywhere.connector.beajms\q11 的连接器发送一个状态报告。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <requestId>myEventRequest</requestId>
   <client>ianywhere.server</client>
   <onEvent>statusChange</onEvent>
   <onEvent>error</onEvent>
  </request>
 </ClientStatusRequest>
</actions>
```

多个同时请求

对于任意数量的连接器（包括服务器连接器），每个返回地址都可以有其自己的一组事件监听器。将事件监听器添加到连接器不会干扰服务器中的任何其它事件监听器（除了可能要替换的监听器）。

请求替换

如果将事件监听器添加到连接器，而该连接器已有使用同一返回地址注册的事件监听器，则会用新监听器替换旧监听器。例如，如果连接器 abc 的 statusChange 监听器注册到地址 x/y，而您又将 abc 的 error 监听器注册到地址 x/y，则 abc 将不再对 statusChange 事件做出响应。

要将多个事件注册到同一地址，必须使用多个 <onEvent> 标记创建单个请求。

删除请求

如果连接器的一个事件监听器注册到某个地址，则您可以通过提供来自同一地址并且指定了 "none" 事件的另一个客户端状态请求来删除该事件监听器。

在以下示例中，注册到地址 ianywhere.connector.beajms\q11 的服务器连接器的所有事件监听器都被删除：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
```



```
    <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
    <client>ianywhere.server</client>
    <onEvent>none</onEvent>
   </request>
  </ClientStatusRequest>
 </actions>
```

持久客户端状态请求

要指定将请求的详细信息都保存到消息存储库的全局属性表中（服务器重新启动后它们可以在此处自动恢复），请在客户端状态请求中包括 <persistent> 标记。持久请求可以与调度事件和事件监听器一起使用，但一次性请求则不能与这两者一起使用。除了不能单独添加调度事件和事件监听器外，添加和删除持久请求的规则与常规请求的规则类似。而添加持久请求时，客户端必须在同一请求中为特定连接器/回复地址对指定所有事件监听器和调度。

以下示例将事件监听器和调度添加到 ianywhere.connector.myConnector 中，并使其持久保留。它还覆盖来自此连接器/回复地址对的任何以前的持久请求。每半小时会发送一次报告，连接器状态发生变化时也会发送报告。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <client>ianywhere.connector.myConnector</client>
   <onEvent>statusChange</onEvent>
   <schedule>
    <everyminute>30</everyminute>
   </schedule>
   <persistent/>
  </request>
 </ClientStatusRequest>
</actions>
```

事件监听器持久性

如果连接器关闭，则该连接器已注册到其地址的任何事件监听器都将在服务器中持续保留，直到服务器关闭。如果重新打开连接器，则存储的事件监听器会再次处于活动状态。

连接器状态

连接器可以处于以下两种状态中的一种：

- **正在运行** 连接器正在接受和处理进来的和外发的消息。此状态以连接器属性 ianywhere.connector.state=1 反映。
- **未运行** 连接器没有接受或处理进来的或外发的消息。此状态以连接器属性 ianywhere.connector.state=2 反映。连接器状态变为 "正在运行" 时，连接器会从头开始初始化。

有关如何更改连接器状态的信息，请参见“修改连接器”一节第 180 页。

客户端状态报告

每次连接器请求报告或发生注册事件时，服务器都会生成客户端状态报告。它以简单文本消息形式生成，不包含任何消息属性。

根据事件发生时哪些信息可用，可以在每个组件报告中包括以下任何值：

- client（始终存在）
- UTCDatetime（始终存在）
- vendorStatusDescription（始终存在）
- statusCode（始终存在）
- vendorStatusCode
- statusSubCode
- statusDescription

例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<ClientStatusReport>
 <requestId>myRequest</requestId>
 <componentReport>
  <client>ianywhere.server</client>
  <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
  <statusCode>Running</statusCode>
  <vendorStatusDescription></vendorStatusDescription>
 </componentReport>
 <componentReport>
  <client>ianywhere.connector.beajms</client>
  <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
  <statusCode>Not running</statusCode>
  <vendorStatusDescription></vendorStatusDescription>
 </componentReport>
</ClientStatusReport>
```


使用服务器管理请求设置服务器属性

<SetProperty> 标记包含一个或多个 <prop> 标记，每个 <prop> 标记指定一个要设置的属性。每个 prop 标记由一个 <client> 标记、一个 <name> 标记和一个 <value> 标记组成。要删除一个属性，请省略 <value> 标记。

<prop> 子标记	说明
<client>	要设置服务器属性的客户端的名称。
<name>	要设置的属性的名称。
<value>	要设置的属性的值。如果不包括该标记，属性会被删除。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

示例

以下服务器管理请求将名为 simpleGroup 的目标别名的 ianywhere.qa.member.client3 属性设置为 Y，这样便将 client3 添加到 simpleGroup 中。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
     <prop>
        <client>simpleGroup</client>
        <name>ianywhere.qa.member.client3</name>
        <value>Y</value>
     </prop>
    </SetProperty>
</actions>
```

下一示例执行以下操作：

- 将 client1 属性 myProp1 的值创建或修改为 3。
- 删除 client1 属性 myProp2。
- 将 client2 属性 myProp3 的值修改为 "some value"。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>client1</client>
     <name>myProp1</name>
     <value>3</value>
 </prop>
 <prop>
     <client>client1</client>
     <name>myProp2</name>
 </prop>
 <prop>
     <client>client2</client>
     <name>myProp3</name>
     <value>some value</value>
```

```
 </prop>
    </SetProperty>
</actions>
```


使用服务器管理请求指定传输规则

使用服务器管理请求，可以指定适用于所有用户的缺省服务器传输规则，也可以为每个客户端指定传输规则。

要（为服务器）指定缺省传输规则，请为客户端 ianywhere.server.defaultClient 设置 ianywhere.qa.server.rules 属性。对于客户端，可使用 ianywhere.qa.server.rules 属性来指定服务器传输规则。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

示例

以下示例创建缺省规则，该规则指定仅发送高优先级消息（优先级大于 6）：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>ianywhere.server.defaultClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_Priority &gt; 6</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>ianywhere.server.defaultClient</client>
  </RestartRules>
</actions>
```

以下示例为客户端 myClient 创建一个规则，该规则指定在业务时间（上午 8 时至下午 6 时）内仅传输内容大小小于 100 的消息：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>myClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_ContentSize &lt; 100
        or ias_CurrentTime &gt; '8:00:00'
        or ias_CurrentTime &lt; '18:00:00'</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>myClient</client>
  </RestartRules>
</actions>
```


使用服务器管理请求创建目标别名

可以使用服务器管理请求创建和修改目标别名。

有关目标别名的详细信息，请参见“目标别名”一节第 150 页。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

要创建一个目标别名，可发送一个服务器管理请求，其中的客户端名称为目标别名的名称，并指定以下属性。组由 group、address 和 nativeConnection 属性标识。组的成员使用成员属性指定。

```
<prop>
  <client>simpleGroup</client>
  <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.group.GroupConnector
  </value>
</prop>
```

属性	说明

ianywhere.qa.group	将此属性设置为 Y 以表示正在配置目标别名。例如：
	
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.qa.group</name>`
	`  <value>Y</value>`
	`</prop>`

ianywhere.connector.address	指定目标别名的客户端 ID。例如：
	
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.connector.address</name>`
	`  <value>simpleGroup</value>`
	`</prop>`

ianywhere.connector.nativeConnection	设置为 ianywhere.message.connector.group.GroupConnector。例如：
	
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.connector.nativeConnection</name>`
	
	`<value>ianywhere.message.connector.group.GroupConnector`
	`  </value>`
	`</prop>`

属性	说明

ianywhere.qa.member.*client-name*
queue-name
	指定 Y 添加一个成员，或指定 N 删除一个成员。也可以选择指定传送条件。请参见“条件语法”一节第 763 页。例如，要将 client1 添加到目标别名 simpleGroup，可如下设置属性。queue-name 是可选的。对要添加的每个客户端重复此属性：
	
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.qa.member.client1\queue1</name>`
	`  <value>Y</value>`
	`</prop>`

有关服务器管理请求的详细信息，请参见“服务器管理请求简介”一节第 172 页。

另请参见

- “QAnywhere 传输和删除规则”第 761 页

示例

以下服务器管理请求创建名为 simpleGroup 的目标别名，其中包含名为 client1 和 client2\q11 的成员。此示例启动目标别名使它立即开始处理消息。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.group</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.address</name>
      <value>simpleGroup</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.nativeConnection</name>
      <value>ianywhere.message.connector.group.GroupConnector</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.logLevel</name>
      <value>4</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client1</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client2\q11</name>
      <value>Y</value>
    </prop>
  </SetProperty>
  <OpenConnector>
    <client>simpleGroup</client>
```

```
  </OpenConnector>
</actions>
```

在目标别名中添加和删除成员

要将成员添加到目标别名，可创建一个在某个属性中指定该成员的服务器管理请求。必须重新启动组才能使成员设置生效。

以下示例添加成员 client3 并重新启动组 simpleGroup：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>simpleGroup</client>
     <name>ianywhere.qa.member.client3</name>
     <value>Y</value>
 </prop>
    </SetProperty>
    <CloseConnector>
 <client>simpleGroup</client>
    </CloseConnector>
    <OpenConnector>
 <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

要将成员从目标别名中删除，请创建一个服务器管理请求，其中包含的属性设置指示必须删除该成员。必须重新启动组才能使成员删除设置生效。

以下示例删除成员 client3 并重新启动组 simpleGroup：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>simpleGroup</client>
     <name>ianywhere.qa.member.client3</name>
 </prop>
    </SetProperty>
    <CloseConnector>
 <client>simpleGroup</client>
    </CloseConnector>
    <OpenConnector>
 <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

监控 QAnywhere

可以使用服务器管理请求以取得有关一组消息的信息。服务器编译该信息并通过消息将其发送回客户端。可以创建一次性消息详细信息请求，也可以将消息详细信息请求安排为自动发生。此外，可以指定您的请求应持久保留，这样即使重新启动服务器也会发送消息。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

消息详细信息请求

要为消息详细信息编写服务器管理请求，请使用 <MessageDetailsRequest> 标记。

消息详细信息请求包含一个或多个 <request> 标记，<request> 标记包含注册该请求的所有必需信息。指定多个 <request> 标记等效于分别发送多个消息详细信息请求。

使用可选的 <replyAddr> 标记为作为该请求的结果而生成的每个报告指定返回地址。如果省略此标记，则报告的缺省返回地址为源消息的回复地址。

使用 <requestId> 标记为包括在作为该请求的结果而生成的每个报告中的请求指定唯一标识符。对此字段使用不同值可以使多个请求同时处于活动状态。使用同一请求 ID 可以使客户端覆盖或删除活动请求。

指定 <condition> 标记以确定哪些消息应包括在报告中。请参见“Condition 标记”一节第 694 页。

也可以指定一个详细信息列表以确定每条消息的哪些详细信息应包括在报告中。为此，可在请求中包括一组空详细信息元素标记。

可以使用 <persistent> 标记指定事件详细信息在任何服务器停机期间都应持久保留。此标记不需要任何数据，其形式可以为 <persistent/> 或 <persistent></persistent>。

可以使用 <schedule> 包括注册已调度报告需要的所有详细信息。请参见“服务器管理请求父标记”一节第 694 页。

<MessageDetailsRequest> 子标记	说明
<request>	有关特定请求的分组信息。指定多个 <request> 标记等效于为消息信息分别发送多个服务器管理请求。请参见以下内容。

Request 标记

<Request> 子标记	说明
<address>	请求每条消息的地址。
<archived>	请求消息是否在档案存储库中。
<condition>	用于在报告中包括消息的分组条件。请参见“Condition 标记”一节第 694 页。

<Request> 子标记	说明
<contentSize>	请求每条消息的内容大小。
<expires>	请求每条消息的到期时间。
<kind>	请求消息是文本格式还是二进制格式。
<messageId>	请求每条消息的消息 ID。
<originator>	请求每条消息的发出方。
<persistent>	包括此标记表示请求的结果应在服务器数据库中持久保留（这样即使重新启动服务器也发送报告）。
<priority>	请求每条消息的优先级。
<property>	请求每条消息的所有消息属性和值的列表。
<statusTime>	请求每条消息的状态时间。
<replyAddr>	为作为该请求的结果而生成的每个报告指定返回地址。如果省略此标记，则报告的缺省返回地址为源消息的回复地址。
<requestId>	此值是请求的唯一标识符，并包括在作为该请求的结果而生成的每个报告中。对此字段使用不同值可以使多个请求同时处于活动状态。使用同一请求 ID 可以使客户端覆盖或删除活动请求。
<schedule>	若包含此标记，则表示应按调度生成报告。<schedule> 的子标记标识报告据以运行的调度。请参见“服务器管理请求父标记”一节第 694 页。
<status>	请求每条消息的状态。
<transmissionStatus>	请求每条消息的传输状态。

监控 QAnywhere 客户端

可以使用服务器管理请求获取当前服务器上的客户端列表。此列表包含在服务器上注册的客户端，包括远程客户端、打开的连接器和目标别名。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

要获取客户端列表，请在您的服务器管理请求中使用 <GetClientList> 标记。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <GetClientList/>   (or <GetClientList></GetClientList> )
</actions>
```


生成的响应被发送到包含该请求的消息的回复地址。该响应包含一列 <client> 标记，每个 <client> 标记命名一个连接到服务器的客户端。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<GetClientListResponse>
 <client>ianywhere.server</client>
 <client>ianywhere.connector.myConnector</client>
 <client>myClient</client>
</GetClientListResponse>
```

监控属性

可以使用服务器管理请求查看为客户端设置了哪些属性。响应仅列出已为客户端设置的属性（非缺省属性）。

有关如何使用服务器管理请求（包括如何对它们进行验证和调度）的概述，请参见“服务器管理请求简介”一节第 172 页。

要获取客户端属性列表，请在您的服务器管理请求中使用 <GetProperties> 标记。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <GetProperties>
  <client>ianywhere.connector.myConnector</client>
 </GetProperties>
</actions>
```

生成的响应被发送到包含该请求的消息的回复地址。该响应包含客户端名称和一列 <prop> 标记，每个 <prop> 标记包含一个属性的详细信息。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<GetPropertiesResponse>
 <client>ianywhere.connector.myConnector</client>
 <prop>
  <name>ianywhere.connector.logLevel</name>
  <value>4</value>
 </prop>
 <prop>
  <name>ianywhere.connector.state</name>
  <value>2</value>
 </prop>
</GetPropertiesResponse>
```


教程：探讨 TestMessage

目录

关于本教程

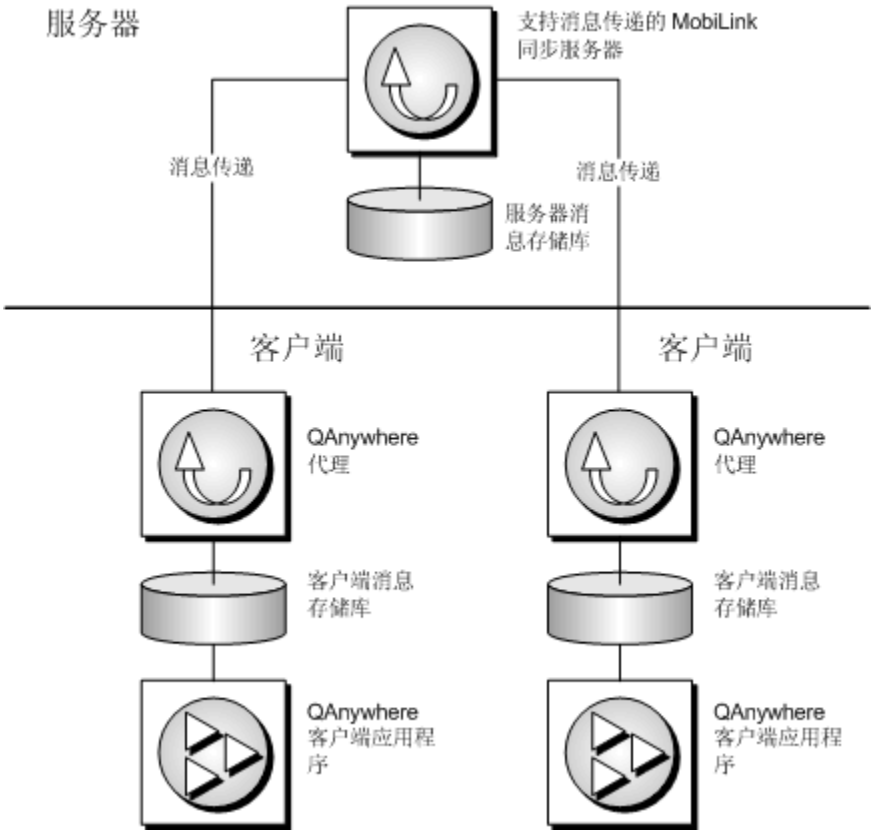
TestMessage 是一个 QAnywhere 示例客户端应用程序。此应用程序演示如何使用 QAnywhere 来创建您自己的消息传递客户端应用程序。TestMessage 提供一个客户端到客户端界面来发送、接收和显示消息。文本消息明白易懂，因此它能很好地演示 QAnywhere 消息传递；然而，QAnywhere 所提供的功能远不只于文本消息传递。它是一般性的应用程序到应用程序的消息传递系统，能在多个客户端之间提供基于消息的通信。

本教程为 Windows 桌面操作系统而编写。使用这些平台是为了便于演示，您也可以使用 QAnywhere API 来编写运行于 Windows Mobile 设备上的应用程序。提供用于 Windows Mobile 以 C++、Visual Basic、C# 和 Java 语言编写的源代码。还有针对 .NET Compact Framework 编写的 C# 版本。

第 1 课：启动支持消息传递的 MobiLink

背景

QAnywhere 使用 MobiLink 同步来发送和接收信息。从一个客户端到另一个客户端的所有消息都通过 MobiLink 中央服务器来传送。典型系统的体系结构（只有两个 QAnywhere 客户端）如下图所示。



服务器消息存储库是一个配置用作 MobiLink 统一数据库的数据库。TestMessage 示例使用一个 SQL Anywhere 统一数据库作为其服务器消息存储库。

在服务器消息存储库中唯一需要的表是 MobiLink 系统表，这些系统表包括在任何设置为 MobiLink 统一数据库的支持数据库中。

系统表由 MobiLink 维护。关系数据库提供了高性能的安全消息存储库。使您可以轻松地将消息传递功能集成到现有的数据管理和同步系统中。

QAnywhere 消息传递通常在不同的计算机上运行，但在此教程中，所有组件都运行在同一计算机上。跟踪哪些活动是客户端活动、哪些活动是服务器活动非常重要。

在本课中，您将在服务器上执行操作。

活动

通过提供 -m 选项并指定连接到服务器消息存储库的连接字符串，MobiLink 服务器可以在启动时启用消息传递功能。TestMessage 示例使用 QAnywhere 示例数据库作为服务器消息存储库。在 TestMessage 示例中，您可以使用命令行选项、使用 SQL Anywhere 安装中的示例快捷方式或通过 Sybase Central 的 QAnywhere 插件来启动启用消息传递功能的 MobiLink 服务器。

♦ 启动消息传递服务器

1. 从 [开始] 菜单中选择 [程序] » [SQL Anywhere 11] » [QAnywhere] » [MobiLink QAnywhere 示例]。

   或者，在命令提示符下，运行以下命令：

   ```
   mlsrv11 -m -c "dsn=QAnywhere 11 Demo" -vcrs -zu+
   ```

   此命令行使用以下 mlsrv11 选项：

选项	说明
-m	-m 选项启用消息传递。请参见“-m 选项”一节 《MobiLink - 服务器管理》。
-c	-c 选项指定连接到服务器消息存储库的连接字符串，在本例中使用 QAnywhere 11.0 Demo ODBC 数据源。请参见“-c 选项”一节 《MobiLink - 服务器管理》。
-vcrs	-vcrs 选项提供服务器活动的详细记录，在开发期间会用到这些记录。请参见“-v 选项”一节 《MobiLink - 服务器管理》。
-zu+	-zu+ 选项将用户名自动添加到系统中；使用此选项将便于您学习教程或进行开发，但通常不会在生产环境中使用它。请参见“-zu 选项”一节 《MobiLink - 服务器管理》。

2. 将 [MobiLink 服务器] 窗口移到您屏幕的中央，它表示本教程中的服务器。

另请参见

- “启动启用了 MobiLink 的 QAnywhere”一节第 30 页
- “-m 选项”一节 《MobiLink - 服务器管理》
- “QAnywhere 快速入门”一节第 12 页
- “简单消息传递方案”一节第 4 页

第 2 课：运行 TestMessage 应用程序

背景

TestMessage 是一个使用 QAnywhere 来发送和接收文本消息的简单应用程序。本教程之所以使用文本消息传递，是因为它能提供简单且可访问的消息传递的演示。然而，QAnywhere 并不仅仅是一个文本消息传递系统，它提供通用的应用程序到应用程序的消息传递功能。

在本课中，您将在客户端上执行活动。通常，客户端与服务器在不同的计算机上运行。

在本课中，您将启动客户端消息存储库，这是 TestMessage 示例的一部分。在第 3 课中，您使用该消息存储库将消息发送到另一客户端消息存储库。

活动

♦ 启动具有 TestMessage 客户端消息存储库的 QAnywhere 代理

1. 从 [**开始**] 菜单，选择 [**程序**] » [**SQL Anywhere 11**] » [**QAnywhere**] » [**使用 SQL Anywhere 的教程**] » [**QAnywhere Agent For SQLAnywhere - saclient1**]。

   [**QAnywhere 代理**] 将连接到第一个 TestMessage 示例客户端消息存储库，并管理传输自/至该消息存储库的消息。

2. 将第一个 [**QAnywhere 代理**] 窗口移到您屏幕的右侧。

   > 在继续进行下一步之前必须为 QAnywhere 代理的第一个实例留出几秒时间启动。

3. 从 [**开始**] 菜单，选择 [**程序**] » [**SQL Anywhere 11**] » [**QAnywhere**] » [**使用 SQL Anywhere 的教程**] » [**QAnywhere Agent For SQLAnywhere - saclient2**]。

   第二个 [**QAnywhere 代理**] 启动并连接到第二个 TestMessage 示例客户端消息存储库，管理传输自/至该消息存储库的消息。

4. 将第二个 [**QAnywhere 代理**] 窗口移到您屏幕的左侧。

♦ 启动 TestMessage

1. 从 [**开始**] 菜单，选择 [**程序**] » [**SQL Anywhere 11**] » [**QAnywhere**] » [**使用 SQL Anywhere 的教程**] » [**TestMessage -- saclient1**]。
2. 将 [**saclient1 - TestMessage**] 窗口移到您屏幕的右侧。
3. 在 [**saclient1 - TestMessage**] 窗口中，单击 [**工具**] » [**选项**]。
4. 检验 [**Queue Name Used To Listen For Incoming Messages**] 字段中是否显示 **testmessage**。单击 [**取消**]。
5. 从 [**开始**] 菜单，选择 [**程序**] » [**SQL Anywhere 11**] » [**QAnywhere**] » [**使用 SQL Anywhere 的教程**] » [**TestMessage -- saclient2**]。
6. 将 [**saclient2 - TestMessage**] 窗口移到您屏幕的左侧。
7. 在 [**saclient2 - TestMessage**] 窗口中，单击 [**工具**] » [**选项**]。

8. 检验 [**Queue Name Used To Listen For Incoming Messages**] 字段中是否显示 **testmessage**。单击 [**Cancel**]。

讨论

您可以通过设置消息传输策略来配置 QAnywhere 代理监控消息的方式。此示例的设计初衷是仅使用 [自动] 或 [调度] 策略，并且将使用 [自动] 策略启动 QAnywhere 代理。这些 QAnywhere 策略为：

- **调度** 该策略设置指示 QAnywhere 代理定期传输消息。如果没有指定时间间隔，则缺省为 15 分钟。
- **自动** 该缺省设置会使 QAnywhere 代理在从客户端消息存储库收发的消息做好传送准备时随时传输消息。
- **要求时** 该策略设置将使 QAnywhere 代理只在接到应用程序指示时才传输消息。
- **自定义** 在此模式下，您可以提供一组规则以指定更复杂的传输行为。

QAnywhere 消息被传递到由一个客户端消息存储库 ID 和一个队列名组成的 QAnywhere 地址中。缺省 ID 为运行 QAnywhere 代理的计算机的名称。每个消息存储库都需要其自己的 QAnywhere 代理。每个应用程序都可以监听多个队列，但每个队列应该只特定于一个应用程序。

另请参见

- “启动 QAnywhere 代理”一节第 48 页
- “确定在客户端进行消息传输的时间”一节第 51 页
- “qaagent 实用程序”一节第 704 页
- “QAnywhere 传输和删除规则”第 761 页
- “编写 QAnywhere 客户端应用程序”第 53 页
- 安装到 *samples-dir\QAnywhere* 的 QAnywhere 示例。（有关 *samples-dir* 的详细信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）

第 3 课：发送消息

背景

在本课中，您要将消息从 TestMessage saclient1 应用程序发送到 TestMessage saclient2 应用程序。

活动

♦ **从 TestMessage 发送消息**

1. 在 [**saclient1 - TestMessage**] 窗口中，单击 [**消息**] » [**新建**]。
2. 在 [**Destination ID**] 字段中，键入 **saclient2**。
3. 在 [**主题**] 字段中，键入当前时间。使用当前时间可便于跟踪各条消息。
4. 在 [**消息**] 字段中，键入 **sample**。
5. 单击 [**发送**]。
6. 单击 [**确定**]。
7. 在 [**saclient2 - TestMessage**] 窗口中，选择该消息。底部窗格中将出现该消息的内容。

讨论

与其它 QAnywhere 应用程序类似，TestMessage 也使用 QAnywhere API 来管理消息。QAnywhere API 以 C++ API、Java API、Microsoft .NET API 和 SQL API 的形式提供。

另请参见

- “QAnywhere 消息地址”一节第 63 页
- “发送 QAnywhere 消息”一节第 66 页
- “消息删除规则”一节第 772 页

第 4 课：探讨 TestMessage 客户端源代码

背景

本节教程引导您简要地浏览 TestMessage 客户端应用程序的源代码。

有很多代码用于实现 Windows 接口，通过接口您可以发送、接收和查看消息。不过，教程的这一部分侧重于提供给 QAnywhere 的部分代码。

可以在 *Samples\QAnywhere* 中找到 TestMessage 源代码。

提供了多个版本的 TestMessage 源代码。以下版本用于 Windows 2000 和 Windows XP：

- 提供使用 Microsoft 基础类建立的 C++ 版本作为 *Samples\QAnywhere\Windows\MFC\TestMessage\TestMessage.sln*。
- 提供使用 .NET Framework 建立的 C# 版本作为 *Samples\QAnywhere\Windows\.NET\CS\TestMessage\TestMessage.sln*。
- 提供 Java 版本作为 *Samples\QAnywhere\Java\TestMessage\TestMessage.java*。

为 .NET Compact Framework 提供了以下版本：

- 提供在 .NET Compact Framework 上建立的 C# 版本作为 *Samples\QAnywhere\Windows Mobile Classic\.NET\CS\TestMessage\TestMessage.sln*。

必需的软件

需要 Visual Studio 2005 或更高版本才能打开解决方案文件并构建 .NET Framework 项目和 .NET Compact Framework 项目。

探讨 C# 源代码

本节将引导您浏览 C# 源代码。这两个版本的结构非常相似。

本课不是让您查看应用程序中的每一行，而是重点介绍对理解 QAnywhere 应用程序很有帮助的某些行。将使用 C# 版本来演示这些行。

1. 打开您感兴趣的 TestMessage 项目版本。

   双击解决方案文件以在 Visual Studio 中打开项目。例如，*Samples\QAnywhere\Windows\.NET\CS\TestMessage\TestMessage.sln* 是解决方案文件。有多个解决方案文件用在不同的环境中。

2. 确保打开了 [Solution Explorer]。

   可以从 [**View**] 菜单中打开 [Solution Explorer]。

3. 检查 [**Source Files**] 文件夹。

   有两个文件特别重要。*MessageList* 文件 (*MessageList.cs*) 接收消息并允许您查看消息。*NewMessage* 文件 (*NewMessage.cs*) 允许您构造和发送消息。

4. 从 [Solution Explorer] 中打开 *MessageList* 文件。

5. 检查所包含的命名空间。

每个 QAnywhere 应用程序都需要 iAnywhere.QAnywhere.Client 命名空间。定义此命名空间的程序集作为 *iAnywhere.QAnywhere.Client.dll* DLL 提供。这些文件位于以下位置：

- .NET Framework 2.0：*install-dir\Assembly\v2*
- .NET Compact Framework 2.0：*install-dir\ce\Assembly\v2*

对于您自己的项目，编译时必须包括对此 DLL 的引用。在每个文件的顶部使用以下行包含该命名空间：

```
using iAnywhere.QAnywhere.Client;
```

6. 检查 startReceiver 方法。

   此方法执行 QAnywhere 应用程序共有的初始化任务：

   - 创建 QAManager 对象。

     ```
     _qaManager =
     QAManagerFactory.Instance.CreateQAManager();
     ```

     QAnywhere 提供了一个 QAManagerFactory 对象来创建 QAManager 对象。QAManager 对象处理 QAnywhere 消息传递操作：特别是接收消息（从队列中获取消息）和发送消息（将消息置于队列中）。

     QAnywhere 提供了两种类型的管理器：QAManager 和 QATransactionalManager。当使用 QATransactionalManager 时，所有发送和接收操作都在一个事务内发生，以便要么发送（或接收）所有消息，要么都不发送（或都不接收）。

   - 编写一个方法来处理消息。

     onMessage() 方法由 QAnywhere 调用以处理常规非系统消息。它接收的消息被编码为 QAMessage 对象。QAMessage 类及其子类（QATextMessage 和 QABinaryMessage）提供保存 QAnywhere 应用程序所需的与消息相关的所有信息的属性和方法。

     ```
     private void onMessage( QAMessage msg ) {
        Invoke( new onMessageDelegate( onMessageReceived ),
           new Object [] { msg } );
     }
     ```

     该代码使用 Form 的 Invoke 方法，让运行基础窗口的线程来处理事件，以便用户界面可更新以显示消息。该线程也是创建 QAManager 的线程。在一些例外情况中，QAManager 只能从创建它的线程进行访问。

   - 声明以 MessageList_Load 方法定义的 MessageListener。

     ```
     _receiveListener = new
         QAManager.MessageListener( onMessage );
     ```

     每当 QAnywhere 代理接收到一条消息并将其置于应用程序监听的队列中时，就会调用 OnMessage() 方法。

消息监听器和通知监听器

消息监听器与在“使用推式通知进行消息传递的方案”一节第 6 页中描述的监听器组件不同。监听器组件接收通知，而消息监听器对象从队列中检索消息。

当您为队列设置一个消息监听器时，QAnywhere Manager 将把到达该队列的消息传递给该监听器。只能为一个给定的队列设置一个监听器。设置一个空监听器将清除该队列的所有监听器。

MessageListener 能够异步接收消息。您也可以同步接收信息，也就是说，应用程序显式地去队列中查找消息（可能是响应一个用户操作，如单击 [刷新] 按钮），而不是在消息出现时获得通知。

其它初始化任务包括：

- 打开并启动 QAManager 对象。

  ```
  _qaManager.Open(
          AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
  _qaManager.Start();
  ```

  AcknowledgementMode 枚举常量确定如何向发送者确认消息已接收。EXPLICIT_ACKNOWLEDGEMENT 常量指示直到调用一个 QAManager 确认方法时才确认消息。

- 装载在队列中等待的任何消息。

  ```
  loadMessages();
  ```

- 将一个监听器指派给将来的消息队列。

  该监听器在 MessageList_Load() 方法中声明。

  ```
  _qaManager.SetMessageListener(
    _options.ReceiveQueueName,
    _receiveListener );
  ```

  选项 ReceiveQueueName 属性包含字符串 **testmessage**，它是在 [TestMessage 选项] 窗口中设置的 TestMessage 队列。

7. 在同一文件中检查 addMessage() 方法。

   每当应用程序接收到消息时，就会调用此方法。它将获取消息的属性，例如消息的回复地址、首选名称和发送时间（时间戳），并在 TestMessage 用户界面中显示此信息。以下行将进来的消息转换为 QATextMessage 对象并获取消息的回复地址：

   ```
   text_msg = ( QATextMessage )msg;
   from = text_msg.ReplyToAddress;
   ```

   这里简要分析了 *MessageList* 文件中的一些主要任务。

8. 从 [Solution Explorer] 中打开 *NewMessage* 文件。

9. 检查 sendMessage() 方法。

   此方法获取在 [**New Message**] 窗口中输入的信息并构造一个 QATextMessage 对象。然后，QAManager 将该消息置于要发送的队列中。

   下面的行用于创建一个 QATextMessage 对象并设置它的 ReplyToAddress 属性：

   ```
   qa_manager = MessageList.GetQAManager();
   msg = qa_manager.CreateTextMessage();
   msg.ReplyToAddress = MessageList.getOptions().ReceiveQueueName;
   ```

   下面的行用于将消息置于要发送的队列中。变量 `dest` 为目标地址，将作为参数提供给函数。


```
qa_manager.PutMessage( dest, msg );
```

另请参见

- “QAnywhere C++ API 参考”第 385 页
- “用于客户端的 QAnywhere .NET (.NET 2.0)”一节第 212 页
- “编写 QAnywhere 客户端应用程序”第 53 页
- 安装到 *samples-dir**QAnywhere* 的 TestMessage 示例。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）

教程清理

关闭 TestMessage、QAnywhere 代理和 MobiLink 服务器的所有实例。

QAnywhere 参考

本部分提供 QAnywhere API 的参考文档。

QAnywhere .NET API 参考

目录

用于客户端的 QAnywhere .NET (.NET 2.0)

命名空间

- iAnywhere.QAnywhere.Client（用于常规客户端）
- iAnywhere.QAnywhere.StandAloneClient（用于独立客户端）

AcknowledgementMode 枚举

指出 QAnywhere 客户端应用程序应如何确认消息。

语法

Visual Basic
Public Enum **AcknowledgementMode**

C#
public enum **AcknowledgementMode**

注释

隐式和显式确认模式被指派给使用 QAManager.Open(AcknowledgementMode) 方法的 QAManager 实例。

有关详细信息，请参见“初始化 QAnywhere API”一节第 57 页。

使用隐式确认时，客户端应用程序在收到消息时会进行确认。使用显式确认时，必须调用其中一个 QAManager 确认方法。服务器在客户端之间传播全部状态变化。

有关详细信息，请参见“同步接收消息”一节第 74 页和“异步接收消息”一节第 75 页。

成员名称

成员名称	说明
EXPLICIT_ACKNOWLEDGEMENT	表示使用其中一个 QAManager 确认方法确认已接收到的消息。
IMPLICIT_ACKNOWLEDGEMENT	表示客户端应用程序在接收到每条消息时都会进行确认。如果同步接收消息，则返回 QAManagerBase.GetMessage(string) 方法时将确认消息。如果异步接收消息，则事件处理函数返回时将确认消息。
TRANSACTIONAL	此模式表示消息仅作为进行中事务的一部分进行确认。此模式自动指派给 QATransactionalManager 实例。

另请参见

- “QAManager 接口”一节第 253 页
- “QATransactionalManager 接口”一节第 328 页
- “QAManagerBase 接口”一节第 257 页

ExceptionListener 委派

ExceptionListener 委派定义。将 ExceptionListener 传递给 SetExceptionListener 方法。

语法

Visual Basic

```
Public Delegate Sub ExceptionListener( _
  ByVal ex As QAException, _
  ByVal msg As QAMessage _
)
```

C#

```
public delegate void ExceptionListener(
  QAException ex,
  QAMessage msg
);
```

参数

- **ex** 发生的异常。
- **msg** 返回接收到的消息；如果无法构造消息，则返回空值。

另请参见

- “SetExceptionListener 方法”一节第 286 页

ExceptionListener2 委派

ExceptionListener2 委派定义。将 ExceptionListener2 传递给 SetExceptionListener2 方法。

语法

Visual Basic

```
Public Delegate Sub ExceptionListener2( _
  ByVal mgr As QAManagerBase, _
  ByVal ex As QAException, _
  ByVal msg As QAMessage _
)
```

C#

```
public delegate void ExceptionListener2(
  QAManagerBase mgr,
  QAException ex,
  QAMessage msg
);
```


参数

- **mgr** 处理消息的 QAManagerBase。
- **ex** 发生的异常。
- **msg** 返回接收到的消息；如果无法构造消息，则返回空值。

另请参见

- “SetExceptionListener2 方法”一节第 286 页

MessageListener 委派

MessageListener 委派定义。您将 MessageListener 传递给 SetMessageListener 方法。

语法

Visual Basic

```
Public Delegate Sub MessageListener( _
  ByVal msg As QAMessage _
)
```

C#

```
public delegate void MessageListener(
  QAMessage msg
);
```

参数

- **msg** 接收到的消息。

另请参见

- “SetMessageListener 方法”一节第 289 页

MessageListener2 委派

MessageListener2 委派定义。将 MessageListener2 传递给 SetMessageListener2 方法。

语法

Visual Basic

```
Public Delegate Sub MessageListener2( _
  ByVal mgr As QAManagerBase, _
  ByVal msg As QAMessage _
)
```

C#

```
public delegate void MessageListener2(
  QAManagerBase mgr,
  QAMessage msg
);
```


参数

- **mgr** 接收消息的 QAManagerBase。
- **msg** 接收到的消息。

另请参见

- “SetMessageListener2 方法”一节第 290 页

MessageProperties 类

提供存储标准消息属性名称的字段。

语法

Visual Basic
Public Class **MessageProperties**

C#
public class **MessageProperties**

注释

MessageProperties 类提供标准消息属性名称。您可以将 MessageProperties 字段传递给用于获取和设置消息属性的 QAMessage 方法。

有关详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “MessageProperties 成员”一节第 215 页
- “QAMessage 接口”一节第 304 页
- “GetIntProperty 方法”一节第 314 页
- “GetStringProperty 方法”一节第 318 页

MessageProperties 成员

公共静态字段（共享）

成员名称	说明
“ADAPTER 字段”一节第 216 页	用于 system 队列消息，指用来连接 QAnywhere 服务器的网络适配器。
“ADAPTERS 字段”一节第 217 页	此属性名称是指一个可用于连接 QAnywhere 服务器的网络适配器的分隔列表。
“DELIVERY_COUNT 字段”一节第 217 页	此属性名称指目前已尝试传送消息的次数。

成员名称	说明
“IP 字段”一节第 218 页	用于 system 队列消息，指用来连接 QAnywhere 服务器的网络适配器的 IP 地址。
“MAC 字段”一节第 218 页	用于 system 队列消息，指用来连接 QAnywhere 服务器的网络适配器的 MAC 地址。
“MSG_TYPE 字段”一节第 219 页	此属性名称是指与 QAnywhere 消息相关的 MessageType 值。
“NETWORK_STATUS 字段”一节第 219 页	此属性名称是指网络连接的状态。
“ORIGINATOR 字段”一节第 220 页	此属性名称是指消息发出方的消息存储库 ID。
“RAS 字段”一节第 220 页	用于 system 队列消息，是指用来连接 QAnywhere 服务器的 RAS 条目名。
“RASNAMES 字段”一节第 221 页	用于 system 队列消息，是指可用来连接 QAnywhere 服务器的 RAS 条目名的分隔列表。
“STATUS 字段”一节第 221 页	此属性名称是指消息的当前状态。
“STATUS_TIME 字段”一节第 222 页	此属性名称是指消息转为其当前状态的时间。
“TRANSMISSION_STATUS 字段”一节第 222 页	此属性名称是指消息的当前传输状态。

ADAPTER 字段

用于 system 队列消息，指用来连接 QAnywhere 服务器的网络适配器。

语法

Visual Basic
Public Shared **ADAPTER** As String

C#
public const string **ADAPTER**;

注释

此字段的值为 ias_Network.Adapter。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.ADAPTER 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

ADAPTERS 字段

此属性名称是指一个可用于连接 QAnywhere 服务器的网络适配器的分隔列表。

语法

Visual Basic
Public Shared **ADAPTERS** As String

C#
public const string **ADAPTERS**;

注释

它用于系统队列消息。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.ADAPTERS 以访问相关属性。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

DELIVERY_COUNT 字段

此属性名称指目前已尝试传送消息的次数。

语法

Visual Basic
Public Shared **DELIVERY_COUNT** As String

C#
public const string **DELIVERY_COUNT**;

注释

此字段的值为 ias_DeliveryCount。

您可以在 QAMessage.GetIntProperty 方法中传递 MessageProperties.DELIVERY_COUNT 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetIntProperty 方法”一节第 314 页

IP 字段

用于 system 队列消息，指用来连接 QAnywhere 服务器的网络适配器的 IP 地址。

语法

Visual Basic
Public Shared **IP** As String

C#
public const string **IP**;

注释

此字段的值为 ias_Network.IP。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.IP 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

MAC 字段

用于 system 队列消息，指用来连接 QAnywhere 服务器的网络适配器的 MAC 地址。

语法

Visual Basic
Public Shared **MAC** As String

C#
public const string **MAC**;

注释

此字段的值为 ias_Network.MAC。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.MAC 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

MSG_TYPE 字段

此属性名称是指与 QAnywhere 消息相关的 MessageType 值。

语法

Visual Basic
Public Shared **MSG_TYPE** As String

C#
public const string **MSG_TYPE**;

注释

此字段的值为 ias_MessageType。

您可以在 QAMessage.GetIntProperty 方法中传递 MessageProperties.MSG_TYPE 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “MessageType 枚举”一节第 225 页
- “GetIntProperty 方法”一节第 314 页
- “GetStringProperty 方法”一节第 318 页

NETWORK_STATUS 字段

此属性名称是指网络连接的状态。

语法

Visual Basic
Public Shared **NETWORK_STATUS** As String

C#
public const string **NETWORK_STATUS**;

注释

如果网络可访问，则值为 1，否则值为 0。

网络状态用于 system 队列消息（如网络状态更改）。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

您可以在 QAMessage.GetIntProperty 方法中传递 MessageProperties.NETWORK_STATUS 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetIntProperty 方法”一节第 314 页

ORIGINATOR 字段

此属性名称是指消息发出方的消息存储库 ID。

语法

Visual Basic
Public Shared **ORIGINATOR** As String

C#
public const string **ORIGINATOR**;

注释

此字段的值为 ias_Originator。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.ORIGINATOR 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

RAS 字段

用于 system 队列消息，是指用来连接 QAnywhere 服务器的 RAS 条目名。

语法

Visual Basic
Public Shared **RAS** As String

C#
public const string **RAS**;

注释

此字段的值为 ias_Network.RAS。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.RAS 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

RASNAMES 字段

用于 system 队列消息，是指可用来连接 QAnywhere 服务器的 RAS 条目名的分隔列表。

语法

Visual Basic
Public Shared **RASNAMES** As String

C#
public const string **RASNAMES**;

注释

此字段的值为 ias_RASNames。

有关详细信息，请参见“预定义的客户端消息存储库属性”一节第 746 页。

您可以在 QAMessage.GetStringProperty 方法中传递 MessageProperties.RASNAMES 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “MessageProperties 类”一节第 215 页
- “GetStringProperty 方法”一节第 318 页

STATUS 字段

此属性名称是指消息的当前状态。

语法

Visual Basic
Public Shared **STATUS** As String

C#
public const string **STATUS**;

注释

有关属性值的列表，请参见“StatusCodes 枚举”一节第 331 页。

此字段的值为 ias_Status。

您可以在 QAMessage.GetIntProperty 方法中传递 MessageProperties.STATUS 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “StatusCodes 枚举”一节第 331 页
- “MessageProperties 类”一节第 215 页
- “GetIntProperty 方法”一节第 314 页

STATUS_TIME 字段

此属性名称是指消息转为其当前状态的时间。

语法

Visual Basic
Public Shared **STATUS_TIME** As String

C#
public const string **STATUS_TIME**;

注释

它为本地时间。当 STATUS_TIME 传递给 QAMessage.GetProperty 时，它会返回一个 DateTime 对象。此字段的值为 ias_StatusTime。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “GetProperty 方法”一节第 315 页
- “MessageProperties 类”一节第 215 页
- “GetProperty 方法”一节第 315 页

TRANSMISSION_STATUS 字段

此属性名称是指消息的当前传输状态。

语法

Visual Basic
Public Shared **TRANSMISSION_STATUS** As String

C#
public const string **TRANSMISSION_STATUS**;

注释

有关属性值的列表，请参见“StatusCodes 枚举”一节第 331 页。

此字段的值为 ias_TransmissionStatus。

您可以在 QAMessage.GetIntProperty 方法中传递 MessageProperties.TRANSMISSION_STATUS 以访问相关属性。

另请参见

- “MessageProperties 类”一节第 215 页
- “MessageProperties 成员”一节第 215 页
- “StatusCodes 枚举”一节第 331 页
- “MessageProperties 类”一节第 215 页
- “GetIntProperty 方法”一节第 314 页

MessageStoreProperties 类

此类为有用的消息存储库属性名称定义常量值。

语法

Visual Basic
Public Class **MessageStoreProperties**

C#
public class **MessageStoreProperties**

注释

MessageStoreProperties 类提供标准消息属性名称。您可以将 MessageProperties 字段传递给用于获取和设置预定义或自定义消息存储库属性的 QAManagerBase 方法。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

MessageStoreProperties 成员

公共静态字段（共享）

成员名称	说明
“MAX_DELIVERY_ATTEMPTS 字段”一节第 224 页	此属性名称是指不使用显式确认，在消息的状态设置为 StatusCodes.UNRECEIVABLE 之前，能够接收消息的最大次数。

公共构造函数

成员名称	说明
“MessageStoreProperties 构造函数”一节第 224 页	初始化 MessageStoreProperties 类的一个新实例。请参见“MessageStoreProperties 类”一节第 223 页。

MessageStoreProperties 构造函数

初始化 MessageStoreProperties 类的一个新实例。

语法

Visual Basic
Public Sub **New()**

C#
public **MessageStoreProperties();**

另请参见

- “MessageStoreProperties 类”一节第 223 页

MAX_DELIVERY_ATTEMPTS 字段

此属性名称是指不使用显式确认，在消息的状态设置为 StatusCodes.UNRECEIVABLE 之前，能够接收消息的最大次数。

语法

Visual Basic
Public Shared **MAX_DELIVERY_ATTEMPTS** As String

C#
public const string **MAX_DELIVERY_ATTEMPTS**;

注释

此字段的值为 ias_MaxDeliveryAttempts。

MessageType 枚举

为消息属性 MessageProperties.MSG_TYPE 定义常量值。

语法

Visual Basic
Public Enum **MessageType**

C#
public enum **MessageType**

成员名称

成员名称	说明
NETWORK_STATUS_NOTIFICATION	标识用于通知 QAnywhere 客户端应用程序网络状态更改的 QAnywhere 系统消息。
PUSH_NOTIFICATION	标识用于通知 QAnywhere 客户端应用程序推式通知的 QAnywhere 系统消息。
REGULAR	如果不存在任何消息类型属性，则假定消息类型为 REGULAR。

PropertyType 枚举

QAMessage 属性类型枚举，自然对应 C# 类型。

语法

Visual Basic
Public Enum **PropertyType**

C#
public enum **PropertyType**

成员名称

成员名称	说明
BOOLEAN	表示布尔型属性。
BYTE	表示有符号字节型属性。
DOUBLE	表示双精度型属性。
FLOAT	表示浮点型属性。
INT	表示整型属性。

成员名称	说明
LONG	表示长整型属性。
SHORT	表示短整型属性。
STRING	表示字符串型属性。
UNKNOWN	表示未知属性类型，原因通常是属性未知。

QABinaryMessage 接口

QABinaryMessage 对象用于发送包含未解释字节流的消息。它继承自类 QAMessage，并添加了字节消息主体。QABinaryMessage 提供了读取和写入字节消息主体的各种函数。

首次创建消息时，消息的主体处于只写模式。消息发送后，发送消息的客户端可保留并修改该消息，而不会影响已发送的消息。可以多次发送同一消息对象。

接收到消息时，提供程序已调用 QABinaryMessage.Reset()，因此消息主体处于只读模式并且从消息主体的开头开始读取值。

语法

Visual Basic
Public Interface **QABinaryMessage**

C#
public interface **QABinaryMessage**

QABinaryMessage 成员

公共属性

成员名称	说明
“BodyLength 属性”一节 第 228 页	返回消息主体的大小（以字节为单位）.

公共方法

成员名称	说明
“ReadBinary 方法”一节 第 229 页	从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节。
“ReadBoolean 方法”一节 第 230 页	从 QABinaryMessage 实例的消息主体的未读部分开始读取布尔值。

成员名称	说明
“ReadChar 方法”一节 第 231 页	从 QABinaryMessage 消息主体的未读部分开始读取字符值。
“ReadDouble 方法”一节 第 231 页	从 QABinaryMessage 消息主体的未读部分开始读取双精度值。
“ReadFloat 方法”一节 第 232 页	从 QABinaryMessage 消息主体的未读部分开始读取浮点值。
“ReadInt 方法”一节 第 232 页	从 QABinaryMessage 消息主体的未读部分开始读取整数值。
“ReadLong 方法”一节 第 233 页	从 QABinaryMessage 消息主体的未读部分开始读取长整型值。
“ReadSbyte 方法”一节 第 234 页	从 QABinaryMessage 消息主体的未读部分开始读取有符号字节值。
“ReadShort 方法”一节 第 234 页	从 QABinaryMessage 消息主体的未读部分开始读取短整型值。
“ReadString 方法”一节 第 235 页	从 QABinaryMessage 消息主体的未读部分开始读取字符串值。
“Reset 方法”一节第 235 页	重置消息以便从消息主体的开头开始读取值。
“WriteBinary 方法”一节 第 237 页	将字节数组值附加到 QABinaryMessage 实例的消息主体中。
“WriteBoolean 方法”一节 第 237 页	将布尔值附加到 QABinaryMessage 实例的消息主体中。
“WriteChar 方法”一节 第 238 页	将字符值附加到 QABinaryMessage 实例的消息主体中。
“WriteDouble 方法”一节 第 238 页	将双精度值附加到 QABinaryMessage 实例的消息主体中。
“WriteFloat 方法”一节 第 239 页	将浮点值附加到 QABinaryMessage 实例的消息主体中。
“WriteInt 方法”一节 第 240 页	将整数值附加到 QABinaryMessage 实例的消息主体中。
“WriteLong 方法”一节 第 240 页	将长整型值附加到 QABinaryMessage 实例的消息主体中。

成员名称	说明
“WriteSbyte 方法”一节 第 241 页	将有符号字节值附加到 QABinaryMessage 实例的消息主体中。
“WriteShort 方法”一节 第 241 页	将短整型值附加到 QABinaryMessage 实例的消息主体中。
“WriteString 方法”一节 第 242 页	将字符串值附加到 QABinaryMessage 实例的消息主体中。

BodyLength 属性

返回消息主体的大小（以字节为单位）.

语法

Visual Basic
Public Readonly Property **BodyLength** As Long

C#
public long **BodyLength** {get;}

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页

ReadBinary 方法

从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节。

语法

Visual Basic
```
Public Function ReadBinary( _
  ByVal bytes As Byte(), _
) As Integer
```

C#
```
public int ReadBinary(
  byte[] bytes
);
```

参数

- **bytes** 包含所读字节的字节数组。

返回值

从消息主体读取的字节数。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “WriteBinary 方法”一节第 237 页

ReadBinary 方法

从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节。

语法

Visual Basic

```
Public Function ReadBinary( _
   ByVal bytes As Byte(), _
   ByVal len As Integer _
) As Integer
```

C#

```
public int ReadBinary(
   byte[] bytes,
   int len
);
```

参数

- **bytes** 包含所读字节的字节数组。
- **len** 要读取的最大字节数。

返回值

从消息主体读取的字节数。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “WriteBinary 方法”一节第 237 页

ReadBinary 方法

从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节。

语法

Visual Basic
```
Public Function ReadBinary( _
  ByVal bytes As Byte(), _
  ByVal offset As Integer, _
  ByVal len As Integer _
) As Integer
```

C#
```
public int ReadBinary(
  byte[] bytes,
  int offset,
  int len
);
```

参数

- **bytes** 包含所读字节的字节数组。
- **offset** 目标数组的起始偏移。
- **len** 要读取的最大字节数。

返回值

从消息主体读取的字节数。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “WriteBinary 方法”一节第 237 页

ReadBoolean 方法

从 QABinaryMessage 实例的消息主体的未读部分开始读取布尔值。

语法

Visual Basic
```
Public Function ReadBoolean() As Boolean
```

C#
```
public bool ReadBoolean();
```


返回值

从消息主体读取的布尔值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteBoolean 方法”一节第 237 页

ReadChar 方法

从 QABinaryMessage 消息主体的未读部分开始读取字符值。

语法

Visual Basic
Public Function **ReadChar()** As Char

C#
public char **ReadChar();**

返回值

从消息主体读取的字符值。

异常

- “QAException 类”一节第 243 页- 读取值时发生转换错误或没有其它输入时。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteChar 方法”一节第 238 页

ReadDouble 方法

从 QABinaryMessage 消息主体的未读部分开始读取双精度值。

语法

Visual Basic
Public Function **ReadDouble()** As Double

C#
public double **ReadDouble();**

返回值

从消息主体读取的双精度值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteDouble 方法”一节第 238 页

ReadFloat 方法

从 QABinaryMessage 消息主体的未读部分开始读取浮点值。

语法

Visual Basic
Public Function **ReadFloat()** As Single

C#
public float **ReadFloat();**

返回值

从消息主体读取的浮点值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteFloat 方法”一节第 239 页

ReadInt 方法

从 QABinaryMessage 消息主体的未读部分开始读取整数值。

语法

Visual Basic
Public Function **ReadInt()** As Integer

C#
public int **ReadInt();**

返回值

从消息主体读取的整型值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteInt 方法”一节第 240 页

ReadLong 方法

从 QABinaryMessage 消息主体的未读部分开始读取长整型值。

语法

Visual Basic
Public Function **ReadLong()** As Long

C#
public long **ReadLong();**

返回值

从消息主体读取的长整型值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteLong 方法”一节第 240 页

ReadSbyte 方法

从 QABinaryMessage 消息主体的未读部分开始读取有符号字节值。

语法

Visual Basic
Public Function **ReadSbyte()** As System.SByte

C#
public System.Sbyte **ReadSbyte();**

返回值

从消息主体读取的有符号字节值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteSbyte 方法”一节第 241 页

ReadShort 方法

从 QABinaryMessage 消息主体的未读部分开始读取短整型值。

语法

Visual Basic
Public Function **ReadShort()** As Short

C#
public short **ReadShort();**

返回值

从消息主体读取的短整型值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteShort 方法”一节第 241 页

ReadString 方法

从 QABinaryMessage 消息主体的未读部分开始读取字符串值。

语法

Visual Basic
Public Function **ReadString()** As String

C#
public string **ReadString();**

返回值

从消息主体读取的字符串值。

异常

- “QAException 类”一节第 243 页- 如果读取值时发生转换错误或没有其它输入，则抛出。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “WriteString 方法”一节第 242 页

Reset 方法

重置消息以便从消息主体的开头开始读取值。

语法

Visual Basic
Public Sub **Reset()**

C#
public void **Reset();**

注释

Reset 方法还将 QABinaryMessage 消息主体置于只读模式。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页

WriteBinary 方法

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic
```
Public Sub WriteBinary( _
  ByVal val As Byte() _
)
```

C#
```
public void WriteBinary(
  byte[] val
);
```

参数

- **val** 写入消息主体的字节数组值。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “ReadBinary 方法”一节第 229 页

WriteBinary 方法

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic
```
Public Sub WriteBinary( _
  ByVal val As Byte(), _
  ByVal len As Integer _
)
```

C#
```
public void WriteBinary(
  byte[] val,
  int len
);
```

参数

- **val** 写入消息主体的字节数组值。
- **len** 要写入的字节数。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “ReadBinary 方法”一节第 229 页

WriteBinary 方法

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteBinary( _
  ByVal val As Byte(), _
  ByVal offset As Integer, _
  ByVal len As Integer _
)
```

C#

```
public void WriteBinary(
  byte[] val,
  int offset,
  int len
);
```

参数

- **val** 写入消息主体的字节数组值。
- **offset** 要开始写入的字节数组内的偏移。
- **len** 要写入的字节数。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “ReadBinary 方法”一节第 229 页

WriteBoolean 方法

将布尔值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteBoolean( _
  ByVal val As Boolean _
)
```

C#

```
public void WriteBoolean(
  bool val
);
```

参数

- **val** 要写入消息主体的布尔值。

注释

布尔值由一个单字节值表示。1 代表 true；0 代表 false。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadBoolean 方法”一节第 230 页

WriteChar 方法

将字符值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteChar( _
  ByVal val As Char _
)
```

C#

```
public void WriteChar(
  char val
);
```

参数

- **val** 要写入消息主体的字符值。

注释

字符由一个双字节值表示，首先附加高位字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadChar 方法”一节第 231 页

WriteDouble 方法

将双精度值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteDouble( _
  ByVal val As Double _
)
```


C#
```
public void WriteDouble(
  double val
);
```

参数

- **val** 要写入消息主体的双精度值。

注释

双精度值可转换为 8 字节长整型的表示形式，首先附加高位字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadDouble 方法”一节第 231 页

WriteFloat 方法

将浮点值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic
```
Public Sub WriteFloat( _
  ByVal val As Single _
)
```

C#
```
public void WriteFloat(
  float  val
);
```

参数

- **val** 要写入消息主体的浮点值。

注释

浮点参数可转换为 4 字节整型表示形式，首先附加高位字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadFloat 方法”一节第 232 页

WriteInt 方法

将整数值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic
```
Public Sub WriteInt( _
  ByVal val As Integer _
)
```

C#
```
public void WriteInt(
  int val
);
```

参数

- **val** 要写入消息主体的整型值。

注释

整数参数由一个 4 字节值表示，首先附加高位字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadInt 方法”一节第 232 页

WriteLong 方法

将长整型值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic
```
Public Sub WriteLong( _
  ByVal val As Long _
)
```

C#
```
public void WriteLong(
  long val
);
```

参数

- **val** 要写入消息主体的长整型值。

注释

长整型参数由一个 8 字节值表示，首先附加高位字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadLong 方法”一节第 233 页

WriteSbyte 方法

将有符号字节值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteSbyte( _
  ByVal val As System.SByte _
)
```

C#

```
public void WriteSbyte(
  System.Sbyte val
);
```

参数

- **val** 要写入消息主体的有符号字节值。

注释

有符号字节由一个单字节值表示。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadSbyte 方法”一节第 234 页

WriteShort 方法

将短整型值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteShort( _
  ByVal val As Short _
)
```

C#

```
public void WriteShort(
```

```
  short val
);
```

参数

- **val** 要写入消息主体的短整型值。

注释

短整型参数由一个双字节值表示，首先附加高位字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadShort 方法”一节第 234 页

WriteString 方法

将字符串值附加到 QABinaryMessage 实例的消息主体中。

语法

Visual Basic

```
Public Sub WriteString( _
  ByVal val As String _
)
```

C#

```
public void WriteString(
  string  val
);
```

参数

- **val** 要写入消息主体的字符串值。

注释

*注意：*接收应用程序需要为每个 WriteString 调用而调用 ReadString。请参见“ReadString 方法”一节第 235 页。

注意： 要写入的字符串的 UTF-8 表示形式最多可为 32767 个字节。

另请参见

- “QABinaryMessage 接口”一节第 226 页
- “QABinaryMessage 成员”一节第 226 页
- “QABinaryMessage 接口”一节第 226 页
- “ReadString 方法”一节第 235 页

QAException 类

封装 QAnywhere 客户端应用程序异常。可以使用 QAException 类捕获 QAnywhere 异常。

语法

Visual Basic
Public Class **QAException**
  Inherits ApplicationException

C#
public class **QAException** :
  ApplicationException

QAException 成员

公共构造函数

成员名称	说明
“QAException 构造函数”一节 第 244 页	创建提供错误消息文本的 QAException 实例。
“QAException 构造函数”一节 第 244 页	创建提供错误代码和错误消息文本的 QAException 实例。

公共属性

成员名称	说明
“DetailedMessage 属性”一节 第 252 页	异常的详细描述。
“ErrorCode 属性”一节 第 252 页	异常的错误代码。
“NativeErrorCode 属性”一节 第 252 页	异常的本地错误代码。
HelpLink（继承自 Exception）	获取或设置可转到与此异常相关的帮助文件的链接。
InnerException（继承自 Exception）	获取导致当前异常的 System.Exception 实例。
Message（继承自 Exception）	获取描述当前异常的消息。
Source（继承自 Exception）	获取或设置引起错误的应用程序或对象的名称。

成员名称	说明
StackTrace（继承自 Exception）	获取在抛出当前异常时调用堆栈上的框架的字符串表示形式。
TargetSite（继承自 Exception）	获取抛出当前异常的方法。

公共方法

成员名称	说明
GetBaseException（继承自 Exception）	当在派生类中被替换时，返回 System.Exception，它是导致产生一个或多个后续异常的根本原因。
GetObjectData（继承自 Exception）	在派生类中被替换时，利用有关异常的信息设置 System.Runtime.Serialization.SerializationInfo。
ToString（继承自 Exception）	创建并返回当前异常的字符串表示形式。

QAException 构造函数

创建提供错误消息文本的 QAException 实例。

语法

Visual Basic

```
Overloads Public Sub New( _
  ByVal msg As String _
)
```

C#

```
public QAException(
  string  msg
);
```

参数

- **msg** 异常的文本描述。

QAException 构造函数

创建提供错误代码和错误消息文本的 QAException 实例。

语法

Visual Basic

```
Overloads Public Sub New( _
  ByVal msg As String, _
  ByVal errCode As Integer _
)
```

C#
```
public QAException(
  string  msg,
  int errCode
);
```

参数

- **msg** 异常的文本描述。
- **errCode** 错误代码。

COMMON_ALREADY_OPEN_ERROR 字段

QAManager 已打开。

语法

Visual Basic
```
Public Shared COMMON_ALREADY_OPEN_ERROR As Integer
```

C#
```
public const int COMMON_ALREADY_OPEN_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR 字段

获取队列深度时出错。

语法

Visual Basic
```
Public Shared COMMON_GETQUEUEDEPTH_ERROR As Integer
```

C#
```
public const int COMMON_GETQUEUEDEPTH_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 字段

当过滤器为 ALL 时，无法在给定目标上使用 QAManagerBase.getQueueDepth。

语法

Visual Basic
```
Public Shared COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG As Integer
```

C#
```
public const int COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG;
```

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 字段

在未设置消息存储库 ID 时，无法使用 QAManagerBase.getQueueDepth。

语法

Visual Basic
Public Shared **COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID** As Integer

C#
public const int **COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID**;

COMMON_GET_INIT_FILE_ERROR 字段

无法访问客户端属性文件。

语法

Visual Basic
Public Shared **COMMON_GET_INIT_FILE_ERROR** As Integer

C#
public const int **COMMON_GET_INIT_FILE_ERROR**;

COMMON_GET_PROPERTY_ERROR 字段

从消息存储库中检索属性时出错。

语法

Visual Basic
Public Shared **COMMON_GET_PROPERTY_ERROR** As Integer

C#
public const int **COMMON_GET_PROPERTY_ERROR**;

COMMON_INIT_ERROR 字段

初始化错误。

语法

Visual Basic
Public Shared **COMMON_INIT_ERROR** As Integer

C#
public const int **COMMON_INIT_ERROR**;

COMMON_INIT_THREAD_ERROR 字段

初始化后台线程时出错。

语法

Visual Basic
Public Shared **COMMON_INIT_THREAD_ERROR** As Integer

C#
public const int **COMMON_INIT_THREAD_ERROR**;

COMMON_INVALID_PROPERTY 字段

客户端属性文件中有一个无效的属性。

语法

Visual Basic
Public Shared **COMMON_INVALID_PROPERTY** As Integer

C#
public const int **COMMON_INVALID_PROPERTY**;

COMMON_MSG_ACKNOWLEDGE_ERROR 字段

确认消息时出错。

语法

Visual Basic
Public Shared **COMMON_MSG_ACKNOWLEDGE_ERROR** As Integer

C#
public const int **COMMON_MSG_ACKNOWLEDGE_ERROR**;

COMMON_MSG_CANCEL_ERROR 字段

取消消息时出错。

语法

Visual Basic
Public Shared **COMMON_MSG_CANCEL_ERROR** As Integer

C#
public const int **COMMON_MSG_CANCEL_ERROR**;

COMMON_MSG_CANCEL_ERROR_SENT 字段

取消消息时出错。不能取消已发送的消息。

语法

Visual Basic
Public Shared **COMMON_MSG_CANCEL_ERROR_SENT** As Integer

C#
public const int **COMMON_MSG_CANCEL_ERROR_SENT**;

COMMON_MSG_NOT_WRITEABLE_ERROR 字段

不能写入到消息，因为它处于只读模式。

语法

Visual Basic
Public Shared **COMMON_MSG_NOT_WRITEABLE_ERROR** As Integer

C#
public const int **COMMON_MSG_NOT_WRITEABLE_ERROR**;

COMMON_MSG_RETRIEVE_ERROR 字段

从客户端消息存储库中检索消息时出错。

语法

Visual Basic
Public Shared **COMMON_MSG_RETRIEVE_ERROR** As Integer

C#
public const int **COMMON_MSG_RETRIEVE_ERROR**;

COMMON_MSG_STORE_NOT_INITIALIZED 字段

尚未为进行消息传递而初始化消息存储库。

语法

Visual Basic
Public Shared **COMMON_MSG_STORE_NOT_INITIALIZED** As Integer

C#
public const int **COMMON_MSG_STORE_NOT_INITIALIZED**;

COMMON_MSG_STORE_TOO_LARGE 字段

消息存储库相对于设备上的可用磁盘空间过大。

语法

Visual Basic
Public Shared **COMMON_MSG_STORE_TOO_LARGE** As Integer

C#
public const int **COMMON_MSG_STORE_TOO_LARGE**;

COMMON_NOT_OPEN_ERROR 字段

QAManager 没有打开。

语法

Visual Basic
Public Shared **COMMON_NOT_OPEN_ERROR** As Integer

C#
public const int **COMMON_NOT_OPEN_ERROR**;

COMMON_NO_DEST_ERROR 字段

没有目标。

语法

Visual Basic
Public Shared **COMMON_NO_DEST_ERROR** As Integer

C#
public const int **COMMON_NO_DEST_ERROR**;

COMMON_NO_IMPLEMENTATION 字段

未实现函数。

语法

Visual Basic
Public Shared **COMMON_NO_IMPLEMENTATION** As Integer

C#
public const int **COMMON_NO_IMPLEMENTATION**;

COMMON_OPEN_ERROR 字段

打开消息存储库连接时出错。

语法

Visual Basic
Public Shared **COMMON_OPEN_ERROR** As Integer

C#
public const int **COMMON_OPEN_ERROR**;

COMMON_OPEN_LOG_FILE_ERROR 字段

打开日志文件时出错。

语法

Visual Basic
Public Shared **COMMON_OPEN_LOG_FILE_ERROR** As Integer

C#
public const int **COMMON_OPEN_LOG_FILE_ERROR**;

COMMON_OPEN_MAXTHREADS_ERROR 字段

无法打开 QAManager，因为并发服务器请求的最大数不够大。有关详细信息，请参见“-gn 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

语法

Visual Basic
Public Shared **COMMON_OPEN_MAXTHREADS_ERROR** As Integer

C#
public const int **COMMON_OPEN_MAXTHREADS_ERROR**;

COMMON_SELECTOR_SYNTAX_ERROR 字段

给定选择程序存在一个语法错误。

语法

Visual Basic
Public Shared **COMMON_SELECTOR_SYNTAX_ERROR** As Integer

C#
public const int **COMMON_SELECTOR_SYNTAX_ERROR**;

COMMON_SET_PROPERTY_ERROR 字段

将属性存储到消息存储库中时出错。

语法

Visual Basic
Public Shared **COMMON_SET_PROPERTY_ERROR** As Integer

C#
public const int **COMMON_SET_PROPERTY_ERROR**;

COMMON_TERMINATE_ERROR 字段

终止错误。

语法

Visual Basic
Public Shared **COMMON_TERMINATE_ERROR** As Integer

C#
public const int **COMMON_TERMINATE_ERROR**;

COMMON_UNEXPECTED_EOM_ERROR 字段

遇到了意外的消息结尾。

语法

Visual Basic
Public Shared **COMMON_UNEXPECTED_EOM_ERROR** As Integer

C#
public const int **COMMON_UNEXPECTED_EOM_ERROR**;

COMMON_UNREPRESENTABLE_TIMESTAMP 字段

时间戳在可接受范围以外。

语法

Visual Basic
Public Shared **COMMON_UNREPRESENTABLE_TIMESTAMP** As Integer

C#
public const int **COMMON_UNREPRESENTABLE_TIMESTAMP**;

QA_NO_ERROR 字段

无错误。

语法

Visual Basic
Public Shared **QA_NO_ERROR** As Integer

C#
public const int **QA_NO_ERROR**;

DetailedMessage 属性

异常的详细描述。

语法

Visual Basic
Public Readonly Property **DetailedMessage** As String

C#
public string **DetailedMessage** {get;}

ErrorCode 属性

异常的错误代码。

语法

Visual Basic
Public Readonly Property **ErrorCode** As Integer

C#
public int **ErrorCode** {get;}

NativeErrorCode 属性

异常的本地错误代码。

语法

Visual Basic
Public Readonly Property **NativeErrorCode** As Integer

C#
public int **NativeErrorCode** {get;}

QAManager 接口

QAManager 类由 QAManagerBase 派生并且管理非事务性 QAnywhere 消息传递操作。

语法

Visual Basic
Public Interface **QAManager**

C#
public interface **QAManager**

注释

有关派生行为的详细说明，请参见“QAManagerBase 接口”一节第 257 页。

可以如 AcknowledgementMode 类中定义的那样对 QAManager 进行配置，以进行隐式或显式确认。要将消息作为事务的一部分进行确认，请使用 QATransactionalManager。使用 QAManagerFactory 创建 QAManager 和 QATransactionalManager 对象。

另请参见

- “QAManager 成员”一节第 253 页
- “AcknowledgementMode 枚举”一节第 212 页
- “QATransactionalManager 接口”一节第 328 页

QAManager 成员

公共方法

成员名称	说明
“Acknowledge 方法”一节第 254 页	确认客户端应用程序成功接收了一条 QAnywhere 消息。
“AcknowledgeAll 方法”一节第 254 页	确认客户端应用程序成功接收了 QAnywhere 消息。确认所有未确认的消息。
“AcknowledgeUntil 方法”一节第 255 页	对在给定消息之前收到的给定 QAMessage 实例和所有未确认消息进行确认。
“Open 方法”一节第 256 页	使用给定的 AcknowledgementMode 值打开 QAManager。
“Recover 方法”一节第 257 页	强制将所有未确认的消息转为未接收状态。

Acknowledge 方法

确认客户端应用程序成功接收了一条 QAnywhere 消息。

语法

Visual Basic
```
Public Sub Acknowledge( _
  ByVal msg As QAMessage _
)
```

C#
```
public void Acknowledge(
  QAMessage msg
);
```

参数

- **msg** 要确认的消息。

注释

QAMessage 经过确认后，其状态属性变为 StatusCodes.RECEIVED。当 QAMessage MessageProperties.STATUS 消息属性更改为 StatusCodes.RECEIVED 时，可以使用缺省删除规则将其删除。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

异常

- “QAException 类”一节第 243 页- 如果确认消息时存在问题，则抛出。

另请参见

- “QAManager 接口”一节第 253 页
- “QAManager 成员”一节第 253 页
- “QAManager 接口”一节第 253 页
- “AcknowledgeUntil 方法”一节第 255 页
- “StatusCodes 枚举”一节第 331 页
- “MessageProperties 类”一节第 215 页
- “AcknowledgeAll 方法”一节第 254 页

AcknowledgeAll 方法

确认客户端应用程序成功接收了 QAnywhere 消息。确认所有未确认的消息。

语法

Visual Basic
```
Public Sub AcknowledgeAll()
```

C#
```
public void AcknowledgeAll();
```


注释

QAMessage 经过确认后，其 MessageProperties.STATUS 属性变为 StatusCodes.RECEIVED。当 QAMessage 状态更改为 StatusCodes.RECEIVED 时，可以使用缺省删除规则将其删除。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

异常

- “QAException 类”一节第 243 页- 如果确认消息时存在问题，则抛出。

另请参见

- “QAManager 接口”一节第 253 页
- “QAManager 成员”一节第 253 页
- “QAManager 接口”一节第 253 页
- “Acknowledge 方法”一节第 254 页
- “AcknowledgeUntil 方法”一节第 255 页
- “StatusCodes 枚举”一节第 331 页
- “MessageProperties 类”一节第 215 页

AcknowledgeUntil 方法

对在给定消息之前收到的给定 QAMessage 实例和所有未确认消息进行确认。

语法

Visual Basic

```
Public Sub AcknowledgeUntil( _
  ByVal msg As QAMessage _
)
```

C#

```
public void AcknowledgeUntil(
  QAMessage msg
);
```

参数

- **msg** 要确认的最后一条消息。还会确认所有以前未确认的消息。

注释

QAMessage 经过确认后，其 MessageProperties.STATUS 属性变为 StatusCodes.RECEIVED。当 QAMessage 状态更改为 StatusCodes.RECEIVED 时，可以使用缺省删除规则将其删除。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

异常

- “QAException 类”一节第 243 页- 如果确认消息时存在问题，则抛出。

另请参见

- “QAManager 接口”一节第 253 页
- “QAManager 成员”一节第 253 页
- “QAManager 接口”一节第 253 页
- “Acknowledge 方法”一节第 254 页
- “AcknowledgeAll 方法”一节第 254 页
- “StatusCodes 枚举”一节第 331 页
- “MessageProperties 类”一节第 215 页

Open 方法

使用给定的 AcknowledgementMode 值打开 QAManager。

语法

Visual Basic

```
Public Sub Open( _
  ByVal mode As AcknowledgementMode _
)
```

C#

```
public void Open(
  AcknowledgementMode mode
);
```

参数

- **mode** 确认模式，AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 或 AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT 中的一个。

注释

Open 方法必须是创建 QAManager 后调用的第一个方法。

如果检测到数据库连接错误，则可以通过调用 Close 方法并随后调用 open 方法来重新打开 QAManager。重新打开 QAManager 时，不需要重新创建它、重置属性或重置消息监听器。第一次执行 Open 后便无法更改 QAManager 的属性，后续 Open 调用必须提供相同的确认模式。

异常

- “QAException 类”一节第 243 页- 如果打开 QAManager 实例时存在问题，则抛出。

另请参见

- “QAManager 接口”一节第 253 页
- “QAManager 成员”一节第 253 页
- “Close 方法”一节第 266 页

Recover 方法

强制将所有未确认的消息转为未接收状态。

语法

Visual Basic
Public Sub **Recover()**

C#
public void **Recover();**

注释

必须使用 QAManagerBase.GetMessage 再次接收这些消息。

异常

- “QAException 类”一节第 243 页- 如果恢复时存在问题，则抛出。

另请参见

- “QAManager 接口”一节第 253 页
- “QAManager 成员”一节第 253 页
- “QAManager 接口”一节第 253 页
- “GetMessage 方法”一节第 272 页

QAManagerBase 接口

此类用作 QATransactionalManager 和 QAManager 的基类，它们分别管理事务性和非事务性消息传递。

语法

Visual Basic
Public Interface **QAManagerBase**

C#
public interface **QAManagerBase**

注释

使用 QAManagerBase.Start() 方法可允许 QAManagerBase 实例监听消息。应用程序中的每个线程必须只有一个 QAManagerBase 实例。

您可以使用此类的实例来创建和管理 QAnywhere 消息。使用 QAManagerBase.CreateBinaryMessage() 方法和 QAManagerBase.CreateTextMessage() 方法来创建合适的 QAMessage 实例。QAMessage 实例提供设置消息内容和属性的各种方法。

要发送 QAnywhere 消息，可使用 QAManagerBase.PutMessage 方法，将已指定地址的消息放置在本地消息存储库队列中。此消息由 QAnywhere 代理根据其传输策略传输，或在您调用 QAManagerBase.TriggerSendReceive() 时传输。

有关 qaagent 传输策略的详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

使用 QAManagerBase.Close 方法关闭 QAManagerBase 实例时，消息将从内存中释放。

当发生 QAException 时，可以使用 QAManagerBase.LastError 和 QAManagerBase.LastErrorMessage 来返回错误信息。您还可以从 QAException 对象获取错误信息。

QAManagerBase 还提供设置和获取消息存储库属性的方法。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页和 MessageStoreProperties 类。

另请参见

- “QAManagerBase 成员”一节第 258 页
- “CreateBinaryMessage 方法”一节第 267 页
- “TriggerSendReceive 方法”一节第 297 页
- “Close 方法”一节第 266 页
- “QAException 类”一节第 243 页

QAManagerBase 成员

公共属性

成员名称	说明
“Mode 属性”一节第 261 页	返回所接收消息的 QAManager 确认模式。

公共方法

成员名称	说明
“BrowseMessages 方法”一节第 262 页	浏览消息存储库中的全部可用消息。
“BrowseMessages 方法”一节第 262 页	不建议使用此方法。请改用 BrowseMessagesByQueue(string) 方法。
“BrowseMessagesByID 方法”一节第 263 页	浏览具有给定消息 ID 的消息。
“BrowseMessagesByQueue 方法”一节第 264 页	浏览正在等待的已发送到给定地址的下一条可用消息。
“BrowseMessagesBySelector 方法”一节第 265 页	浏览消息存储库中满足给定选择程序的排队消息。
“CancelMessage 方法”一节第 265 页	取消具有给定消息 ID 的消息。

成员名称	说明
“Close 方法”一节第 266 页	关闭到 QAnywhere 消息系统的连接并释放 QAManagerBase 使用的所有资源。
“CreateBinaryMessage 方法”一节第 267 页	创建一个 QABinaryMessage 对象。
“CreateTextMessage 方法”一节第 267 页	创建一个 QATextMessage 对象。
“GetBooleanStoreProperty 方法”一节第 268 页	获取预定义或自定义消息存储库属性的布尔值。
“GetDoubleStoreProperty 方法”一节第 269 页	获取预定义或自定义消息存储库属性的双精度值。
“GetFloatStoreProperty 方法”一节第 269 页	获取预定义或自定义消息存储库属性的浮点值。
“GetIntStoreProperty 方法”一节第 270 页	获取预定义或自定义消息存储库属性的整型值。
“GetLongStoreProperty 方法”一节第 271 页	获取预定义或自定义消息存储库属性的长整型值。
“GetMessage 方法”一节第 272 页	返回下一个发送到指定地址的可用 QAMessage。
“GetMessageBySelector 方法”一节第 272 页	返回下一个发送到指定地址并满足给定选择程序的可用 QAMessage。
“GetMessageBySelectorNoWait 方法”一节第 273 页	返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。
“GetMessageBySelectorTimeout 方法”一节第 274 页	返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。
“GetMessageNoWait 方法”一节第 275 页	返回下一个发送到给定地址的可用 QAMessage。
“GetMessageTimeout 方法”一节第 276 页	返回下一个发送到给定地址的可用 QAMessage。
“GetQueueDepth 方法”一节第 277 页	基于给定过滤器，返回队列深度。

成员名称	说明
“GetQueueDepth 方法”一节第 278 页	基于给定过滤器，返回所有队列的深度总和。
“GetSbyteStoreProperty 方法”一节第 278 页	获取预定义或自定义消息存储库属性的有符号字节值。
“GetShortStoreProperty 方法”一节第 279 页	获取预定义或自定义消息存储库属性的短整型值。
“GetStoreProperty 方法”一节第 280 页	获取表示消息存储库属性的 System.Object。
“GetStorePropertyNames 方法”一节第 281 页	获取消息存储库属性名称的枚举器。
“GetStringStoreProperty 方法”一节第 281 页	获取预定义或自定义消息存储库属性的字符串值。
“PropertyExists 方法”一节第 282 页	检查某一给定的属性是否具有值。
“PutMessage 方法”一节第 283 页	准备将发送给另一 QAnywhere 客户端的消息。
“PutMessageTimeToLive 方法”一节第 283 页	准备将发送给另一 QAnywhere 客户端的消息。
“SetBooleanStoreProperty 方法”一节第 284 页	将预定义或自定义消息存储库属性设置为布尔值。
“SetDoubleStoreProperty 方法”一节第 285 页	将预定义或自定义消息存储库属性设置为双精度值。
“SetExceptionListener 方法”一节第 286 页	将 ExceptionListener 委派设置为在异步处理 QAnywhere 消息时，接收 QAExceptions。请参见“ExceptionListener 委派”一节第 213 页。
“SetExceptionListener2 方法”一节第 286 页	将 ExceptionListener2 委派设置为在异步处理 QAnywhere 消息时，接收 QAExceptions。请参见“ExceptionListener2 委派”一节第 213 页。
“SetFloatStoreProperty 方法”一节第 287 页	将预定义或自定义消息存储库属性设置为浮点值。
“SetIntStoreProperty 方法”一节第 288 页	将预定义或自定义消息存储库属性设置为整型值。

成员名称	说明
“SetLongStoreProperty 方法”一节第 288 页	将预定义或自定义消息存储库属性设置为长整型值。
“SetMessageListener 方法”一节第 289 页	将 MessageListener 委派设置为异步接收 QAnywhere 消息。请参见“MessageListener 委派”一节第 214 页。
“SetMessageListener2 方法”一节第 290 页	将 MessageListener2 委派设置为异步接收 QAnywhere 消息。请参见“MessageListener2 委派”一节第 214 页。
“SetMessageListenerBySelector 方法”一节第 291 页	将 MessageListener 委派设置为使用消息选择程序异步接收 QAnywhere 消息。请参见“MessageListener 委派”一节第 214 页。
“SetMessageListenerBySelector 2 方法”一节第 292 页	将 MessageListener2 委派设置为使用消息选择程序异步接收 QAnywhere 消息。请参见“MessageListener2 委派”一节第 214 页。
“SetProperty 方法”一节第 293 页	用于通过编程方式设置 QAnywhere Manager 配置属性。
“SetSbyteStoreProperty 方法”一节第 293 页	将预定义或自定义消息存储库属性设置为 sbyte 值。
“SetShortStoreProperty 方法”一节第 294 页	将预定义或自定义消息存储库属性设置为短整型值。
“SetStoreProperty 方法”一节第 295 页	将预定义或自定义消息存储库属性设置为 System.Object 值。
“SetStringStoreProperty 方法”一节第 296 页	将预定义或自定义消息存储库属性设置为字符串值。
“Start 方法”一节第 296 页	启动 QAManagerBase 来接收消息监听器中进来的消息。
“Stop 方法”一节第 297 页	使 QAManagerBase 停止接收进来的消息。
“TriggerSendReceive 方法”一节第 297 页	在上载任何发送给其它客户端的消息和下载任何发送给本地客户端的消息时，皆与 QAnywhere 消息服务器保持同步。

Mode 属性

返回所接收消息的 QAManager 确认模式。

语法

Visual Basic
Public Readonly Property **Mode** As AcknowledgementMode

C#
public AcknowledgementMode **Mode** {get;}

注释

有关可能值的列表，请参见“AcknowledgementMode 枚举”一节第 212 页。
AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 和
AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT 适用于 QAManager 实例；
AcknowledgementMode.TRANSACTIONAL 是适用于 QATransactionalManager 实例的模式。

BrowseMessages 方法

浏览消息存储库中的全部可用消息。

语法

Visual Basic
Overloads Public Function **BrowseMessages()** As System.Collections.IEnumerator

C#
public System.Collections.IEnumerator **BrowseMessages();**

返回值

可用消息的枚举器。

注释

正在浏览消息，因此无法对这些消息进行确认。由于浏览消息会分配本机资源，因此应在完成时调用枚举器的 Reset() 方法。如果不调用此方法，则直到释放此 QAManagerBase 对象时才会释放本地资源。

使用 QAManagerBase.GetMessage 接收消息，以便可以对这些消息进行确认。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “BrowseMessagesByQueue 方法”一节第 264 页
- “BrowseMessagesByID 方法”一节第 263 页
- “BrowseMessages 方法”一节第 262 页

BrowseMessages 方法

不建议使用此方法。请改用 BrowseMessagesByQueue(string) 方法。

语法

Visual Basic
```
Overloads Public Function BrowseMessages( _
  ByVal address As String _
) As System.Collections.IEnumerator
```

C#
```
public System.Collections.IEnumerator BrowseMessages(
  string  address
);
```

参数

- **address**　消息的地址。

返回值

可用消息的枚举器。

注释

浏览正在等待的已发送到给定地址的下一个可用消息。address 参数采用 store-id\queue-name 或 queue-name 的形式。正在浏览消息，因此无法对这些消息进行确认。

由于浏览消息分配了本机资源，应该在完成时，调用枚举器的 Reset() 方法。如果不调用此方法，则直到释放此 QAManagerBase 对象时才会释放本地资源。

使用 QAManagerBase.GetMessage 接收消息，以便可以对这些消息进行确认。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “BrowseMessagesByQueue 方法”一节第 264 页
- “BrowseMessagesByID 方法”一节第 263 页
- “BrowseMessagesBySelector 方法”一节第 265 页
- “BrowseMessages 方法”一节第 262 页

BrowseMessagesByID 方法

浏览具有给定消息 ID 的消息。

语法

Visual Basic
```
Public Function BrowseMessagesByID( _
  ByVal msgid As String _
) As System.Collections.IEnumerator
```

C#
```
public System.Collections.IEnumerator BrowseMessagesByID(
  string  msgid
);
```

参数

- **msgid** 消息的消息 ID。

返回值

包含 0 或 1 消息的枚举器。

注释

正在浏览消息，因此无法对其进行确认。由于浏览消息会分配本机资源，因此应在完成时调用枚举器的 Reset() 方法。如果不调用此方法，则直到释放此 QAManagerBase 对象时才会释放本地资源。

使用 QAManagerBase.GetMessage 接收消息，以便可以对这些消息进行确认。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “BrowseMessagesByQueue 方法”一节第 264 页
- “BrowseMessages 方法”一节第 262 页
- “BrowseMessages 方法”一节第 262 页

BrowseMessagesByQueue 方法

浏览正在等待的已发送到给定地址的下一条可用消息。

语法

Visual Basic

```
Public Function BrowseMessagesByQueue( _
  ByVal address As String _
) As System.Collections.IEnumerator
```

C#

```
public System.Collections.IEnumerator BrowseMessagesByQueue(
  string  address
);
```

参数

- **address** 消息的地址。

返回值

可用消息的枚举器。

注释

正在浏览消息，因此无法对这些消息进行确认。由于浏览消息会分配本机资源，因此应在完成时调用枚举器的 Reset() 方法。如果不调用此方法，则直到释放此 QAManagerBase 对象时才会释放本地资源。

使用 QAManagerBase.GetMessage 接收消息，以便可以对这些消息进行确认。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “BrowseMessagesByID 方法”一节第 263 页
- “BrowseMessages 方法”一节第 262 页
- “BrowseMessages 方法”一节第 262 页

BrowseMessagesBySelector 方法

浏览消息存储库中满足给定选择程序的排队消息。

语法

Visual Basic

```
Public Function BrowseMessagesBySelector( _
  ByVal selector As String _
) As System.Collections.IEnumerator
```

C#

```
public System.Collections.IEnumerator BrowseMessagesBySelector(
  string  selector
);
```

参数

- **selector**　选择程序。

返回值

可用消息的枚举器。

注释

正在浏览消息，因此无法对其进行确认。由于浏览消息分配了本机资源，应该在完成时，调用枚举器的 Reset() 方法。如果不调用此方法，则直到释放此 QAManagerBase 对象时才会释放本地资源。

使用 QAManagerBase.GetMessage 接收消息，以便可以对这些消息进行确认。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “BrowseMessagesByQueue 方法”一节第 264 页
- “BrowseMessages 方法”一节第 262 页
- “BrowseMessages 方法”一节第 262 页
- “BrowseMessagesByID 方法”一节第 263 页

CancelMessage 方法

取消具有给定消息 ID 的消息。

语法

Visual Basic
```
Public Sub CancelMessage( _
  ByVal msgid As String _
)
```

C#
```
public void CancelMessage(
  string  msgid
);
```

参数

- **msgid** 要取消的消息的消息 ID。

注释

CancelMessage 在消息传输前将其置于已取消状态。当使用 QAnywhere 代理的缺省删除规则时，已取消的消息最终会从消息存储库中删除。

如果消息已经处于最终状态，或已经传输到中央消息传递服务器，则 CancelMessage 失败。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

异常

- “QAException 类”一节第 243 页- 如果取消消息时存在问题，则抛出。

Close 方法

关闭到 QAnywhere 消息系统的连接并释放 QAManagerBase 使用的所有资源。

语法

Visual Basic
```
Public Sub Close()
```

C#
```
public void Close();
```

注释

除对 Close() 的第一次调用外，其余对 Close() 的调用都将被忽略。随后的任何对 QAManagerBase 方法（Close() 除外）的调用都会导致出现 QAException。在这种情况下，您必须创建并打开一个新的 QAManagerBase 实例。

如果检测到数据库连接错误，则可以通过调用 Close 方法并随后调用 Open 方法来重新打开 QAManager。重新打开 QAManager 时，不需要重新创建它、重置属性或重置消息监听器。第一次执行 Open 后便无法更改 QAManager 的属性，后续 Open 调用必须提供相同的确认模式。

异常

- “QAException 类”一节第 243 页- 如果关闭 QAManagerBase 实例时存在问题，则抛出。

另请参见

- “Open 方法”一节第 256 页

CreateBinaryMessage 方法

创建一个 QABinaryMessage 对象。

语法

Visual Basic
Public Function **CreateBinaryMessage()** As QABinaryMessage

C#
public QABinaryMessage **CreateBinaryMessage();**

返回值

一个新的 QABinaryMessage 实例。

注释

QABinaryMessage 对象用于发送包含未解释字节的消息主体的消息。

异常

- “QAException 类”一节第 243 页- 如果创建消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QABinaryMessage 接口”一节第 226 页

CreateTextMessage 方法

创建一个 QATextMessage 对象。

语法

Visual Basic
Public Function **CreateTextMessage()** As QATextMessage

C#
public QATextMessage **CreateTextMessage();**

返回值

一个新的 QATextMessage 实例。

注释

QATextMessage 对象用于发送包含字符串消息主体的消息。

异常

- “QAException 类”一节第 243 页- 如果创建消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QATextMessage 接口”一节第 325 页

GetBooleanStoreProperty 方法

获取预定义或自定义消息存储库属性的布尔值。

语法

Visual Basic

```
Public Function GetBooleanStoreProperty( _
  ByVal propName As String _
) As Boolean
```

C#

```
public bool GetBooleanStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

布尔属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetDoubleStoreProperty 方法

获取预定义或自定义消息存储库属性的双精度值。

语法

Visual Basic
```
Public Function GetDoubleStoreProperty( _
  ByVal propName As String _
) As Double
```

C#
```
public double GetDoubleStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

双精度属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetFloatStoreProperty 方法

获取预定义或自定义消息存储库属性的浮点值。

语法

Visual Basic
```
Public Function GetFloatStoreProperty( _
  ByVal propName As String _
) As Single
```

C#
```
public float  GetFloatStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

浮点属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetIntStoreProperty 方法

获取预定义或自定义消息存储库属性的整型值。

语法

Visual Basic
```
Public Function GetIntStoreProperty( _
  ByVal propName As String _
) As Integer
```

C#
```
public int GetIntStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

整型属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetLongStoreProperty 方法

获取预定义或自定义消息存储库属性的长整型值。

语法

Visual Basic

```
Public Function GetLongStoreProperty( _
  ByVal propName As String _
) As Long
```

C#

```
public long GetLongStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

长整型属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetMessage 方法

返回下一个发送到指定地址的可用 QAMessage。

语法

Visual Basic

```
Public Function GetMessage( _
  ByVal address As String _
) As QAMessage
```

C#

```
public QAMessage GetMessage(
  string  address
);
```

参数

- **address**　指定 QAnywhere 客户端用于接收消息的队列名称。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

注释

address 参数指定本地队列名称。地址可以采用 store-id\queue-name 或 queue-name 的形式。

如果没有可用消息，则此调用将无限期阻塞直到有可用消息。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

异常

- “QAException 类”一节第 243 页- 如果获取消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QAMessage 接口”一节第 304 页

GetMessageBySelector 方法

返回下一个发送到指定地址并满足给定选择程序的可用 QAMessage。

语法

Visual Basic

```
Public Function GetMessageBySelector( _
  ByVal address As String, _
  ByVal selector As String _
) As QAMessage
```

C#

```
public QAMessage GetMessageBySelector(
  string  address,
  string  selector
);
```

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。
- **selector** 选择程序。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

注释

address 参数指定本地队列名称。地址可以采用 store-id\queue-name 或 queue-name 的形式。

如果没有可用消息，则此调用将无限期阻塞直到有可用消息。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

异常

- “QAException 类”一节第 243 页- 如果获取消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QAMessage 接口”一节第 304 页

GetMessageBySelectorNoWait 方法

返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。

语法

Visual Basic

```
Public Function GetMessageBySelectorNoWait( _
  ByVal address As String, _
  ByVal selector As String _
) As QAMessage
```

C#

```
public QAMessage GetMessageBySelectorNoWait(
```

```
  string  address,
  string  selector
);
```

参数

- **address** 指定 QAnywhere 客户端用于接收消息的队列名称。
- **selector** 选择程序。

返回值

返回下一个可用消息；如果没有可用消息，则返回空值。

注释

address 参数指定本地队列名称。地址可以采用 store-id\queue-name 或 queue-name 的形式。如果没有可用消息，则此方法立即返回。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

异常

- “QAException 类”一节第 243 页- 如果获取消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QAMessage 接口”一节第 304 页

GetMessageBySelectorTimeout 方法

返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。

语法

Visual Basic

```
Public Function GetMessageBySelectorTimeout( _
  ByVal address As String, _
  ByVal selector As String, _
  ByVal timeout As Long _
) As QAMessage
```

C#

```
public QAMessage GetMessageBySelectorTimeout(
  string  address,
  string  selector,
  long timeout
);
```

参数

- **address** 指定 QAnywhere 客户端用于接收消息的队列名称。
- **selector** 选择程序。

- **timeout** 等待消息可用的时间（以毫秒为单位）。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

注释

address 参数指定本地队列名称。地址可以采用 store-id\queue-name 或 queue-name 的形式。如果没有可用消息，此方法等待指定的超时时间，然后返回。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

异常

- “QAException 类”一节第 243 页- 如果获取消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QAMessage 接口”一节第 304 页

GetMessageNoWait 方法

返回下一个发送到给定地址的可用 QAMessage。

语法

Visual Basic

```
Public Function GetMessageNoWait( _
  ByVal address As String _
) As QAMessage
```

C#

```
public QAMessage GetMessageNoWait(
  string  address
);
```

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。

返回值

返回下一个可用消息；如果没有可用消息，则返回空值。

注释

address 参数指定本地队列名称。地址可以采用 store-id\queue-name 或 queue-name 的形式。如果没有可用消息，则此方法立即返回。使用此方法同步接收消息。有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

异常

- “QAException 类”一节第 243 页- 如果获取消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QAMessage 接口”一节第 304 页

GetMessageTimeout 方法

返回下一个发送到给定地址的可用 QAMessage。

语法

Visual Basic

```
Public Function GetMessageTimeout( _
  ByVal address As String, _
  ByVal timeout As Long _
) As QAMessage
```

C#

```
public QAMessage GetMessageTimeout(
  string  address,
  long timeout
);
```

参数

- **address** 指定 QAnywhere 客户端用于接收消息的队列名称。
- **timeout** 等待消息可用的时间（以毫秒为单位）。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

注释

address 参数指定本地队列名称。地址可以采用 store-id\queue-name 或 queue-name 的形式。

如果没有可用消息，此方法等待指定的超时时间，然后返回。使用此方法同步接收消息。请参见“异步接收消息”一节第 75 页。

异常

- “QAException 类”一节第 243 页- 如果获取消息时存在问题，则抛出。

GetProperty 方法

返回 QAnywhere Manager 配置属性。

语法

Visual Basic

```
Public Sub GetProperty( _
  ByVal name As String _
)
```

C#

```
public void GetProperty(
  string  name
);
```

参数

- **name** QAnywhere Manager 配置属性名。

异常

- “QAException 类”一节第 243 页- 如果获取属性时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页

GetQueueDepth 方法

基于给定过滤器返回队列深度（包括未提交的外发消息）。

语法

Visual Basic

```
Overloads Public Function GetQueueDepth( _
  ByVal queue As String, _
  ByVal filter As QueueDepthFilter _
) As Integer
```

C#

```
public int GetQueueDepth(
  string  address,
  QueueDepthFilter filter
);
```

参数

- **queue** 队列名。
- **filter** filter 指示进来的消息、外发的消息或全部消息。

返回值

INTEGER - 未接收到的消息的数量。

注释

队列的深度是指没有被接收（如使用 QAManagerBase.GetMessage 方法）的消息的数量。

异常

- “QAException 类”一节第 243 页.

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QueueDepthFilter 枚举”一节第 331 页

GetQueueDepth 方法

基于给定过滤器，返回所有队列的深度总和。

语法

Visual Basic

```
Overloads Public Function GetQueueDepth( _
  ByVal filter As QueueDepthFilter _
) As Integer
```

C#

```
public int GetQueueDepth(
  QueueDepthFilter filter
);
```

参数

- **filter** filter 指示进来的消息、外发的消息或全部消息。

返回值

INTEGER - 未接收到的消息的数量。

注释

队列的深度是指没有被接收（如使用 QAManagerBase.GetMessage 方法）的消息的数量，其中包括未提交的外发消息。

异常

- “QAException 类”一节第 243 页.

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “QueueDepthFilter 枚举”一节第 331 页

GetSbyteStoreProperty 方法

获取预定义或自定义消息存储库属性的有符号字节值。

语法

Visual Basic

```
Public Function GetSbyteStoreProperty( _
  ByVal propName As String _
) As System.SByte
```

C#

```
public System.Sbyte GetSbyteStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

有符号字节属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetShortStoreProperty 方法

获取预定义或自定义消息存储库属性的短整型值。

语法

Visual Basic

```
Public Function GetShortStoreProperty( _
  ByVal propName As String _
) As Short
```

C#

```
public short GetShortStoreProperty(
  string  propName
);
```

参数

- **propName** 预定义或自定义属性名称。

返回值

短整型属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetStoreProperty 方法

获取表示消息存储库属性的 System.Object。

语法

Visual Basic

```
Public Function GetStoreProperty( _
  ByVal propName As String _
) As Object
```

C#

```
public object GetStoreProperty(
  string  propName
);
```

参数

- **propName** 预定义或自定义属性名称。

返回值

属性值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

异常

- “QAException 类”一节第 243 页- 若属性不存在，则抛出

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

GetStorePropertyNames 方法

获取消息存储库属性名称的枚举器。

语法

Visual Basic
Public Function **GetStorePropertyNames()** As System.Collections.IEnumerator

C#
public System.Collections.IEnumerator **GetStorePropertyNames();**

返回值

消息存储库属性名称的枚举器。

注释

有关客户端存储库属性的详细信息，请参见“客户端消息存储库属性”一节第 26 页。

GetStringStoreProperty 方法

获取预定义或自定义消息存储库属性的字符串值。

语法

Visual Basic
```
Public Function GetStringStoreProperty( _
  ByVal propName As String _
) As String
```

C#
```
public string  GetStringStoreProperty(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

返回字符串属性值；如果属性不存在，则返回空值。

注释

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

PropertyExists 方法

检查是否有用于给定属性的值。

语法

Visual Basic

```
Public Sub PropertyExists( _
  ByVal propName As String _
) As Boolean
```

C#

```
public bool PropertyExists(
  string  propName
);
```

参数

- **propName**　预定义或自定义属性名称。

返回值

如果消息存储库具有映射到属性的值，则返回 true；否则为 false。

异常

- “QAException 类”一节第 243 页- 如果检索属性值时出现问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页

PutMessage 方法

准备将发送给另一 QAnywhere 客户端的消息。

语法

Visual Basic

```
Public Sub PutMessage( _
  ByVal address As String, _
  ByVal msg As QAMessage _
)
```

C#

```
public void PutMessage(
  string  address,
  QAMessage msg
);
```

参数

- **address** 指定目标队列名称的消息的地址。
- **msg** 放入本地信息存储库以进行传输的消息。

注释

PutMessage 方法将消息和目标地址插入本地消息存储库中。消息传输的时间取决于 QAnywhere 代理的传输策略。

有关详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

地址采用 id\queue-name 的形式，其中 id 是目标消息存储库 ID，而 queue-name 则用于标识目标 QAnywhere 客户端用于监听或接收消息的队列。

有关 QAnywhere 地址的详细信息，请参见“QAnywhere 消息地址”一节第 63 页。

异常

- “QAException 类”一节第 243 页- 如果放入消息时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “PutMessageTimeToLive 方法”一节第 283 页

PutMessageTimeToLive 方法

准备将发送给另一 QAnywhere 客户端的消息。

语法

Visual Basic

```
Public Sub PutMessageTimeToLive( _
  ByVal address As String, _
```

```
  ByVal msg As QAMessage, _
  ByVal ttl As Long _
)

C#
public void PutMessageTimeToLive(
  string  address,
  QAMessage msg,
  long ttl
);
```

参数

- **address** 指定目标队列名称的消息的地址。
- **msg** 要插入的消息。
- **ttl** 延迟时间（以毫秒为单位），如果此时间前还没有发送消息，则消息将到期。值为 0 表示消息并未到期。

注释

PutMessageTimeToLive 方法将消息和目标地址插入本地消息存储库中。消息传输的时间取决于 QAnywhere 代理的传输策略。但是，如果下一条消息的传输时间超出给定生存期值，消息将到期。

有关详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

地址采用 id\queue-name 的形式，其中 id 是目标消息存储库 ID，而 queue-name 则用于标识目标 QAnywhere 客户端用于监听或接收消息的队列。

有关 QAnywhere 地址的详细信息，请参见“QAnywhere 消息地址”一节第 63 页。

异常

- “QAException 类”一节第 243 页- 如果放入消息时存在问题，则抛出。

SetBooleanStoreProperty 方法

将预定义或自定义消息存储库属性设置为布尔值。

语法

```
Visual Basic
Public Sub SetBooleanStoreProperty( _
  ByVal propName As String, _
  ByVal val As Boolean _
)

C#
public void SetBooleanStoreProperty(
  string  propName,
  bool val
);
```


参数

- **propName** 预定义或自定义属性名称。
- **val** 布尔属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetDoubleStoreProperty 方法

将预定义或自定义消息存储库属性设置为双精度值。

语法

Visual Basic

```
Public Sub SetDoubleStoreProperty( _
  ByVal propName As String, _
  ByVal val As Double _
)
```

C#

```
public void SetDoubleStoreProperty(
  string  propName,
  double val
);
```

参数

- **propName** 预定义或自定义属性名称。
- **val** 双精度属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetExceptionListener 方法

将 ExceptionListener 委派设置为在异步处理 QAnywhere 消息时，接收 QAExceptions。请参见“ExceptionListener 委派”一节第 213 页。

语法

Visual Basic

```
Public Sub SetExceptionListener( _
  ByVal address As String, _
  ByVal listener As ExceptionListener _
)
```

C#

```
public void SetExceptionListener(
  string  address,
  ExceptionListener listener
);
```

参数

- **address**　消息的地址。
- **listener**　要注册的异常监听器。

注释

ExceptionListener 委派接受 QAException 和 QAMessage 参数。您可以为给定地址设置 ExceptionListener 和 MessageListener，但是您必须与 Listener/Listener2 委派保持一致。也就是说，您无法为同一地址设置 ExceptionListener 和 MessageListener2，或 ExceptionListener2 和 MessageListener。

有关详细信息，请参见“异步接收消息”一节第 75 页。

SetExceptionListener2 方法

将 ExceptionListener2 委派设置为在异步处理 QAnywhere 消息时，接收 QAExceptions。请参见“ExceptionListener2 委派”一节第 213 页。

语法

Visual Basic

```
Public Sub SetExceptionListener2( _
  ByVal address As String, _
  ByVal listener As ExceptionListener2 _
)
```


C#

```
public void SetExceptionListener2(
  string  address,
  ExceptionListener2 listener
);
```

参数

- **address** 消息的地址。
- **listener** 要注册的异常监听器。

注释

ExceptionListener2 委派接受 QAManagerBase、QAException 和 QAMessage 参数。您可以为给定地址设置 ExceptionListener2 和 MessageListener2，但是您必须与 Listener/Listener2 委派保持一致。也就是说，您无法为同一地址设置 ExceptionListener 和 MessageListener2，或 ExceptionListener2 和 MessageListener。

有关详细信息，请参见“异步接收消息”一节第 75 页。

SetFloatStoreProperty 方法

将预定义或自定义消息存储库属性设置为浮点值。

语法

Visual Basic

```
Public Sub SetFloatStoreProperty( _
  ByVal propName As String, _
  ByVal val As Single _
)
```

C#

```
public void SetFloatStoreProperty(
  string  propName,
  float  val
);
```

参数

- **propName** 预定义或自定义属性名称。
- **val** 浮点属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetIntStoreProperty 方法

将预定义或自定义消息存储库属性设置为整型值。

语法

Visual Basic

```
Public Sub SetIntStoreProperty( _
  ByVal propName As String, _
  ByVal val As Integer _
)
```

C#

```
public void SetIntStoreProperty(
  string  propName,
  int val
);
```

参数

- **propName**　预定义或自定义属性名称。
- **val**　整型属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetLongStoreProperty 方法

将预定义或自定义消息存储库属性设置为长整型值。

语法

Visual Basic

```
Public Sub SetLongStoreProperty( _
  ByVal propName As String, _
```



```
  ByVal val As Long _
)
```

C#

```
public void SetLongStoreProperty(
  string  propName,
  long val
);
```

参数

- **propName** 预定义或自定义属性名称。
- **val** 长整型属性值

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetMessageListener 方法

将 MessageListener 委派设置为异步接收 QAnywhere 消息。请参见“MessageListener 委派”一节第 214 页。

语法

Visual Basic

```
Public Sub SetMessageListener( _
  ByVal address As String, _
  ByVal listener As MessageListener _
)
```

C#

```
public void SetMessageListener(
  string  address,
  MessageListener listener
);
```

参数

- **address** 消息的地址。
- **listener** 要注册的监听器。

注释

使用此方法异步接收消息。

MessageListener 委派接受一个单独的 QAMessage 参数。

SetMessageListener 地址参数指定用于接收消息的本地队列名称。只能为指定队列指派一个监听器委派。可以为给定地址设置 ExceptionListener 和 MessageListener，但是必须与 Listener/Listener2 委派保持一致。也就是说，您无法为同一地址设置 ExceptionListener 和 MessageListener2，或 ExceptionListener2 和 MessageListener。

如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 system 作为队列名称。

有关详细信息，请参见“异步接收消息”一节第 75 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageListener 委派”一节第 214 页

SetMessageListener2 方法

将 MessageListener2 委派设置为异步接收 QAnywhere 消息。请参见“MessageListener2 委派”一节第 214 页。

语法

Visual Basic

```
Public Sub SetMessageListener2( _
  ByVal address As String, _
  ByVal listener As MessageListener2 _
)
```

C#

```
public void SetMessageListener2(
  string  address,
  MessageListener2 listener
);
```

参数

- **address**　消息的地址。
- **listener**　要注册的监听器。

注释

使用此方法异步接收消息。

MessageListener2 委派接受 QAManagerBase 和 QAMessage 参数。

SetMessageListener2 地址参数指定用于接收消息的本地队列名称。只能为给定队列指派一个监听器委派。可以为给定地址设置 ExceptionListener2 和 MessageListener2，但是必须与 Listener/Listener2

委派保持一致。也就是说，您无法为同一地址设置 ExceptionListener 和 MessageListener2，或 ExceptionListener2 和 MessageListener。

如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 system 作为队列名称。

有关详细信息，请参见“异步接收消息”一节第 75 页。

SetMessageListenerBySelector 方法

将 MessageListener 委派设置为使用消息选择程序异步接收 QAnywhere 消息。请参见“MessageListener 委派”一节第 214 页。

语法

Visual Basic

```
Public Sub SetMessageListenerBySelector( _
  ByVal address As String, _
  ByVal selector As String, _
  ByVal listener As MessageListener _
)
```

C#

```
public void SetMessageListenerBySelector(
  string  address,
  string  selector,
  MessageListener listener
);
```

参数

- **address** 消息的地址。
- **listener** 要注册的监听器。
- **selector** 用于过滤要接收的消息的选择程序。

注释

使用此方法异步接收消息。

MessageListener 委派接受一个单独的 QAMessage 参数。

SetMessageListener 地址参数指定用于接收消息的本地队列名称。只能为给定队列指派一个监听器委派。selector 参数指定用于过滤在给定地址上接收的消息的选择程序。可以为给定地址设置 ExceptionListener 和 MessageListener，但是必须与 Listener/Listener2 委派保持一致。也就是说，您无法为同一地址设置 ExceptionListener 和 MessageListener2，或 ExceptionListener2 和 MessageListener。

如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 system 作为队列名称。

有关详细信息，请参见“异步接收消息”一节第 75 页和“系统队列”一节第 63 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageListener 委派”一节第 214 页

SetMessageListenerBySelector2 方法

将 MessageListener2 委派设置为使用消息选择程序异步接收 QAnywhere 消息。请参见“MessageListener2 委派”一节第 214 页。

语法

Visual Basic

```
Public Sub SetMessageListenerBySelector2( _
  ByVal address As String, _
  ByVal selector As String, _
  ByVal listener As MessageListener2 _
)
```

C#

```
public void SetMessageListenerBySelector2(
  string  address,
  string  selector,
  MessageListener2 listener
);
```

参数

- **address** 消息的地址。
- **listener** 要注册的监听器。
- **selector** 用于过滤要接收的消息的选择程序。

注释

使用此方法异步接收消息。

MessageListener2 委派接受一个单独的 QAMessage 参数。

SetMessageListener2 地址参数指定用于接收消息的本地队列名称。只能为给定队列指派一个监听器委派。selector 参数指定用于过滤在给定地址上接收的消息的选择程序。可以为给定地址设置 ExceptionListener2 和 MessageListener2，但是必须与 Listener/Listener2 委派保持一致。也就是说，您无法为同一地址设置 ExceptionListener 和 MessageListener2，或 ExceptionListener2 和 MessageListener。

如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 system 作为队列名称。

有关详细信息，请参见“异步接收消息”一节第 75 页和“系统队列”一节第 63 页。

SetProperty 方法

用于通过编程方式设置 QAnywhere Manager 配置属性。

语法

Visual Basic

```
Public Sub SetProperty( _
  ByVal name As String, _
  ByVal val As String _
)
```

C#

```
public void SetProperty(
  string  name,
  string  val
);
```

参数

- **name** QAnywhere Manager 配置属性名。
- **val** QAnywhere Manager 配置属性值

注释

可以使用此方法通过指定属性名称和值来替换缺省 QAnywhere Manager 配置属性。有关属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

还可以使用属性文件和 QAManagerFactory.CreateQAManager 方法设置 QAnywhere Manager 配置属性。

有关详细信息，请参见“在文件中设置 QAnywhere Manager 配置属性”一节第 91 页。**注意：**调用 QAManager.Open 或 QATransactionalManager.Open() 前，必须设置必需的属性。

异常

- “QAException 类”一节第 243 页- 如果设置属性时出现问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “Open 方法”一节第 256 页
- “Open 方法”一节第 330 页

SetSbyteStoreProperty 方法

将预定义或自定义消息存储库属性设置为 sbyte 值。

语法

Visual Basic

```
Public Sub SetSbyteStoreProperty( _
```

```
  ByVal propName As String, _
  ByVal val As System.SByte _
)

C#
public void SetSbyteStoreProperty(
  string  propName,
  System.Sbyte val
);
```

参数

- **propName** 预定义或自定义属性名称。
- **val** sbyte 属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetShortStoreProperty 方法

将预定义或自定义消息存储库属性设置为短整型值。

语法

```
Visual Basic
Public Sub SetShortStoreProperty( _
  ByVal propName As String, _
  ByVal val As Short _
)

C#
public void SetShortStoreProperty(
  string  propName,
  short val
);
```

参数

- **propName** 预定义或自定义属性名称。
- **val** 短整型属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetStoreProperty 方法

将预定义或自定义消息存储库属性设置为 System.Object 值。

语法

Visual Basic

```
Public Sub SetStoreProperty( _
  ByVal propName As String, _
  ByVal val As Object _
)
```

C#

```
public void SetStoreProperty(
  string  propName,
  object val
);
```

参数

- **propName**　预定义或自定义属性名称。
- **val**　属性值。

注释

属性类型必须为可接受的原始类型之一或字符串型。可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

SetStringStoreProperty 方法

将预定义或自定义消息存储库属性设置为字符串值。

语法

Visual Basic
```
Public Sub SetStringStoreProperty( _
  ByVal propName As String, _
  ByVal val As String _
)
```

C#
```
public void SetStringStoreProperty(
  string  propName,
  string  val
);
```

参数

- **propName** 预定义或自定义属性名称。
- **val** 字符串属性值。

注释

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 223 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “MessageStoreProperties 类”一节第 223 页

Start 方法

启动 QAManagerBase 来接收消息监听器中进来的消息。

语法

Visual Basic
```
Public Sub Start()
```

C#
```
public void Start();
```

注释

如果未设置消息监听器（即用 GetMessage 方法接收消息），则不需要启动 QAManagerBase。不建议使用 GetMessage 方法和消息监听器来接收消息。应该使用异步（消息监听器）或同步

(GetMessage) 模型二者中的一种来接收消息。除了第一个 Start() 调用外，其它没有使用 QAManagerBase.Stop() 调用进行干预的 Start() 调用都将被忽略。

异常

- “QAException 类”一节第 243 页- 如果启动 QAManagerBase 实例时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “Stop 方法”一节第 297 页

Stop 方法

使 QAManagerBase 停止接收进来的消息。

语法

Visual Basic
Public Sub **Stop()**

C#
public void **Stop();**

注释

消息并没有丢失。再次启动管理器后才接收这些消息。除了第一个 Stop() 调用外，其它没有使用 QAManagerBase.Start() 调用进行干预的 Stop() 调用都将被忽略。

异常

- “QAException 类”一节第 243 页- 如果停止 QAManagerBase 实例时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “Start 方法”一节第 296 页

TriggerSendReceive 方法

在上载任何发送给其它客户端的消息和下载任何发送给本地客户端的消息时，皆与 QAnywhere 消息服务器保持同步。

语法

Visual Basic
Public Sub **TriggerSendReceive()**

C#
public void **TriggerSendReceive();**

注释

QAManagerBase 的 TriggerSendReceive 会使消息在 QAnywhere 代理与中央消息传递服务器之间立即得到同步。手工调用 TriggerSendReceive 将导致消息立即传输，与 QAnywhere 代理传输策略无关。

QAnywhere 代理传输策略决定消息传输的方式。例如，当客户端接收推式通知，或者当您调用 QAManagerBase.PutMessage 方法来发送消息时，消息传输将定期自动进行。

有关详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

异常

- “QAException 类”一节第 243 页- 如果触发发送/接收时存在问题，则抛出。

另请参见

- “QAManagerBase 接口”一节第 257 页
- “QAManagerBase 成员”一节第 258 页
- “PutMessage 方法”一节第 283 页

QAManagerFactory 类

此类充当用于创建 QATransactionalManager 和 QAManager 对象的工厂类。

语法

Visual Basic
MustInherit Public Class **QAManagerFactory**

C#
public abstract class **QAManagerFactory**

注释

只能有一个 QAManagerFactory 实例。

QAManagerFactory 成员

公共静态属性（共享）

成员名称	说明
“Instance 属性”一节第 299 页	单个 QAManagerFactory 实例。

公共构造函数

成员名称	说明
“QAManagerFactory 构造函数”一节第 299 页	

公共方法

成员名称	说明
“CreateQAManager 方法”一节第 300 页	返回带有指定属性的新 QAManager 实例。
“CreateQATransactionalManager 方法”一节第 302 页	返回带有指定属性的新 QATransactionalManager 实例。

QAManagerFactory 构造函数

语法

Visual Basic
Public Sub **New()**

C#
public **QAManagerFactory();**

Instance 属性

单个 QAManagerFactory 实例。

语法

Visual Basic
Public Shared Readonly Property **Instance** As QAManagerFactory

C#
public const QAManagerFactory **Instance** {get;}

异常

- “QAException 类”一节第 243 页- 如果创建管理器工厂时存在问题，则抛出。

CreateQAManager 方法

创建 QAManager 实例。

语法

Visual Basic
Public Function **CreateQAManager()** As QAManager

C#
public QAManager **CreateQAManager();**

返回值

一个新的 QAManager 实例。

注释

使用缺省属性创建 QAManager。创建此实例后，可以使用 QAManagerBase.SetProperty 通过编程方式设置 QAnywhere Manager 配置属性。请参见“SetProperty 方法”一节第 293 页。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

异常

- “QAException 类”一节第 243 页.

另请参见

- “在文件中设置 QAnywhere Manager 配置属性”一节第 91 页
- “QAManagerFactory 类”一节第 298 页
- “QAManagerFactory 成员”一节第 298 页
- “QAManager 接口”一节第 253 页

CreateQAManager 方法

使用指定属性创建 QAManager 实例。

语法

Visual Basic
```
Public Function CreateQAManager( _
  ByVal iniFile As String _
) As QAManager
```

C#
```
public QAManager CreateQAManager(
  string  iniFile
);
```

参数

- **iniFile**　指定用于配置 QAManager 实例的属性文件。

返回值

一个新的 QAManager 实例。

注释

如果 *iniFile* 为空值，则使用缺省属性创建 QAManager。创建此实例后，可以使用 QAManagerBase.SetProperty 通过编程方式设置 QAnywhere Manager 配置属性。请参见“SetProperty 方法”一节第 293 页。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

异常

- “QAException 类”一节第 243 页.

另请参见

- “在文件中设置 QAnywhere Manager 配置属性”一节第 91 页
- “QAManagerFactory 类”一节第 298 页
- “QAManagerFactory 成员”一节第 298 页
- “QAManager 接口”一节第 253 页

CreateQAManager 方法

使用指定属性创建 QAManager 实例。

语法

Visual Basic

```
Public Function CreateQAManager( _
  ByVal props As Hashtable _
) As QAManager
```

C#

```
public QAManager CreateQAManager(
  Hashtable  props
);
```

参数

- **props**　指定用于配置 QAManager 实例的属性 hashtable。

返回值

一个新的 QAManager 实例。

注释

如果 *props* 为空值，则使用缺省属性创建 QAManager。创建此实例后，可以使用 QAManagerBase.SetProperty 通过编程方式设置 QAnywhere Manager 配置属性。请参见“SetProperty 方法”一节第 293 页。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

异常

- “QAException 类”一节第 243 页.

另请参见

- “QAManagerFactory 类”一节第 298 页
- “QAManagerFactory 成员”一节第 298 页
- “QAManager 接口”一节第 253 页

CreateQATransactionalManager 方法

创建 QATransactionalManager 实例。

语法

Visual Basic
Public Function **CreateQATransactionalManager()** As QATransactionalManager

C#
public QATransactionalManager **CreateQATransactionalManager();**

返回值

一个新的 QATransactionalManager 实例。

注释

使用缺省属性创建 QATransactionalManager。创建此实例后，可以使用 QAManagerBase.SetProperty 通过编程方式设置 QAnywhere Manager 配置属性。请参见“SetProperty 方法”一节第 293 页。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

异常

- “QAException 类”一节第 243 页.

另请参见

- “在文件中设置 QAnywhere Manager 配置属性”一节第 91 页
- “QAManagerFactory 类”一节第 298 页
- “QAManagerFactory 成员”一节第 298 页
- “QATransactionalManager 接口”一节第 328 页

CreateQATransactionalManager 方法

使用指定属性创建 QATransactionalManager 实例。

语法

Visual Basic

```
Public Function CreateQATransactionalManager( _
  ByVal iniFile As String _
) As QATransactionalManager
```

C#

```
public QATransactionalManager CreateQATransactionalManager(
  string  iniFile
);
```

参数

- **iniFile**　指定用于配置 QATransactionalManager 实例的属性文件。

返回值

一个新配置的 QATransactionalManager 实例。

注释

如果 *iniFile* 为空值，则使用缺省属性创建 QATransactionalManager。创建此实例后，可以使用 QAManagerBase.SetProperty 通过编程方式设置 QAnywhere Manager 配置属性。请参见“SetProperty 方法”一节第 293 页。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

异常

- “QAException 类”一节第 243 页.

另请参见

- “在文件中设置 QAnywhere Manager 配置属性”一节第 91 页
- “QAManagerFactory 类”一节第 298 页
- “QAManagerFactory 成员”一节第 298 页
- “QATransactionalManager 接口”一节第 328 页

CreateQATransactionalManager 方法

使用指定属性创建 QATransactionalManager 实例。

语法

Visual Basic

```
Public Function CreateQATransactionalManager( _
  ByVal props As Hashtable _
) As QATransactionalManager
```

C#

```
public QATransactionalManager CreateQATransactionalManager(
  Hashtable  props
);
```

参数

- **props** 指定用于配置 QATransactionalManager 实例的属性 hashtable。

返回值

一个新配置的 QATransactionalManager 实例。

注释

如果 *props* 为空值，则使用缺省属性创建 QATransactionalManager。创建此实例后，可以使用 QAManagerBase.SetProperty 通过编程方式设置 QAnywhere Manager 配置属性。请参见“SetProperty 方法”一节第 293 页。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

异常

- “QAException 类”一节第 243 页.

另请参见

- “QAManagerFactory 类”一节第 298 页
- “QAManagerFactory 成员”一节第 298 页
- “QATransactionalManager 接口”一节第 328 页

QAMessage 接口

提供用于设置消息属性和标头字段的接口。

语法

Visual Basic
Public Interface **QAMessage**

C#
public interface **QAMessage**

注释

派生类 QABinaryMessage 和 QATextMessage 提供读写消息主体的专用方法。可以使用 QAMessage 方法设置预定义或自定义消息属性。

有关预定义属性名称的列表，请参见“MessageProperties 类”一节第 215 页。

有关设置消息属性和标头字段的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 成员”一节第 305 页
- “QABinaryMessage 接口”一节第 226 页
- “QATextMessage 接口”一节第 325 页

QAMessage 成员

公共属性

成员名称	说明
“Address 属性”一节 第 307 页	QAMessage 实例的目标地址。
“Expiration 属性”一节 第 307 页	获取消息的有效期值。
“InReplyToID 属性”一节 第 308 页	此消息所回复的消息的消息 ID。
“MessageID 属性”一节 第 308 页	消息的全局唯一消息 ID。
“Priority 属性”一节第 309 页	消息的优先级（从 0 到 9）。
“Redelivered 属性”一节 第 309 页	指示消息是否以前已接收但尚未确认。
“ReplyToAddress 属性”一节 第 310 页	此消息的回复地址。
“Timestamp 属性”一节 第 310 页	消息时间戳。

公共方法

成员名称	说明
“ClearBody 方法”一节 第 310 页	清除消息的主体。
“ClearProperties 方法”一节 第 310 页	清除消息的所有属性。
“GetBooleanProperty 方法”一节第 311 页	获取布尔型消息属性。
“GetByteProperty 方法”一节 第 311 页	获取字节消息属性。
“GetDoubleProperty 方法”一节第 312 页	获取双精度型消息属性。

成员名称	说明
“GetFloatProperty 方法”一节 第 313 页	获取浮点型消息属性。
“GetIntProperty 方法”一节 第 314 页	获取整型消息属性。
“GetLongProperty 方法”一节 第 314 页	获取长整型消息属性。
“GetProperty 方法”一节 第 315 页	获取消息属性。
“GetPropertyNames 方法”一节 第 316 页	获取消息属性名称的枚举器。
“GetPropertyType 方法”一节 第 316 页	返回给定属性的属性类型。
“GetSbyteProperty 方法”一节 第 316 页	获取有符号字节型消息属性。
“GetShortProperty 方法”一节 第 317 页	获取短整型消息属性。
“GetStringProperty 方法”一节 第 318 页	获取字符串型消息属性。
“PropertyExists 方法”一节 第 319 页	指示是否已为此消息设置了给定属性。
“SetBooleanProperty 方法”一节第 319 页	设置布尔型属性。
“SetByteProperty 方法”一节 第 320 页	设置字节型属性。
“SetDoubleProperty 方法”一节 第 320 页	设置双精度型属性。
“SetFloatProperty 方法”一节 第 321 页	设置浮点型属性。
“SetIntProperty 方法”一节 第 322 页	设置整型属性。

成员名称	说明
“SetLongProperty 方法”一节 第 322 页	设置长整型属性。
“SetProperty 方法”一节 第 323 页	设置属性。
“SetSbyteProperty 方法”一节 第 323 页	设置有符号字节型属性。
“SetShortProperty 方法”一节 第 324 页	设置短整型属性。
“SetStringProperty 方法”一节 第 325 页	设置字符串型属性。

Address 属性

QAMessage 实例的目标地址。

语法

Visual Basic
Public Readonly Property **Address** As String

C#
public string **Address** {get;}

注释

发送消息时，此字段被忽略。发送操作完成后，此字段会保存在 QAManagerBase.PutMessage 中指定的目标地址。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “PutMessage 方法”一节第 283 页

Expiration 属性

获取消息的有效期值。

语法

Visual Basic
Public Readonly Property **Expiration** As Date

C#
public DateTime **Expiration** {get;}

注释

发送消息时，Expiration 标头字段保留未指派状态。发送方法完成后，它保留消息的到期时间。

它是一个只读属性，因为消息的到期时间是通过将 QAManagerBase::PutMessageTimeToLive 的生存期参数值与当前时间相加而设置的。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

InReplyToID 属性

此消息所回复的消息的消息 ID。

语法

Visual Basic
Public Property **InReplyToID** As String

C#
public string **InReplyToID** {get;set;}

注释

可以为空值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

MessageID 属性

消息的全局唯一消息 ID。

语法

Visual Basic
Public Readonly Property **MessageID** As String

C#
public string **MessageID** {get;}

注释

此属性在放入消息前一直为空值。

使用 QAManagerBase.PutMessage 发送消息时，MessageID 为空值并且可以被忽略。发送方法返回时，它包含一个指派的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “PutMessage 方法”一节第 283 页

Priority 属性

消息的优先级（从 0 到 9）。

语法

Visual Basic
Public Property **Priority** As Integer

C#
public int **Priority** {get;set;}

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

Redelivered 属性

指示消息是否以前已接收但尚未确认。

语法

Visual Basic
Public Readonly Property **Redelivered** As Boolean

C#
public bool **Redelivered** {get;}

注释

如果接收方 QAManager 检测到以前曾收到过正在接收的消息，则设置 Redelivered。

例如，应用程序使用以 AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 打开的 QAManager 接收消息，并且没有确认此消息就关闭。当应用程序再次启动并接收同一消息时，Redelivered 标头变为 true。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “QAManager 接口”一节第 253 页
- “AcknowledgementMode 枚举”一节第 212 页

ReplyToAddress 属性

此消息的回复地址。

语法

Visual Basic
Public Property **ReplyToAddress** As String

C#
public string **ReplyToAddress** {get;set;}

注释

可以为空值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

Timestamp 属性

消息时间戳。

语法

Visual Basic
Public Readonly Property **Timestamp** As Date

C#
public DateTime **Timestamp** {get;}

注释

此 Timestamp 标头字段包含消息的创建时间。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

ClearBody 方法

清除消息的主体。

语法

Visual Basic
Public Sub **ClearBody()**

C#
public void **ClearBody();**

ClearProperties 方法

清除消息的所有属性。

语法

Visual Basic
Public Sub **ClearProperties()**

C#
public void **ClearProperties();**

GetBooleanProperty 方法

获取布尔型消息属性。

语法

Visual Basic
Public Function **GetBooleanProperty(_**
  ByVal *propName* As String _
) As Boolean

C#
public bool **GetBooleanProperty(**
  string *propName*
);

参数

- **propName** 属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetByteProperty 方法

获取字节消息属性。

语法

Visual Basic
```
Public Function GetByteProperty( _
  ByVal propName As String _
) As Byte
```

C#
```
public byte GetByteProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetDoubleProperty 方法

获取双精度型消息属性。

语法

Visual Basic
```
Public Function GetDoubleProperty( _
  ByVal propName As String _
) As Double
```

C#
```
public double GetDoubleProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetFloatProperty 方法

获取浮点型消息属性。

语法

Visual Basic

```
Public Function GetFloatProperty( _
  ByVal propName As String _
) As Single
```

C#

```
public float  GetFloatProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetIntProperty 方法

获取整型消息属性。

语法

Visual Basic
```
Public Function GetIntProperty( _
  ByVal propName As String _
) As Integer
```

C#
```
public int GetIntProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetLongProperty 方法

获取长整型消息属性。

语法

Visual Basic
```
Public Function GetLongProperty( _
```

```
  ByVal propName As String _
) As Long

C#
public long GetLongProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetProperty 方法

获取消息属性。

语法

```
Visual Basic
Public Function GetProperty( _
  ByVal propName As String _
) As Object

C#
public object GetProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

属性值。

注释

属性必须为可接受的原始类型之一、字符串或 DateTime。

异常

- “QAException 类”一节第 243 页- 若属性不存在，则抛出。

GetPropertyNames 方法

获取消息属性名称的枚举器。

语法

Visual Basic
Public Function **GetPropertyNames()** As System.Collections.IEnumerator

C#
public System.Collections.IEnumerator **GetPropertyNames();**

返回值

消息属性名称的枚举器。

GetPropertyType 方法

返回给定属性的属性类型。

语法

Visual Basic
Public Function **GetPropertyType(_**
  ByVal *propName* As String _
) As PropertyType

C#
public PropertyType **GetPropertyType(**
  string *propName*
);

参数

- **propName** 属性的名称。

返回值

属性类型。

GetSbyteProperty 方法

获取有符号字节型消息属性。

语法

Visual Basic
```
Public Function GetSbyteProperty( _
  ByVal propName As String _
) As System.SByte
```

C#
```
public System.Sbyte GetSbyteProperty(
  string  propName
);
```

参数

- **propName** 属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetShortProperty 方法

获取短整型消息属性。

语法

Visual Basic
```
Public Function GetShortProperty( _
  ByVal propName As String _
) As Short
```

C#
```
public short GetShortProperty(
  string  propName
);
```

参数

- **propName** 属性名称。

返回值

属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

异常

- “QAException 类”一节第 243 页- 如果获取属性值时出现转换错误或如果此属性不存在，则抛出。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

GetStringProperty 方法

获取字符串型消息属性。

语法

Visual Basic

```
Public Function GetStringProperty( _
  ByVal propName As String _
) As String
```

C#

```
public string  GetStringProperty(
  string  propName
);
```

参数

- **propName**　属性名称。

返回值

返回属性值；如果属性不存在，则返回空值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

PropertyExists 方法

指示是否已为此消息设置了给定属性。

语法

Visual Basic

```
Public Function PropertyExists( _
  ByVal propName As String _
) As Boolean
```

C#

```
public bool PropertyExists(
  string  propName
);
```

参数

- **propName** 属性名称。

返回值

当属性存在时返回 true。

SetBooleanProperty 方法

设置布尔型属性。

语法

Visual Basic

```
Public Sub SetBooleanProperty( _
  ByVal propName As String, _
  ByVal val As Boolean _
)
```

C#

```
public void SetBooleanProperty(
  string  propName,
  bool val
);
```

参数

- **propName** 属性名称。
- **val** 属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetByteProperty 方法

设置字节型属性。

语法

Visual Basic
```
Public Sub SetByteProperty( _
   ByVal propName As String, _
   ByVal val As Byte _
)
```

C#
```
public void SetByteProperty(
   string  propName,
   byte val
);
```

参数

- **propName**　属性名称。
- **val**　属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetDoubleProperty 方法

设置双精度型属性。

语法

Visual Basic
```
Public Sub SetDoubleProperty( _
   ByVal propName As String, _
   ByVal val As Double _
)
```


C#

```
public void SetDoubleProperty(
  string  propName,
  double val
);
```

参数

- **propName** 属性名称。
- **val** 属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetFloatProperty 方法

设置浮点型属性。

语法

Visual Basic

```
Public Sub SetFloatProperty( _
  ByVal propName As String, _
  ByVal val As Single _
)
```

C#

```
public void SetFloatProperty(
  string  propName,
  float  val
);
```

参数

- **propName** 属性名称。
- **val** 属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetIntProperty 方法

设置整型属性。

语法

Visual Basic
```
Public Sub SetIntProperty( _
  ByVal propName As String, _
  ByVal val As Integer _
)
```

C#
```
public void SetIntProperty(
  string  propName,
  int val
);
```

参数

- **propName**　属性名称。
- **val**　属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetLongProperty 方法

设置长整型属性。

语法

Visual Basic
```
Public Sub SetLongProperty( _
  ByVal propName As String, _
  ByVal val As Long _
)
```

C#
```
public void SetLongProperty(
  string  propName,
  long val
);
```

参数

- **propName**　属性名称。

- **val** 属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetProperty 方法

设置属性。

语法

Visual Basic

```
Public Sub SetProperty( _
  ByVal propName As String, _
  ByVal val As Object _
)
```

C#

```
public void SetProperty(
  string  propName,
  object val
);
```

参数

- **propName** 属性名称。
- **val** 属性值。

注释

属性类型必须为可接受的原始类型之一或字符串型。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetSbyteProperty 方法

设置有符号字节型属性。

语法

Visual Basic
```
Public Sub SetSbyteProperty( _
  ByVal propName As String, _
  ByVal val As System.SByte _
)
```

C#
```
public void SetSbyteProperty(
  string  propName,
  System.Sbyte val
);
```

参数

- **propName** 属性名称。
- **val** 属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetShortProperty 方法

设置短整型属性。

语法

Visual Basic
```
Public Sub SetShortProperty( _
  ByVal propName As String, _
  ByVal val As Short _
)
```

C#
```
public void SetShortProperty(
  string  propName,
  short val
);
```

参数

- **propName** 属性名称。
- **val** 属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

SetStringProperty 方法

设置字符串型属性。

语法

Visual Basic

```
Public Sub SetStringProperty( _
  ByVal propName As String, _
  ByVal val As String _
)
```

C#

```
public void SetStringProperty(
  string  propName,
  string  val
);
```

参数

- **propName**　属性名称。
- **val**　属性值。

注释

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

- “QAMessage 接口”一节第 304 页
- “QAMessage 成员”一节第 305 页
- “MessageProperties 类”一节第 215 页

QATextMessage 接口

QATextMessage 继承自 QAMessage 类，并添加了文本消息主体。QATextMessage 提供读写文本消息主体的方法。

语法

Visual Basic

```
Public Interface QATextMessage
```

C#
public interface **QATextMessage**

注释

首次创建消息时，消息的主体处于只写模式。消息发送后，发送消息的客户端可保留和修改该消息，而不会影响已发送的消息。可以多次发送同一消息对象。

接收到消息时，提供程序已调用 QATextMessage.Reset()，因此消息主体处于只读模式并且从消息主体的开头开始读取值。

另请参见

- “QATextMessage 成员”一节第 326 页
- “QABinaryMessage 接口”一节第 226 页
- “QAMessage 接口”一节第 304 页

QATextMessage 成员

公共属性

成员名称	说明
“Text 属性”一节第 326 页	消息文本。
“TextLength 属性”一节第 327 页	消息长度（以字符数表示）。

公共方法

成员名称	说明
“ReadText 方法”一节第 327 页	将未读文本读取到给定缓冲区中。
“Reset 方法”一节第 328 页	将消息的文本位置重置为开头。
“WriteText 方法”一节第 328 页	将文本附加到消息文本中。

Text 属性

消息文本。

语法

Visual Basic
Public Property **Text** As String

C#
public string **Text** {get;set;}

注释

如果消息大小超出 QAManager.MAX_IN_MEMORY_MESSAGE_SIZE 指定的最大值，则此属性为空值。在这种情况下，使用 QATextMessage.ReadText 方法读取文本。

有关 QAManager 属性的详细信息，请参见“QAnywhere Manager 配置属性”一节第 90 页。

另请参见

- “QATextMessage 接口”一节第 325 页
- “QATextMessage 成员”一节第 326 页
- “ReadText 方法”一节第 327 页

TextLength 属性

消息长度（以字符数表示）。

语法

Visual Basic
Public Readonly Property **TextLength** As Long

C#
public long **TextLength** {get;}

ReadText 方法

将未读文本读取到给定缓冲区中。

语法

Visual Basic
Public Function **ReadText(_**
  ByVal *buf* As System.Text.StringBuilder _
) As Integer

C#
public int **ReadText(**
  System.Text.string Builder *buf*
);

参数

- **buf**　任何读取文本的目标缓冲区。

返回值

返回读取的字符数；如果没有要读取的字符，则返回 -1。

注释

任何其它未读文本必须通过随后调用此方法来读取。从任何未读文本的开头读取文本。

Reset 方法

将消息的文本位置重置为开头。

语法

Visual Basic
Public Sub **Reset()**

C#
public void **Reset();**

WriteText 方法

将文本附加到消息文本中。

语法

Visual Basic
Public Sub **WriteText(_**
  ByVal *val* As String **_**
)

C#
public void **WriteText(**
  string *val*
);

参数

- **val** 要附加的文本。

QATransactionalManager 接口

QATransactionalManager 类从 QAManagerBase 派生并且管理事务性 QAnywhere 消息传递操作。

语法

Visual Basic
Public Interface **QATransactionalManager**

C#
public interface **QATransactionalManager**

注释

有关派生行为的详细说明，请参见“QAManagerBase 接口”一节第 257 页。

QATransactionalManager 只能用于事务性确认。使用 QATransactionalManager.Commit() 方法提交全部 QAManagerBase.PutMessage 和 QAManagerBase.GetMessage 调用。

有关详细信息，请参见“实现事务性消息传递”一节第 67 页。

另请参见

- “QATransactionalManager 成员”一节第 329 页
- “QATransactionalManager 接口”一节第 328 页

QATransactionalManager 成员

公共方法

成员名称	说明
“Commit 方法”一节第 329 页	提交当前事务并开始新的事务。
“Open 方法”一节第 330 页	打开一个 QATransactionalManager 实例。
“Rollback 方法”一节第 330 页	回退当前事务并开始新的事务。

Commit 方法

提交当前事务并开始新的事务。

语法

Visual Basic
Public Sub **Commit()**

C#
public void **Commit();**

注释

此方法会提交全部 QAManagerBase.PutMessage 和 QAManagerBase.GetMessage 调用。*注意：*第一个事务以调用 QATransactionalManager.Open() 开始。

异常

- “QAException 类”一节第 243 页- 如果提交时存在问题，则抛出。

另请参见

- “QATransactionalManager 接口”一节第 328 页
- “QATransactionalManager 成员”一节第 329 页
- “QATransactionalManager 接口”一节第 328 页

Open 方法

打开一个 QATransactionalManager 实例。

语法

Visual Basic
Public Sub **Open()**

C#
public void **Open();**

注释

Open 方法必须是创建管理器后调用的第一个方法。

异常

- “QAException 类”一节第 243 页- 如果打开管理器时存在问题，则抛出

另请参见

- “QATransactionalManager 接口”一节第 328 页
- “QATransactionalManager 成员”一节第 329 页
- “QATransactionalManager 接口”一节第 328 页

Rollback 方法

回退当前事务并开始新的事务。

语法

Visual Basic
Public Sub **Rollback()**

C#
public void **Rollback();**

注释

此方法会回退全部未提交的 QAManagerBase.PutMessage 和 QAManagerBase.GetMessage 调用。

异常

- “QAException 类”一节第 243 页- 如果回退时存在问题，则抛出

另请参见

- “QATransactionalManager 接口”一节第 328 页
- “QATransactionalManager 成员”一节第 329 页
- “QATransactionalManager 接口”一节第 328 页

QueueDepthFilter 枚举

为 QAManagerBase.GetQueueDepth(QueueDepthFilter) 和 QAManagerBase.GetQueueDepth(string,QueueDepthFilter) 提供队列深度过滤器值。

语法

Visual Basic
Public Enum **QueueDepthFilter**

C#
public enum **QueueDepthFilter**

成员名称

成员名称	说明
ALL	对进来的消息和外发的消息进行计数。
INCOMING	只对进来的消息进行计数。
LOCAL	如果指定了队列，则对发送到该队列的未接收本地消息进行计数；否则，对消息存储库中未接收的本地消息的总数进行计数，但不包括系统消息。
OUTGOING	只对外发的消息进行计数。

另请参见

- “GetQueueDepth 方法”一节第 278 页
- “GetQueueDepth 方法”一节第 277 页

StatusCodes 枚举

此枚举为消息状态定义了一组代码。

语法

Visual Basic
Public Enum **StatusCodes**

C#
public enum **StatusCodes**

成员名称

成员名称	说明
CANCELLED	消息已取消。

成员名称	说明
EXPIRED	此消息因未在到期时间前收到而到期。
FINAL	此常量用于确定消息是否已达到最终状态。当且仅当消息状态大于此常量时，消息达到最终状态。
LOCAL	此消息被发送到本地消息存储库，没有传输到服务器。
PENDING	消息已发送但未被接收。
RECEIVED	此消息已由接收方接收并确认。
RECEIVING	消息正处于接收过程中，或者已接收但未确认。
TRANSMITTED	消息已经传输到服务器。
TRANSMITTING	此消息正处于传输到服务器的过程中。
UNRECEIVABLE	此消息已标记为无法接收。消息格式错误或尝试发送时失败次数过多。
UNTRANSMITTED	消息未传输到服务器。

用于 Web 服务的 QAnywhere .NET (.NET 2.0)

命名空间

iAnywhere.QAnywhere.WS

WSBase 类

它是由移动 Web 服务编译器生成的主要 Web 服务代理类的基类。

语法

Visual Basic
Public Class **WSBase**

C#
public class **WSBase**

WSBase 成员

公共构造函数

成员名称	说明
“WSBase 构造函数”一节 第 334 页	使用配置属性文件指定的属性构造一个 WSBase 实例。
“WSBase 构造函数”一节 第 335 页	使用默认属性构造一个 WSBase 实例。

公共方法

成员名称	说明
“ClearRequestProperties 方法”一节第 335 页	清除为此 WSBase 设置的全部请求属性。
“GetResult 方法”一节 第 335 页	获取代表 Web 服务请求结果的 WSResult 对象。
“GetServiceID 方法”一节 第 336 页	获取此 WSBase 实例的服务 ID。
“SetListener 方法”一节 第 336 页	为给定 Web 服务请求的结果设置监听器。

成员名称	说明
“SetListener 方法”一节 第 337 页	为此 WSBase 实例发出的所有 Web 服务请求的结果设置监听器。
“SetProperty 方法”一节 第 337 页	为此 WSBase 实例设置配置属性。
“SetQAManager 方法”一节 第 338 页	设置此 Web 服务客户端用于发出 Web 服务请求的 QAManagerBase。
“SetRequestProperty 方法”一节第 338 页	为此 WSBase 发出的 Web 服务请求设置请求属性。
“SetServiceID 方法”一节 第 339 页	为此 WSBase 实例设置用户定义 ID。

WSBase 构造函数

使用配置属性文件指定的属性构造一个 WSBase 实例。

语法

Visual Basic

```
Overloads Public Sub New( _
  ByVal iniFile As String _
)
```

C#

```
public WSBase(
  string  iniFile
);
```

参数

- **iniFile** 包含配置属性的文件。

注释

有效配置属性有：

LOG_FILE，记录运行时信息的文件。

LOG_LEVEL，值为 0 到 6，用于控制所记录信息的详细程度，6 为最高的详细程度。

WS_CONNECTOR_ADDRESS，MobiLink 服务器中 Web 服务连接器的地址。

缺省 WS_CONNECTOR_ADDRESS 是 "ianywhere.connector.webservices\\"。

异常

- “WSException 类”一节第 339 页- 如果构造 WSBase 时存在问题，则抛出。

WSBase 构造函数

使用默认属性构造一个 WSBase 实例。

语法

Visual Basic
Overloads Public Sub **New()**

C#
public **WSBase();**

异常

- “WSException 类”一节第 339 页- 如果构造 WSBase 时存在问题，则抛出。

ClearRequestProperties 方法

清除为此 WSBase 设置的全部请求属性。

语法

Visual Basic
Public Sub **ClearRequestProperties()**

C#
public void **ClearRequestProperties();**

GetResult 方法

获取代表 Web 服务请求结果的 WSResult 对象。

语法

Visual Basic
```
Public Function GetResult( _
  ByVal requestID As String _
) As iAnywhere.QAnywhere.WS.WSResult
```

C#
```
public iAnywhere.QAnywhere.WS.WSResult GetResult(
  string  requestID
);
```

参数

- **requestID**　Web 服务请求的 ID。

返回值

代表 Web 服务请求结果的一个 WSResult 实例。

另请参见

- “WSBase 类”一节第 333 页
- “WSBase 成员”一节第 333 页
- “WSStatus 枚举”一节第 383 页

GetServiceID 方法

获取此 WSBase 实例的服务 ID。

语法

Visual Basic
```
Public Function GetServiceID() As String
```

C#
```
public string  GetServiceID();
```

返回值

服务 ID。

SetListener 方法

为给定 Web 服务请求的结果设置监听器。

语法

Visual Basic
```
Overloads Public Sub SetListener( _
  ByVal requestID As String, _
  ByVal listener As iAnywhere.QAnywhere.WS.WSListener _
)
```

C#
```
public void SetListener(
  string  requestID,
  iAnywhere.QAnywhere.WS.WSListener listener
);
```

参数

- **requestID**　要监听结果的 Web 服务请求的 ID。
- **listener**　给定 Web 服务请求的结果可用时调用的监听器对象。

注释

监听器通常用于获取此服务的 asyncXYZ 方法的结果。

要删除监听器，可在调用 SetListener 时将 listener 参数设置为空值。

此方法会替换由先前的任意一次 SetListener 调用所设置的监听器。

SetListener 方法

为此 WSBase 实例发出的所有 Web 服务请求的结果设置监听器。

语法

Visual Basic

```
Overloads Public Sub SetListener( _
  ByVal listener As iAnywhere.QAnywhere.WS.WSListener _
)
```

C#

```
public void SetListener(
  iAnywhere.QAnywhere.WS.WSListener listener
);
```

参数

- **listener** Web 服务请求的结果可用时调用的监听器对象。

注释

监听器通常用于获取此服务的 asyncXYZ 方法的结果。

要删除监听器，可在调用 SetListener 时将 listener 参数设置为空值。

此方法会替换由先前的任意一次 SetListener 调用所设置的监听器。

SetProperty 方法

为此 WSBase 实例设置配置属性。

语法

Visual Basic

```
Public Sub SetProperty( _
  ByVal property As String, _
  ByVal val As String _
)
```

C#

```
public void SetProperty(
  string  property,
  string  val
);
```

参数

- **property** 要设置的属性名称。
- **val** 属性值。

注释

在发出任何异步或同步 Web 服务请求前，必须设置配置属性。如果在发出 Web 服务请求后调用此方法，则此方法无效。

有效配置属性有：

LOG_FILE，记录运行时信息的文件。

LOG_LEVEL，值为 0 到 6，用于控制所记录信息的详细程度，6 为最高的详细程度。

WS_CONNECTOR_ADDRESS，MobiLink 服务器中 Web 服务连接器的地址。缺省值为："ianywhere.connector.webservices\\"。

SetQAManager 方法

设置此 Web 服务客户端用于发出 Web 服务请求的 QAManagerBase。

语法

Visual Basic

```
Public Sub SetQAManager( _
  ByVal mgr As QAManagerBase _
)
```

C#

```
public void SetQAManager(
  QAManagerBase mgr
);
```

参数

- **mgr** 要使用的 QAManagerBase。

注释

如果使用 EXPLICIT_ACKNOWLEDGEMENT QAManager，可以通过调用 WSResult 的 acknowledge() 方法确认异步 Web 服务请求的结果。即使是使用 EXPLICIT_ACKNOWLEDGEMENT QAManager，也会自动确认同步 Web 服务请求的结果。如果使用 IMPLICIT_ACKNOWLEDGEMENT QAManager，任何 Web 服务请求的结果都会自动确认。

SetRequestProperty 方法

为此 WSBase 发出的 Web 服务请求设置请求属性。

语法

Visual Basic

```
Public Sub SetRequestProperty( _
  ByVal name As String, _
  ByVal value As Object _
)
```


C#
```
public void SetRequestProperty(
  string  name,
  object value
);
```

参数

- **name** 要设置的属性名称。
- **value** 属性值。

注释

对由此 WSBase 发送的每个 QAMessage 都设置一个请求属性，直到此属性被清除。请求属性可通过将其设置为空值来清除。消息属性的类型由值参数的类来确定。例如，如果值是一个 Int32 实例，则 SetIntProperty 用于设置 QAMessage 的属性。

SetServiceID 方法

为此 WSBase 实例设置用户定义 ID。

语法

Visual Basic
```
Public Sub SetServiceID( _
  ByVal serviceID As String _
)
```

C#
```
public void SetServiceID(
  string  serviceID
);
```

参数

- **serviceID** 服务 ID。

注释

服务 ID 应设置为此 WSBase 实例独有的值。它在内部使用，用来构成发送和接收 Web 服务请求的队列名称。服务 ID 应在应用程序会话间持久存在，以便检索之前会话中发出的 Web 服务请求的结果。

WSException 类

此类表示处理 Web 服务请求期间出现的异常。

语法

Visual Basic
```
Public Class WSException
  Inherits Exception
```

C#
public class **WSException** :
  Exception

WSException 成员

公共静态字段（共享）

成员名称	说明
“WS_STATUS_HTTP_ERROR 字段”一节第 342 页	错误代码，指出 Web 服务连接器发出的 Web 服务 HTTP 请求中存在错误。
“WS_STATUS_HTTP_OK 字段”一节第 343 页	错误代码，指出 Web 服务连接器发出的 Web 服务 HTTP 请求成功。
“WS_STATUS_HTTP_RETRIES_EXCEEDED 字段”一节第 343 页	错误代码，指出 Web 服务连接器的 HTTP 重试次数超出限制。
“WS_STATUS_SOAP_PARSE_ERROR 字段”一节第 343 页	错误代码，指出分析 SOAP 响应或请求时，Web 服务运行库或 Web 服务连接器中存在错误。

公共构造函数

成员名称	说明
“WSException 构造函数”一节第 341 页	使用指定的错误消息构造新异常。
“WSException 构造函数”一节第 341 页	使用指定的错误消息和错误代码构造新异常。
“WSException 构造函数”一节第 342 页	构造一个新异常。

公共属性

成员名称	说明
“ErrorCode 属性”一节第 343 页	与此异常相关联的错误代码。
HelpLink（继承自 Exception）	获取或设置可转到与此异常相关联的帮助文件的链接。

成员名称	说明
InnerException（继承自 Exception）	获取导致当前异常的 System.Exception 实例。
Message（继承自 Exception）	获取描述当前异常的消息。
Source（继承自 Exception）	获取或设置引起错误的应用程序或对象的名称。
StackTrace（继承自 Exception）	获取在抛出当前异常时调用堆栈上的框架的字符串表示形式。
TargetSite（继承自 Exception）	获取抛出当前异常的方法。

公共方法

成员名称	说明
GetBaseException（继承自 Exception）	当在派生类中被替换时，返回 System.Exception，它是导致产生一个或多个后续异常的根本原因。
GetObjectData（继承自 Exception）	在派生类中被替换时，利用有关异常的信息设置 System.Runtime.Serialization.SerializationInfo。
ToString（继承自 Exception）	创建并返回当前异常的字符串表示形式。

WSException 构造函数

使用指定的错误消息构造新异常。

语法

Visual Basic

```
Overloads Public Sub New( _
  ByVal msg As String _
)
```

C#

```
public WSException(
  string  msg
);
```

参数

- **msg**　错误消息。

WSException 构造函数

使用指定的错误消息和错误代码构造新异常。

语法

Visual Basic
```
Overloads Public Sub New( _
  ByVal msg As String, _
  ByVal errorCode As Integer _
)
```

C#
```
public WSException(
  string  msg,
  int errorCode
);
```

参数

- **msg**　错误消息。
- **errorCode**　错误代码。

WSException 构造函数

构造一个新异常。

语法

Visual Basic
```
Overloads Public Sub New( _
  ByVal ex As System.Exception _
)
```

C#
```
public WSException(
  System.Exception ex
);
```

参数

- **ex**　异常。

WS_STATUS_HTTP_ERROR 字段

错误代码，指出 Web 服务连接器发出的 Web 服务 HTTP 请求中存在错误。

语法

Visual Basic
```
Public Shared WS_STATUS_HTTP_ERROR As Integer
```

C#
```
public const int WS_STATUS_HTTP_ERROR;
```

WS_STATUS_HTTP_OK 字段

错误代码，指出 Web 服务连接器发出的 Web 服务 HTTP 请求成功。

语法

Visual Basic
Public Shared **WS_STATUS_HTTP_OK** As Integer

C#
public const int **WS_STATUS_HTTP_OK**;

WS_STATUS_HTTP_RETRIES_EXCEEDED 字段

错误代码，指出 Web 服务连接器的 HTTP 重试次数超出限制。

语法

Visual Basic
Public Shared **WS_STATUS_HTTP_RETRIES_EXCEEDED** As Integer

C#
public const int **WS_STATUS_HTTP_RETRIES_EXCEEDED**;

WS_STATUS_SOAP_PARSE_ERROR 字段

错误代码，指出分析 SOAP 响应或请求时，Web 服务运行库或 Web 服务连接器中存在错误。

语法

Visual Basic
Public Shared **WS_STATUS_SOAP_PARSE_ERROR** As Integer

C#
public const int **WS_STATUS_SOAP_PARSE_ERROR**;

ErrorCode 属性

与此异常相关联的错误代码。

语法

Visual Basic
Public Property **ErrorCode** As Integer

C#
public int **ErrorCode** {get;set;}

WSFaultException 类

此类表示来自 Web 服务连接器的 SOAP 错误异常。

语法

Visual Basic
Public Class **WSFaultException**
  Inherits WSException

C#
public class **WSFaultException** :
  WSException

WSFaultException 成员

公共构造函数

成员名称	说明
“WSFaultException 构造函数”一节第 345 页	使用指定的错误消息构造新异常。

公共属性

成员名称	说明
“ErrorCode 属性”一节第 343 页（继承自 WSException）	与此异常相关联的错误代码。
HelpLink（继承自 Exception）	获取或设置可转到与此异常相关联的帮助文件的链接。
InnerException（继承自 Exception）	获取导致当前异常的 System.Exception 实例。
Message（继承自 Exception）	获取描述当前异常的消息。
Source（继承自 Exception）	获取或设置引起错误的应用程序或对象的名称。
StackTrace（继承自 Exception）	获取在抛出当前异常时调用堆栈上的框架的字符串表示形式。
TargetSite（继承自 Exception）	获取抛出当前异常的方法。

公共方法

成员名称	说明
GetBaseException（继承自 Exception）	当在派生类中被替换时，返回 System.Exception，它是导致产生一个或多个后续异常的根本原因。
GetObjectData（继承自 Exception）	在派生类中被替换时，利用有关异常的信息设置 System.Runtime.Serialization.SerializationInfo。
ToString（继承自 Exception）	创建并返回当前异常的字符串表示形式。

WSFaultException 构造函数

使用指定的错误消息构造新异常。

语法

Visual Basic

```
Public Sub New( _
  ByVal msg As String _
)
```

C#

```
public WSFaultException(
  string  msg
);
```

参数

- **msg**　错误消息。

WSListener 接口

此类表示用于监听 Web 服务请求结果的监听器。

语法

Visual Basic

```
Public Interface WSListener
```

C#

```
public interface WSListener
```

WSListener 成员

公共方法

成员名称	说明
“OnException 方法”一节 第 346 页	在处理异步 Web 服务请求的结果期间出现异常时调用。
“OnResult 方法”一节 第 346 页	使用异步 Web 服务请求的结果调用。

OnException 方法

在处理异步 Web 服务请求的结果期间出现异常时调用。

语法

Visual Basic

```
Public Sub OnException( _
  ByVal e As iAnywhere.QAnywhere.WS.WSException, _
  ByVal wsResult As iAnywhere.QAnywhere.WS.WSResult _
)
```

C#

```
public void OnException(
  iAnywhere.QAnywhere.WS.WSException e,
  iAnywhere.QAnywhere.WS.WSResult wsResult
);
```

参数

- **e** 处理结果期间发生的 WSException。
- **wsResult** 可从中获取请求 ID 的 WSResult。此 WSResult 的值未定义。

OnResult 方法

使用异步 Web 服务请求的结果调用。

语法

Visual Basic

```
Public Sub OnResult( _
  ByVal wsResult As iAnywhere.QAnywhere.WS.WSResult _
)
```

C#

```
public void OnResult(
  iAnywhere.QAnywhere.WS.WSResult wsResult
);
```


参数

- **WSResult** 描述 Web 服务请求的结果的 WSResult。

WSResult 类

此类表示 Web 服务请求的结果。

语法

Visual Basic
Public Class **WSResult**

C#
public class **WSResult**

注释

可通过以下三种方式获取 WSResult 对象：

- 将其传递到 WSListener.onResult。

- 由编译器生成的服务代理的 asyncXYZ 方法返回。

- 通过调用具有指定请求 ID 的 WSBase.getResult 获取。

WSResult 成员

公共方法

成员名称	说明
“Acknowledge 方法”一节 第 351 页	确认此 WSResult 已处理。
“GetArrayValue 方法”一节 第 352 页	从此 WSResult 获取一组复杂类型的值。
“GetBoolArrayValue 方法”一节第 352 页	从此 WSResult 获取一组布尔 (bool) 值。
“GetBooleanArrayValue 方法”一节第 353 页	从此 WSResult 获取一组布尔 (Boolean) 值。
“GetBooleanValue 方法”一节 第 353 页	从此 WSResult 获取一个布尔 (Boolean) 值。
“GetBoolValue 方法”一节 第 354 页	从此 WSResult 获取一个布尔 (bool) 值。

成员名称	说明
“GetByteArrayValue 方法”一节第 354 页	从此 WSResult 获取一组字节值。
“GetByteValue 方法”一节第 355 页	从此 WSResult 获取一个字节值。
“GetCharArrayValue 方法”一节第 355 页	从此 WSResult 获取一组字符值。
“GetCharValue 方法”一节第 356 页	从此 WSResult 获取一个字符值。
“GetDecimalArrayValue 方法”一节第 356 页	从此 WSResult 获取一组十进制值。
“GetDecimalValue 方法”一节第 357 页	从此 WSResult 获取一个十进制值。
“GetDoubleArrayValue 方法”一节第 357 页	从此 WSResult 获取一组双精度值。
“GetDoubleValue 方法”一节第 358 页	从此 WSResult 获取一个双精度值。
“GetErrorMessage 方法”一节第 359 页	获取错误消息。
“GetFloatArrayValue 方法”一节第 359 页	从此 WSResult 获取一组浮点值。
“GetFloatValue 方法”一节第 359 页	从此 WSResult 获取一个浮点值。
“GetInt16ArrayValue 方法”一节第 360 页	从此 WSResult 获取一组 Int16 值。
“GetInt16Value 方法”一节第 360 页	从此 WSResult 获取一个 Int16 值。
“GetInt32ArrayValue 方法”一节第 361 页	从此 WSResult 获取一组 Int32 值。
“GetInt32Value 方法”一节第 362 页	从此 WSResult 获取一个 Int32 值。

成员名称	说明
“GetInt64ArrayValue 方法”一节第 362 页	从此 WSResult 获取一组 Int64 值。
“GetInt64Value 方法”一节第 363 页	从此 WSResult 获取一个 Int64 值。
“GetIntArrayValue 方法”一节第 363 页	从此 WSResult 获取一组整型值。
“GetIntValue 方法”一节第 364 页	从此 WSResult 获取一个整型值。
“GetLongArrayValue 方法”一节第 364 页	从此 WSResult 获取一组长整型值。
“GetLongValue 方法”一节第 365 页	从此 WSResult 获取一个长整型值。
“GetNullableBoolArrayValue 方法”一节第 365 页	从此 WSResult 获取一组布尔 (bool) 值。
“GetNullableBoolValue 方法”一节第 366 页	从此 WSResult 获取一个布尔 (bool) 值。
“GetNullableDecimalArrayValue 方法”一节第 366 页	从此 WSResult 获取一组 NullableDecimal 值。
“GetNullableDecimalValue 方法”一节第 367 页	从此 WSResult 获取一个 NullableDecimal 值。
“GetNullableDoubleArrayValue 方法”一节第 367 页	从此 WSResult 获取一组双精度值。
“GetNullableDoubleValue 方法”一节第 368 页	从此 WSResult 获取一个双精度值。
“GetNullableFloatArrayValue 方法”一节第 368 页	从此 WSResult 获取一组浮点值。
“GetNullableFloatValue 方法”一节第 369 页	从此 WSResult 获取一个浮点值。
“GetNullableIntArrayValue 方法”一节第 369 页	从此 WSResult 获取一组整型值。

成员名称	说明
“GetNullableIntValue 方法”一节第 370 页	从此 WSResult 获取一个整型值。
“GetNullableLongArrayValue 方法”一节第 370 页	从此 WSResult 获取一组长整型值。
“GetNullableLongValue 方法”一节第 371 页	从此 WSResult 获取一个 Int64 值。
“GetNullableSByteArrayValue 方法”一节第 371 页	从此 WSResult 获取一组字节值。
“GetNullableSByteValue 方法”一节第 372 页	从此 WSResult 获取一个字节值。
“GetNullableShortArrayValue 方法”一节第 372 页	从此 WSResult 获取一组短整型值。
“GetNullableShortValue 方法”一节第 373 页	从此 WSResult 获取一个短整型值。
“GetObjectArrayValue 方法”一节第 373 页	从此 WSResult 获取一组对象值。
“GetObjectValue 方法”一节第 374 页	从此 WSResult 获取一个对象值。
“GetRequestID 方法”一节第 374 页	获取此 WSResult 代表的请求 ID。
“GetSByteArrayValue 方法”一节第 374 页	从此 WSResult 获取一组 sbyte 值。
“GetSByteValue 方法”一节第 375 页	从此 WSResult 获取一个 sbyte 值。
“GetShortArrayValue 方法”一节第 376 页	从此 WSResult 获取一组短整型值。
“GetShortValue 方法”一节第 376 页	从此 WSResult 获取一个短整型值。
“GetSingleArrayValue 方法”一节第 377 页	从此 WSResult 获取一组单精度值。

成员名称	说明
“GetSingleValue 方法”一节 第 377 页	从此 WSResult 获取一个单精度值。
“GetStatus 方法”一节 第 378 页	获取此 WSResult 的状态。
“GetStringArrayValue 方法”一节第 378 页	从此 WSResult 获取一组字符串值。
“GetStringValue 方法”一节 第 379 页	从此 WSResult 获取一个字符串值。
“GetUIntArrayValue 方法”一节第 379 页	从此 WSResult 获取一组无符号整型值。
“GetUIntValue 方法”一节 第 380 页	从此 WSResult 获取一个无符号整型值。
“GetULongArrayValue 方法”一节第 380 页	从此 WSResult 获取一组无符号长整型值。
“GetULongValue 方法”一节 第 381 页	从此 WSResult 获取一个无符号长整型值。
“GetUShortArrayValue 方法”一节第 381 页	从此 WSResult 获取一组无符号短整型值。
“GetUShortValue 方法”一节 第 382 页	从此 WSResult 获取一个无符号短整型值。
“GetValue 方法”一节 第 382 页	从此 WSResult 获取复杂类型值。
“SetLogger 方法”一节 第 383 页	打开或关闭调试。

Acknowledge 方法

确认此 WSResult 已处理。

语法

Visual Basic
Public Sub **Acknowledge()**

C#
public void **Acknowledge();**

注释

仅在使用 EXPLICIT_ACKNOWLEDGEMENT QAManager 时，此方法才有用。

GetArrayValue 方法

从此 WSResult 获取一组复杂类型的值。

语法

Visual Basic
Public Function **GetArrayValue(_**
  ByVal *parentName* As String _
) As iAnywhere.QAnywhere.WS.WSSerializable()

C#
public iAnywhere.QAnywhere.WS.WSSerializable[] **GetArrayValue(**
  string *parentName*
);

参数

- **parentName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetBoolArrayValue 方法

从此 WSResult 获取一组布尔 (bool) 值。

语法

Visual Basic
Public Function **GetBoolArrayValue(_**
  ByVal *elementName* As String _
) As Boolean()

C#
public bool[] **GetBoolArrayValue(**
  string *elementName*
);

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetBooleanArrayValue 方法

从此 WSResult 获取一组布尔 (Boolean) 值。

语法

Visual Basic

```
Public Function GetBooleanArrayValue( _
   ByVal elementName As String _
) As Boolean()
```

C#

```
public bool[] GetBooleanArrayValue(
   string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetBooleanValue 方法

从此 WSResult 获取一个布尔 (Boolean) 值。

语法

Visual Basic

```
Public Function GetBooleanValue( _
   ByVal childName As String _
) As Boolean
```

C#

```
public bool GetBooleanValue(
   string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetBoolValue 方法

从此 WSResult 获取一个布尔 (bool) 值。

语法

Visual Basic

```
Public Function GetBoolValue( _
   ByVal childName As String _
) As Boolean
```

C#

```
public bool GetBoolValue(
   string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetByteArrayValue 方法

从此 WSResult 获取一组字节值。

语法

Visual Basic

```
Public Function GetByteArrayValue( _
   ByVal elementName As String _
) As Byte()
```

C#

```
public byte[] GetByteArrayValue(
```



```
  string  elementName
);
```

参数

- **elementName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetByteValue 方法

从此 WSResult 获取一个字节值。

语法

Visual Basic

```
Public Function GetByteValue( _
  ByVal childName As String _
) As Byte
```

C#

```
public byte GetByteValue(
  string  childName
);
```

参数

- **childName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetCharArrayValue 方法

从此 WSResult 获取一组字符值。

语法

Visual Basic

```
Public Function GetCharArrayValue( _
  ByVal elementName As String _
) As Char()
```

C#
```
public char[] GetCharArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetCharValue 方法

从此 WSResult 获取一个字符值。

语法

Visual Basic
```
Public Function GetCharValue( _
  ByVal childName As String _
) As Char
```

C#
```
public char GetCharValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetDecimalArrayValue 方法

从此 WSResult 获取一组十进制值。

语法

Visual Basic
```
Public Function GetDecimalArrayValue( _
```



```
  ByVal elementName As String _
) As Decimal()

C#
public decimal[] GetDecimalArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetDecimalValue 方法

从此 WSResult 获取一个十进制值。

语法

```
Visual Basic
Public Function GetDecimalValue( _
  ByVal childName As String _
) As Decimal

C#
public decimal GetDecimalValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetDoubleArrayValue 方法

从此 WSResult 获取一组双精度值。

语法

Visual Basic
```
Public Function GetDoubleArrayValue( _
  ByVal elementName As String _
) As Double()
```

C#
```
public double[] GetDoubleArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetDoubleValue 方法

从此 WSResult 获取一个双精度值。

语法

Visual Basic
```
Public Function GetDoubleValue( _
  ByVal childName As String _
) As Double
```

C#
```
public double GetDoubleValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetErrorMessage 方法

获取错误消息。

语法

Visual Basic
Public Function **GetErrorMessage()** As String

C#
public string **GetErrorMessage();**

返回值

错误消息。

GetFloatArrayValue 方法

从此 WSResult 获取一组浮点值。

语法

Visual Basic
```
Public Function GetFloatArrayValue( _
  ByVal elementName As String _
) As Single()
```

C#
```
public float [] GetFloatArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetFloatValue 方法

从此 WSResult 获取一个浮点值。

语法

Visual Basic
Public Function **GetFloatValue(_**

```
  ByVal childName As String _
) As Single

C#
public float  GetFloatValue(
  string  childName
);
```

参数

- **childName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetInt16ArrayValue 方法

从此 WSResult 获取一组 Int16 值。

语法

```
Visual Basic
Public Function GetInt16ArrayValue( _
  ByVal elementName As String _
) As Short()

C#
public short[] GetInt16ArrayValue(
  string  elementName
);
```

参数

- **elementName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetInt16Value 方法

从此 WSResult 获取一个 Int16 值。

语法

Visual Basic

```
Public Function GetInt16Value( _
  ByVal childName As String _
) As Short
```

C#

```
public short GetInt16Value(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetInt32ArrayValue 方法

从此 WSResult 获取一组 Int32 值。

语法

Visual Basic

```
Public Function GetInt32ArrayValue( _
  ByVal elementName As String _
) As Integer()
```

C#

```
public int[] GetInt32ArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetInt32Value 方法

从此 WSResult 获取一个 Int32 值。

语法

Visual Basic
```
Public Function GetInt32Value( _
  ByVal childName As String _
) As Integer
```

C#
```
public int GetInt32Value(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetInt64ArrayValue 方法

从此 WSResult 获取一组 Int64 值。

语法

Visual Basic
```
Public Function GetInt64ArrayValue( _
  ByVal elementName As String _
) As Long()
```

C#
```
public long[] GetInt64ArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetInt64Value 方法

从此 WSResult 获取一个 Int64 值。

语法

Visual Basic

```
Public Function GetInt64Value( _
  ByVal childName As String _
) As Long
```

C#

```
public long GetInt64Value(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetIntArrayValue 方法

从此 WSResult 获取一组整型值。

语法

Visual Basic

```
Public Function GetIntArrayValue( _
  ByVal elementName As String _
) As Integer()
```

C#

```
public int[] GetIntArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetIntValue 方法

从此 WSResult 获取一个整型值。

语法

Visual Basic
```
Public Function GetIntValue( _
  ByVal childName As String _
) As Integer
```

C#
```
public int GetIntValue(
  string  childName
);
```

参数

- **childName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetLongArrayValue 方法

从此 WSResult 获取一组长整型值。

语法

Visual Basic
```
Public Function GetLongArrayValue( _
  ByVal elementName As String _
) As Long()
```

C#
```
public long[] GetLongArrayValue(
  string  elementName
);
```

参数

- **elementName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetLongValue 方法

从此 WSResult 获取一个长整型值。

语法

Visual Basic

```
Public Function GetLongValue( _
  ByVal childName As String _
) As Long
```

C#

```
public long GetLongValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableBoolArrayValue 方法

从此 WSResult 获取一组布尔 (bool) 值。

语法

Visual Basic

```
Public Function GetNullableBoolArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableBool()
```

C#

```
public iAnywhere.QAnywhere.WS.NullableBool[] GetNullableBoolArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableBoolValue 方法

从此 WSResult 获取一个布尔 (bool) 值。

语法

Visual Basic

```
Public Function GetNullableBoolValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableBool
```

C#

```
public iAnywhere.QAnywhere.WS.NullableBool GetNullableBoolValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableDecimalArrayValue 方法

从此 WSResult 获取一组 NullableDecimal 值。

语法

Visual Basic

```
Public Function GetNullableDecimalArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableDecimal()
```

C#

```
public iAnywhere.QAnywhere.WS.Nullabledecimal[] GetNullableDecimalArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableDecimalValue 方法

从此 WSResult 获取一个 NullableDecimal 值。

语法

Visual Basic

```
Public Function GetNullableDecimalValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableDecimal
```

C#

```
public iAnywhere.QAnywhere.WS.Nullabledecimal GetNullableDecimalValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableDoubleArrayValue 方法

从此 WSResult 获取一组双精度值。

语法

Visual Basic

```
Public Function GetNullableDoubleArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableDouble()
```

C#

```
public iAnywhere.QAnywhere.WS.Nullabledouble[] GetNullableDoubleArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableDoubleValue 方法

从此 WSResult 获取一个双精度值。

语法

Visual Basic

```
Public Function GetNullableDoubleValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableDouble
```

C#

```
public iAnywhere.QAnywhere.WS.Nullabledouble GetNullableDoubleValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableFloatArrayValue 方法

从此 WSResult 获取一组浮点值。

语法

Visual Basic

```
Public Function GetNullableFloatArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableFloat()
```

C#

```
public iAnywhere.QAnywhere.WS.NullableFloat[] GetNullableFloatArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableFloatValue 方法

从此 WSResult 获取一个浮点值。

语法

Visual Basic

```
Public Function GetNullableFloatValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableFloat
```

C#

```
public iAnywhere.QAnywhere.WS.NullableFloat GetNullableFloatValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableIntArrayValue 方法

从此 WSResult 获取一组整型值。

语法

Visual Basic

```
Public Function GetNullableIntArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableInt()
```

C#

```
public iAnywhere.QAnywhere.WS.NullableInt[] GetNullableIntArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableIntValue 方法

从此 WSResult 获取一个整型值。

语法

Visual Basic
```
Public Function GetNullableIntValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableInt
```

C#
```
public iAnywhere.QAnywhere.WS.NullableInt GetNullableIntValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableLongArrayValue 方法

从此 WSResult 获取一组长整型值。

语法

Visual Basic
```
Public Function GetNullableLongArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableLong()
```

C#
```
public iAnywhere.QAnywhere.WS.NullableLong[] GetNullableLongArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableLongValue 方法

从此 WSResult 获取一个 Int64 值。

语法

Visual Basic

```
Public Function GetNullableLongValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableLong
```

C#

```
public iAnywhere.QAnywhere.WS.NullableLong GetNullableLongValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableSByteArrayValue 方法

从此 WSResult 获取一组字节值。

语法

Visual Basic

```
Public Function GetNullableSByteArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableSByte()
```

C#

```
public iAnywhere.QAnywhere.WS.NullableSbyte[] GetNullableSByteArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableSByteValue 方法

从此 WSResult 获取一个字节值。

语法

Visual Basic

```
Public Function GetNullableSByteValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableSByte
```

C#

```
public iAnywhere.QAnywhere.WS.NullableSbyte GetNullableSByteValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableShortArrayValue 方法

从此 WSResult 获取一组短整型值。

语法

Visual Basic

```
Public Function GetNullableShortArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableShort()
```

C#

```
public iAnywhere.QAnywhere.WS.NullableShort[] GetNullableShortArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetNullableShortValue 方法

从此 WSResult 获取一个短整型值。

语法

Visual Basic

```
Public Function GetNullableShortValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableShort
```

C#

```
public iAnywhere.QAnywhere.WS.NullableShort GetNullableShortValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetObjectArrayValue 方法

从此 WSResult 获取一组对象值。

语法

Visual Basic

```
Public Function GetObjectArrayValue( _
  ByVal elementName As String _
) As Object()
```

C#

```
public object[] GetObjectArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetObjectValue 方法

从此 WSResult 获取一个对象值。

语法

Visual Basic
```
Public Function GetObjectValue( _
  ByVal childName As String _
) As Object
```

C#
```
public object GetObjectValue(
  string  childName
);
```

参数

- **childName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetRequestID 方法

获取此 WSResult 代表的请求 ID。

语法

Visual Basic
```
Public Function GetRequestID() As String
```

C#
```
public string  GetRequestID();
```

返回值

请求 ID。

注释

如果想要在所运行的应用程序与发出请求时所运行的应用程序不同的情况下获取与 Web 服务请求相对应的 WSResult，则此请求 ID 在应用程序的运行间隔期应一直保留。

GetSByteArrayValue 方法

从此 WSResult 获取一组 sbyte 值。

语法

Visual Basic

```
Public Function GetSByteArrayValue( _
  ByVal elementName As String _
) As System.SByte()
```

C#

```
public System.Sbyte[] GetSByteArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetSByteValue 方法

从此 WSResult 获取一个 sbyte 值。

语法

Visual Basic

```
Public Function GetSByteValue( _
  ByVal childName As String _
) As System.SByte
```

C#

```
public System.Sbyte GetSByteValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetShortArrayValue 方法

从此 WSResult 获取一组短整型值。

语法

Visual Basic
```
Public Function GetShortArrayValue( _
  ByVal elementName As String _
) As Short()
```

C#
```
public short[] GetShortArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetShortValue 方法

从此 WSResult 获取一个短整型值。

语法

Visual Basic
```
Public Function GetShortValue( _
  ByVal childName As String _
) As Short
```

C#
```
public short GetShortValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetSingleArrayValue 方法

从此 WSResult 获取一组单精度值。

语法

Visual Basic

```
Public Function GetSingleArrayValue( _
  ByVal elementName As String _
) As Single()
```

C#

```
public float [] GetSingleArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetSingleValue 方法

从此 WSResult 获取一个单精度值。

语法

Visual Basic

```
Public Function GetSingleValue( _
  ByVal childName As String _
) As Single
```

C#

```
public float  GetSingleValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetStatus 方法

获取此 WSResult 的状态。

语法

Visual Basic
Public Function **GetStatus()** As iAnywhere.QAnywhere.WS.WSStatus

C#
public iAnywhere.QAnywhere.WS.WSStatus **GetStatus();**

返回值

状态代码。

另请参见

- “WSResult 类”一节第 347 页
- “WSResult 成员”一节第 347 页
- “WSStatus 枚举”一节第 383 页

GetStringArrayValue 方法

从此 WSResult 获取一组字符串值。

语法

Visual Basic
```
Public Function GetStringArrayValue( _
  ByVal elementName As String _
) As String()
```

C#
```
public string [] GetStringArrayValue(
  string  elementName
);
```

参数

- **elementName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetStringValue 方法

从此 WSResult 获取一个字符串值。

语法

Visual Basic

```
Public Function GetStringValue( _
  ByVal childName As String _
) As String
```

C#

```
public string  GetStringValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetUIntArrayValue 方法

从此 WSResult 获取一组无符号整型值。

语法

Visual Basic

```
Public Function GetUIntArrayValue( _
  ByVal elementName As String _
) As UInt32()
```

C#

```
public uint[] GetUIntArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetUIntValue 方法

从此 WSResult 获取一个无符号整型值。

语法

Visual Basic
```
Public Function GetUIntValue( _
  ByVal childName As String _
) As UInt32
```

C#
```
public uint GetUIntValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetULongArrayValue 方法

从此 WSResult 获取一组无符号长整型值。

语法

Visual Basic
```
Public Function GetULongArrayValue( _
  ByVal elementName As String _
) As UInt64()
```

C#
```
public ulong[] GetULongArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetULongValue 方法

从此 WSResult 获取一个无符号长整型值。

语法

Visual Basic

```
Public Function GetULongValue( _
  ByVal childName As String _
) As UInt64
```

C#

```
public ulong GetULongValue(
  string  childName
);
```

参数

- **childName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetUShortArrayValue 方法

从此 WSResult 获取一组无符号短整型值。

语法

Visual Basic

```
Public Function GetUShortArrayValue( _
  ByVal elementName As String _
) As UInt16()
```

C#

```
public ushort[] GetUShortArrayValue(
  string  elementName
);
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetUShortValue 方法

从此 WSResult 获取一个无符号短整型值。

语法

Visual Basic
```
Public Function GetUShortValue( _
  ByVal childName As String _
) As UInt16
```

C#
```
public ushort GetUShortValue(
  string  childName
);
```

参数

- **childName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

GetValue 方法

从此 WSResult 获取复杂类型值。

语法

Visual Basic
```
Public Function GetValue( _
  ByVal childName As String _
) As Object
```

C#
```
public object GetValue(
  string  childName
);
```

参数

- **childName**　此值在 WSDL 文档中的元素名称。

返回值

值。

异常

- “WSException 类”一节第 339 页- 如果获取值时存在问题，则抛出。

SetLogger 方法

打开或关闭调试。

语法

Visual Basic

```
Public Sub SetLogger( _
  ByVal wsLogger As iAnywhere.QAnywhere.WS.WSLogger _
)
```

C#

```
public void SetLogger(
  iAnywhere.QAnywhere.WS.WSLogger wsLogger
);
```

WSStatus 枚举

此类定义 Web 服务请求状态的代码。

语法

Visual Basic

```
Public Enum WSStatus
```

C#

```
public enum WSStatus
```

成员名称

成员名称	说明
STATUS_ERROR	处理请求时存在错误。
STATUS_QUEUED	已将此请求放入队列中等待发送给服务器。
STATUS_RESULT_AVAILABLE	请求的结果可用。
STATUS_SUCCESS	请求成功。

QAnywhere C++ API 参考

目录

QAnywhere C++ API 不支持 UltraLite 数据库。

AcknowledgementMode 类

语法

public **AcknowledgementMode**

注释

指出 QAnywhere 客户端应用程序应如何确认消息。

使用 open 方法将 IMPLICIT_ACKNOWLEDGEMENT 和 EXPLICIT_ACKNOWLEDGEMENT 模式指派给 QAManager 实例。将 TRANSACTIONAL 模式隐式指派给 QATransactionalManager 实例。

有关详细信息，请参见“初始化 QAnywhere API”一节第 57 页。

在隐式确认模式下，客户端应用程序在收到消息时会进行确认。在显式确认模式中，必须调用 QAManager 确认方法中的一种。在事务模式下，必须从 QATransactionalManager 实例调用 commit 方法来确认所有未完成的消息。服务器在客户端之间传播全部状态变化。

对于事务性消息传递，请使用 QATransactionalManager。在这种情况下，应使用 commit 方法来确认属于事务的消息。

可以使用 getMode 属性确定 QAManagerBase 实例的模式。

另请参见

“同步接收消息”一节第 74 页

“异步接收消息”一节第 75 页

“QAManager 类”一节第 420 页

“QATransactionalManager 类”一节第 486 页

“QAManagerBase 类”一节第 425 页

成员

AcknowledgementMode 的所有成员，包括所有继承的成员。

- “EXPLICIT_ACKNOWLEDGEMENT 变量”一节第 386 页
- “IMPLICIT_ACKNOWLEDGEMENT 变量”一节第 387 页
- “TRANSACTIONAL 变量”一节第 387 页

EXPLICIT_ACKNOWLEDGEMENT 变量

语法

const qa_short **AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT**

注释

表示使用其中一个 QAManager 确认方法确认已接收到的消息。

IMPLICIT_ACKNOWLEDGEMENT 变量

语法

const qa_short **AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT**

注释

表示客户端应用程序在接收到每条消息时都会进行确认。

如果同步接收消息，则返回 getMessage 方法时将确认消息。如果异步接收消息，则事件处理函数返回时将确认消息。

TRANSACTIONAL 变量

语法

const qa_short **AcknowledgementMode::TRANSACTIONAL**

注释

表示消息仅作为进行中事务的一部分进行确认。

此模式自动指派给 QATransactionalManager 实例。

MessageProperties 类

语法

public **MessageProperties**

注释

提供存储标准消息属性名称的字段。

MessageProperties 类提供标准消息属性名称。您可以将 MessageProperties 字段传递给用于获取和设置消息属性的 QAMessage 方法。

有关详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

```
QATextMessage * t_msg;
```

以下示例使用 getIntProperty 方法获取与 MSG_TYPE 相对应的值。MessageType 枚举将整数结果映射到合适的消息类型。

```
int msg_type;
t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type)
```

以下示例分别使用 MSG_TYPE 和 RASNAMES 计算消息类型和 RAS 名称。

```
void SystemQueueListener::onMessage(QAMessage * msg) {
    QATextMessage *  t_msg;
    TCHAR    buffer[512];
    int      len;
    int      msg_type;

    t_msg = msg->castToTextMessage();
    if (t_msg != NULL) {
        t_msg->getIntProperty(MessageProperties::MSG_TYPE, &msg_type);
        if (msg_type == MessageType::NETWORK_STATUS_NOTIFICATION) {
            // get RAS names using MessageProperties::RASNAMES
            len = t_msg-
>getStringProperty(MessageProperties::RASNAMES,buffer,sizeof(buffer));
        }
        // ...
    }
}
```

另请参见

“QAMessage 类”一节第 458 页

成员

MessageProperties 的所有成员，包括所有继承的成员。

ADAPTER 变量

语法

const qa_string **MessageProperties::ADAPTER**

注释

此属性名称是指连接到 QAnywhere 服务器所用的当前活动网络适配器。

它用于系统队列消息。

此字段的值为 "ias_Network.Adapter"。

将 ADAPTER 作为第一个参数传递给 getStringProperty 方法以访问关联消息属性。

有关详细信息，请参见“消息属性”一节第 687 页。

另请参见

“getStringProperty 函数”一节第 469 页

ADAPTERS 变量

语法

const qa_string **MessageProperties::ADAPTERS**

注释

此属性名称是指一个可用于连接 QAnywhere 服务器的网络适配器的分隔列表。

它用于系统队列消息。

此字段的值为 "ias_Adapters"。

将 ADAPTERS 作为第一个参数传递给 getStringProperty 方法以访问关联消息属性。

有关详细信息，请参见“消息属性”一节第 687 页和“getStringProperty 函数”一节第 469 页。

DELIVERY_COUNT 变量

语法

const qa_string **MessageProperties::DELIVERY_COUNT**

注释

此属性名称指目前已尝试传送消息的次数。

此字段的值为 "ias_DeliveryCount"。

将 DELIVERY_COUNT 作为第一个参数传递到 setStringProperty 方法或 getStringProperty 方法中以访问关联消息属性。

另请参见

“setStringProperty 函数”一节第 478 页

“getStringProperty 函数”一节第 469 页

IP 变量

语法

const qa_string **MessageProperties::IP**

注释

此属性名称是指连接到 QAnywhere 服务器所用的当前活动网络适配器的 IP 地址。

它用于系统队列消息。

此字段的值为 "ias_Network.IP"。

将 IP 作为第一个参数传递给 getStringProperty 方法以访问关联消息属性。

有关详细信息，请参见“消息属性”一节第 687 页。

另请参见

“getStringProperty 函数”一节第 469 页

MAC 变量

语法

const qa_string **MessageProperties::MAC**

注释

此属性名称是指连接到 QAnywhere 服务器所用的当前活动网络适配器的 MAC 地址。

它用于系统队列消息。

此字段的值为 "ias_Network.MAC"。

将 MAC 作为第一个参数传递给 getStringProperty 方法以访问关联消息属性。

有关详细信息，请参见“消息属性”一节第 687 页。

另请参见

“getStringProperty 函数”一节第 469 页

MSG_TYPE 变量

语法

const qa_string **MessageProperties::MSG_TYPE**

注释

此属性名称是指与 QAnywhere 消息相关的 MessageType 枚举值。

此字段的值为 "ias_MessageType"。将 MSG_TYPE 作为第一个参数传递到 setIntProperty 方法或 getIntProperty 方法中，以确定关联属性。

另请参见

“MessageType 类”一节第 396 页

“setIntProperty 函数”一节第 475 页

“getIntProperty 函数”一节第 465 页

NETWORK_STATUS 变量

语法

const qa_string **MessageProperties::NETWORK_STATUS**

注释

此属性名称是指网络连接的状态。

此字段的值为 "ias_NetworkStatus"。

如果可访问网络，则此属性的值为 1，否则为 0。网络状态用于 system 队列消息（如网络状态更改）。

有关详细信息，请参见“消息属性”一节第 687 页。

将 NETWORK_STATUS 作为第一个参数传递到 setStringProperty 方法或 getStringProperty 方法中以访问关联消息属性。

另请参见

“setStringProperty 函数”一节第 478 页

“getStringProperty 函数”一节第 469 页

ORIGINATOR 变量

语法

const qa_string **MessageProperties::ORIGINATOR**

注释

此属性名称是指消息发出方的消息存储库 ID。

此字段的值为 "ias_Originator"。

将 ORIGINATOR 作为第一个参数传递到 setStringProperty 方法中以访问关联消息属性。

另请参见

“setStringProperty 函数”一节第 478 页

“getStringProperty 函数”一节第 469 页

RAS 变量

语法

const qa_string **MessageProperties::RAS**

注释

此属性名称是指连接到 QAnywhere 服务器所用的当前活动 RAS 名称。

它用于系统队列消息。

此字段的值为 "ias_Network.RAS"。

将 RAS 作为第一个参数传递给 getStringProperty 方法以访问关联消息属性。

有关详细信息，请参见“消息属性”一节第 687 页。

另请参见

“getStringProperty 函数”一节第 469 页

RASNAMES 变量

语法

const qa_string **MessageProperties::RASNAMES**

注释

此属性名称是指一个可用于连接到 QAnywhere 服务器的 RAS 条目名称的分隔列表。

它用于系统队列消息。

此字段的值为 "ias_RASNames"。

有关详细信息，请参见“消息属性”一节第 687 页。

将 RASNAMES 作为第一个参数传递到 getStringProperty 方法中以访问关联消息属性。

另请参见

“setStringProperty 函数”一节第 478 页

“getStringProperty 函数”一节第 469 页

“setIntProperty 函数”一节第 475 页

“getIntProperty 函数”一节第 465 页

STATUS 变量

语法

const qa_string **MessageProperties::STATUS**

注释

此属性名称是指消息的当前状态。

有关值列表，请参见“StatusCodes 类”一节第 492 页。此字段的值为 "ias_Status"。

将 STATUS 作为第一个参数传递到 getIntProperty 方法中以访问关联消息属性。

另请参见

“StatusCodes 类”一节第 492 页

“setIntProperty 函数”一节第 475 页

“getIntProperty 函数”一节第 465 页

STATUS_TIME 变量

语法

const qa_string **MessageProperties::STATUS_TIME**

注释

此属性名称是指消息接收其当前状态的时间。

它的单位符合平台要求。对于 Windows/PocketPC 平台，时间戳为 SYSTEMTIME，转换为 FILETIME 并复制到 qa_long 值中。它为本地时间。此字段的值为 "ias_StatusTime"。

将 STATUS_TIME 作为第一个参数传递到 getLongProperty 方法中以访问关联的只读消息属性。

另请参见

“getLongProperty 函数”一节第 466 页

TRANSMISSION_STATUS 变量

语法

const qa_string **MessageProperties::TRANSMISSION_STATUS**

注释

此属性名称是指消息的当前传输状态。

有关值列表，请参见“StatusCodes 类”一节第 492 页。

此字段的值为 "ias_TransmissionStatus"。

将 TRANSMISSION_STATUS 作为第一个参数传递到 setIntProperty 方法或 getIntProperty 方法中以访问关联消息属性。

另请参见

“StatusCodes 类”一节第 492 页

“setIntProperty 函数”一节第 475 页

“getIntProperty 函数”一节第 465 页

MessageStoreProperties 类

语法

public **MessageStoreProperties**

注释

MessageStoreProperties 类提供标准消息属性名称。

可以将 Properties 字段传递给用于获取和设置预定义或自定义消息存储库属性的 QAManagerBase 方法。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

成员

MessageStoreProperties 的所有成员，包括所有继承的成员。

- “MAX_DELIVERY_ATTEMPTS 变量”一节第 395 页

MAX_DELIVERY_ATTEMPTS 变量

语法

const qa_string **MessageStoreProperties::MAX_DELIVERY_ATTEMPTS**

注释

此属性名称是指不使用显式确认，在消息的状态设置为 UNRECEIVABLE 之前，能够接收消息的最大次数。

此字段的值为 "ias_MaxDeliveryAttempts"。

另请参见

“StatusCodes 类”一节第 492 页

MessageType 类

语法

public **MessageType**

注释

定义 MSG_TYPE 消息属性的常量值。

以下示例显示 onSystemMessage 方法，此方法用于处理 QAnywhere 系统消息。

消息类型将与 NETWORK_STATUS_NOTIFICATION 进行比较。

```
void SystemQueueListener::onMessage(QAMessage * msg) {
    QATextMessage *  t_msg;
    TCHAR    buffer[512];
    int      len;
    int      msg_type;

    t_msg = msg->castToTextMessage();
    if (t_msg != NULL) {
        t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );
        if (msg_type == MessageType::NETWORK_STATUS_NOTIFICATION) {
            // get network names using MessageProperties::NETWORK
            len = t_msg-
>getStringProperty(MessageProperties::NETWORK,buffer,sizeof(buffer));
        }
        // ...
    }
}
```

成员

MessageType 的所有成员，包括所有继承的成员。

- “NETWORK_STATUS_NOTIFICATION 变量”一节第 396 页
- “PUSH_NOTIFICATION 变量”一节第 397 页
- “REGULAR 变量”一节第 397 页

NETWORK_STATUS_NOTIFICATION 变量

语法

const qa_int **MessageType::NETWORK_STATUS_NOTIFICATION**

注释

标识用于通知 QAnywhere 客户端应用程序网络状态更改的 QAnywhere 系统消息。

网络状态更改适用于接收系统消息的设备。使用 ADAPTER、NETWORK 和 NETWORK_STATUS 字段来标识新的网络状态信息。

有关详细信息，请参见“预定义的消息属性”一节第 687 页。

PUSH_NOTIFICATION 变量

语法

const qa_int **MessageType::PUSH_NOTIFICATION**

注释

标识用于通知 QAnywhere 客户端应用程序推式通知的 QAnywhere 系统消息。

如果您使用 [要求时] QAnywhere 代理策略，一个典型响应就是调用 triggerSendReceive 方法以接收随中央消息服务器一同等待的消息。

有关详细信息，请参见“预定义的消息属性”一节第 687 页。

REGULAR 变量

语法

const qa_int **MessageType::REGULAR**

注释

如果不存在任何消息类型属性，则假定消息类型为 REGULAR。

消息系统不会特别处理此类型消息。

QABinaryMessage 类

语法

public **QABinaryMessage**

基类

- “QAMessage 类”一节第 458 页

注释

QABinaryMessage 对象用于发送包含未解释字节流的消息。

它继承自类 QAMessage，并添加了字节消息主体。QABinaryMessage 提供了读取和写入字节消息主体的各种方法。

首次创建消息时，消息的主体处于只写模式。消息发送后，发送消息的客户端可保留和修改该消息，而不会影响已发送的消息。可以多次发送同一消息对象。

接收到消息时，提供程序已调用 reset 方法，因此消息主体处于只读模式并且从消息主体的开头开始读取值。如果客户端尝试在只读模式下写入消息，则设置 COMMON_MSG_NOT_WRITEABLE_ERROR。

以下示例使用 writeString 方法将其后是字符串 "Anywhere" 的字符串 "Q" 写入 QABinaryMessage 实例的消息主体。

```
// Create a binary message instance.
QABinaryMessage * binary_message;
binary_message = qa_manager->createBinaryMessage();

// Set optional message properties.
binary_message->setReplyToAddress("my-queue-name");

// Write to the message body.
binary_message->writeString("Q");
binary_message->writeString("Anywhere");

// Put the message in the local database, ready for sending.
if (!qa_manager->putMessage("store-id\\queue-name", msg)) {
    handleError();
}
```

在接收端，第一次调用 readString 将返回 "Q"，而下一次 readString 调用则返回 "Anywhere"。

由 QAnywhere 代理发送消息。

有关详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页和“编写 QAnywhere 客户端应用程序”第 53 页。

成员

QABinaryMessage 的所有成员，包括所有继承的成员。

getBodyLength 函数

语法

qa_long **QABinaryMessage::getBodyLength()**

注释

返回消息主体的大小（以字节为单位）.

返回值

消息主体的大小（以字节为单位）。

readBinary 函数

语法

```
qa_int QABinaryMessage::readBinary(
  qa_bytes value,
  qa_int length
)
```


参数

- **value** 将数据读取到的缓冲区。
- **length** 要读取的最大字节数。

注释

从 QABinaryMessage 实例消息主体的未读部分开始读取指定字节数。

另请参见

“writeBinary 函数”一节第 405 页

返回值

返回读取到缓冲区的字节总数；如果已到达流末尾而没有其它数据，则返回 -1。

readBoolean 函数

语法

```
qa_bool QABinaryMessage::readBoolean(
  qa_bool * value
)
```

参数

- **value** 从字节消息流读取的 qa_bool 值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取布尔值。

另请参见

“writeBoolean 函数”一节第 406 页

返回值

当且仅当操作成功时返回 true。

readByte 函数

语法

```
qa_bool QABinaryMessage::readByte(
  qa_byte * value
)
```

参数

- **value** 从字节消息流读取的 qa_byte 值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取有符号的 8 位值。

另请参见

“writeByte 函数”一节第 406 页

返回值

当且仅当操作成功时返回 true。

readChar 函数

语法

```
qa_bool QABinaryMessage::readChar(
  qa_char * value
)
```

参数

- **value** 从字节消息流读取的 qa_char 值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取字符值。

另请参见

“writeChar 函数”一节第 407 页

返回值

读取的字符值。

readDouble 函数

语法

```
qa_bool QABinaryMessage::readDouble(
  qa_double * value
)
```

参数

- **value** 从字节消息流读取的双精度值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取双精度值。

另请参见

“writeDouble 函数”一节第 407 页

返回值

当且仅当操作成功时返回 true。

readFloat 函数

语法

```
qa_bool QABinaryMessage::readFloat(
  qa_float * value
)
```

参数

- **value** 从字节消息流读取的浮点值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取浮点值。

另请参见

“writeFloat 函数”一节第 408 页

返回值

当且仅当操作成功时返回 true。

readInt 函数

语法

```
qa_bool QABinaryMessage::readInt(
  qa_int * value
)
```

参数

- **value** 从字节消息流读取的 qa_int 值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取有符号的 32 位整数值。

另请参见

“writeInt 函数”一节第 408 页

返回值

当且仅当操作成功时返回 true。

readLong 函数

语法

```
qa_bool QABinaryMessage::readLong(
  qa_long * value
)
```

参数

- **value** 从字节消息流读取的长整型值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取有符号的 64 位整数值。

另请参见

“writeLong 函数”一节第 408 页

返回值

当且仅当操作成功时返回 true。

readShort 函数

语法

```
qa_bool QABinaryMessage::readShort(
  qa_short * value
)
```

参数

- **value** 从字节消息流读取的 qa_short 值的目标。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取有符号的 16 位值。

另请参见

“writeShort 函数”一节第 409 页

返回值

当且仅当操作成功时返回 true。

readString 函数

语法

```
qa_int QABinaryMessage::readString(
  qa_string dest,
  qa_int maxLen
)
```

参数

- **dest** 从字节消息流读取的 qa_string 值的目标。
- **maxLen** 要读取的最大字符数，包括空终止符。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取字符串值。

另请参见

“writeString 函数”一节第 409 页

返回值

返回读取到缓冲区的非空 qa_char 总数；如果没有其它数据或发生错误，则返回 -1；如果缓冲区过小，则返回 -2。

reset 函数

语法

```
void QABinaryMessage::reset()
```

注释

重置消息以便从消息主体的开头开始读取值。

reset 方法还将 QABinaryMessage 消息主体置于只读模式。

writeBinary 函数

语法

```
void QABinaryMessage::writeBinary(
  qa_const_bytes value,
  qa_int offset,
  qa_int length
)
```


参数

- **value** 写入消息主体的字节数组值。
- **offset** 要开始写入的字节数组内的偏移。
- **length** 要写入的字节数。

注释

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

另请参见

“readBinary 函数”一节第 400 页

writeBoolean 函数

语法

```
void QABinaryMessage::writeBoolean(
  qa_bool value
)
```

参数

- **value** 要写入消息主体的布尔值。

注释

将布尔值附加到 QABinaryMessage 实例的消息主体中。

布尔值由一个单字节值表示。1 代表 true；0 代表 false。

另请参见

“readBoolean 函数”一节第 401 页

writeByte 函数

语法

```
void QABinaryMessage::writeByte(
  qa_byte value
)
```

参数

- **value** 写入消息主体的字节数组值。

注释

将字节值附加到 QABinaryMessage 实例的消息主体中。

字节由一个单字节值表示。

另请参见

“readByte 函数”一节第 401 页

writeChar 函数

语法

```
void QABinaryMessage::writeChar(
  qa_char value
)
```

参数

- **value** 要写入消息主体的字符值。

注释

将字符值附加到 QABinaryMessage 实例的消息主体中。

字符参数由一个双字节值表示，首先附加高位字节。

另请参见

“readChar 函数”一节第 402 页

writeDouble 函数

语法

```
void QABinaryMessage::writeDouble(
  qa_double value
)
```

参数

- **value** 要写入消息主体的双精度值。

注释

将双精度值附加到 QABinaryMessage 实例的消息主体中。

双精度参数转换为用八字节长整型值表示。首先附加高位字节。

另请参见

“readDouble 函数”一节第 402 页

writeFloat 函数

语法

void **QABinaryMessage::writeFloat(**
  qa_float *value*
)

参数

- **value** 要写入消息主体的浮点值。

注释

将浮点值附加到 QABinaryMessage 实例的消息主体中。

浮点值可转换为 4 字节整型表示形式，首先附加高位字节。

另请参见

“readFloat 函数”一节第 403 页

writeInt 函数

语法

void **QABinaryMessage::writeInt(**
  qa_int *value*
)

参数

- **value** 要写入消息主体的整型值。

注释

将整数值附加到 QABinaryMessage 实例的消息主体中。

整型参数由一个四字节值表示，首先附加高位字节。

另请参见

“readInt 函数”一节第 403 页

writeLong 函数

语法

void **QABinaryMessage::writeLong(**
  qa_long *value*
)

参数

- **value** 要写入消息主体的长整型值。

注释

将长整型值附加到 QABinaryMessage 实例的消息主体中。

长整型参数由一个八字节值表示，首先附加高位字节。

另请参见

“readLong 函数”一节第 404 页

writeShort 函数

语法

```
void QABinaryMessage::writeShort(
  qa_short value
)
```

参数

- **value** 要写入消息主体的短整型值。

注释

将短整型值附加到 QABinaryMessage 实例的消息主体中。

短整型参数由一个双字节值表示，首先附加高位字节。

另请参见

“readShort 函数”一节第 404 页

writeString 函数

语法

```
void QABinaryMessage::writeString(
  qa_const_string value
)
```

参数

- **value** 要写入消息主体的字符串值。

注释

将字符串值附加到 QABinaryMessage 实例的消息主体中。

接收应用程序需要为每个 writeString 调用而调用 readString。

字符串的 UTF-8 表示形式最多可为 32767 个字节。

另请参见

“readString 函数”一节第 405 页

~QABinaryMessage 函数

语法

virtual **QABinaryMessage::~QABinaryMessage()**

注释

虚析构函数。

QAError 类

语法

public **QAError**

注释

此类定义与 QAnywhere 客户端应用程序相关联的错误常量。

QAError 对象由 QAManager 对象内部使用，用于跟踪与消息传递操作相关联的错误。应用程序编程人员无需创建此类的实例。应用程序编程人员应在解释 getLastError 返回的错误代码时使用这些错误常量。

```
if (qa_mgr->getLastError() != QAError::QA_NO_ERROR) {
    // Process error.
}
```

另请参见

“getLastErrorMsg 函数”一节第 437 页

成员

QAError 的所有成员，包括所有继承的成员。

COMMON_ALREADY_OPEN_ERROR 变量

语法

const qa_int **QAError::COMMON_ALREADY_OPEN_ERROR**

注释

QAManager 已打开。

COMMON_GETQUEUEDEPTH_ERROR 变量

语法

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR**

注释

获取队列深度时出错。

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 变量

语法

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG**

注释

当过滤器为 ALL 时，无法在给定目标上使用 getQueueDepth。

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 变量

语法

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID**

注释

在未设置消息存储库 ID 时，无法使用 QAManagerBase.getQueueDepth。

COMMON_GET_INIT_FILE_ERROR 变量

语法

const qa_int **QAError::COMMON_GET_INIT_FILE_ERROR**

注释

无法访问客户端属性文件。

COMMON_GET_PROPERTY_ERROR 变量

语法

const qa_int **QAError::COMMON_GET_PROPERTY_ERROR**

注释

从消息存储库中检索属性时出错。

COMMON_INIT_ERROR 变量

语法

const qa_int **QAError::COMMON_INIT_ERROR**

注释

初始化错误。

COMMON_INIT_THREAD_ERROR 变量

语法

const qa_int **QAError::COMMON_INIT_THREAD_ERROR**

注释

初始化后台线程时出错。

COMMON_INVALID_PROPERTY 变量

语法

const qa_int **QAError::COMMON_INVALID_PROPERTY**

注释

客户端属性文件中有一个无效的属性。

COMMON_MSG_ACKNOWLEDGE_ERROR 变量

语法

const qa_int **QAError::COMMON_MSG_ACKNOWLEDGE_ERROR**

注释

确认消息时出错。

COMMON_MSG_CANCEL_ERROR 变量

语法

const qa_int **QAError::COMMON_MSG_CANCEL_ERROR**

注释

取消消息时出错。

COMMON_MSG_CANCEL_ERROR_SENT 变量

语法

const qa_int **QAError::COMMON_MSG_CANCEL_ERROR_SENT**

注释

取消消息时出错。

不能取消已发送的消息。

COMMON_MSG_NOT_WRITEABLE_ERROR 变量

语法

const qa_int **QAError::COMMON_MSG_NOT_WRITEABLE_ERROR**

注释

不能写入到消息，因为它处于只读模式。

COMMON_MSG_RETRIEVE_ERROR 变量

语法

const qa_int **QAError::COMMON_MSG_RETRIEVE_ERROR**

注释

从客户端消息存储库中检索消息时出错。

COMMON_MSG_STORE_ERROR 变量

语法

const qa_int **QAError::COMMON_MSG_STORE_ERROR**

注释

在客户端消息存储库中存储消息时出错。

COMMON_MSG_STORE_NOT_INITIALIZED 变量

语法

const qa_int **QAError::COMMON_MSG_STORE_NOT_INITIALIZED**

注释

尚未为进行消息传递而初始化消息存储库。

COMMON_MSG_STORE_TOO_LARGE 变量

语法

const qa_int **QAError::COMMON_MSG_STORE_TOO_LARGE**

注释

消息存储库相对于设备上的可用磁盘空间过大。

COMMON_NOT_OPEN_ERROR 变量

语法

const qa_int **QAError::COMMON_NOT_OPEN_ERROR**

注释

QAManager 没有打开。

COMMON_NO_DEST_ERROR 变量

语法

const qa_int **QAError::COMMON_NO_DEST_ERROR**

注释

没有目标。

COMMON_NO_IMPLEMENTATION 变量

语法

const qa_int **QAError::COMMON_NO_IMPLEMENTATION**

注释

未实现函数。

COMMON_OPEN_ERROR 变量

语法

const qa_int **QAError::COMMON_OPEN_ERROR**

注释

打开消息存储库连接时出错。

COMMON_OPEN_LOG_FILE_ERROR 变量

语法

const qa_int **QAError::COMMON_OPEN_LOG_FILE_ERROR**

注释

打开日志文件时出错。

COMMON_OPEN_MAXTHREADS_ERROR 变量

语法

const qa_int **QAError::COMMON_OPEN_MAXTHREADS_ERROR**

注释

无法打开 QAManager，因为并发服务器请求的最大数不够大。

有关详细信息，请参见“-gn 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

COMMON_SELECTOR_SYNTAX_ERROR 变量

语法

const qa_int **QAError::COMMON_SELECTOR_SYNTAX_ERROR**

注释

给定选择程序存在一个语法错误。

COMMON_SET_PROPERTY_ERROR 变量

语法

const qa_int **QAError::COMMON_SET_PROPERTY_ERROR**

注释

将属性存储到消息存储库中时出错。

COMMON_TERMINATE_ERROR 变量

语法

const qa_int **QAError::COMMON_TERMINATE_ERROR**

注释

终止错误。

COMMON_UNEXPECTED_EOM_ERROR 变量

语法

const qa_int **QAError::COMMON_UNEXPECTED_EOM_ERROR**

注释

遇到了意外的消息结尾。

COMMON_UNREPRESENTABLE_TIMESTAMP 变量

语法

const qa_int **QAError::COMMON_UNREPRESENTABLE_TIMESTAMP**

注释

时间戳在可接受范围以外。

QA_NO_ERROR 变量

语法

const qa_int **QAError::QA_NO_ERROR**

注释

无错误。

QAManager 类

语法

public **QAManager**

基类

- “QAManagerBase 类”一节第 425 页

注释

QAManager 类由 QAManagerBase 派生并且管理非事务性 QAnywhere 消息传递操作。

有关派生行为的详细说明，请参见“QAManagerBase 类”一节第 425 页。

可以如 AcknowledgementMode 枚举中定义的那样对 QAManager 进行配置，以进行隐式或显式确认。要将消息作为事务的一部分进行确认，请使用 QATransactionalManager。请参见“QAManager 类”一节第 420 页。

使用 QAManagerFactory 创建 QAManager 和 QATransactionalManager 对象。

另请参见

“QAManagerFactory 类”一节第 454 页

“QAManagerBase 类”一节第 425 页

“QATransactionalManager 类”一节第 486 页

“AcknowledgementMode 类”一节第 386 页

成员

QAManager 的所有成员，包括所有继承的成员。

acknowledge 函数

语法

qa_bool **QAManager::acknowledge(**
  QAMessage * *msg*
)

参数

- **msg** 要确认的消息。

注释

确认客户端应用程序成功接收了一条 QAnywhere 消息。

确认 QAMessage 时，其 STATUS 属性将更改为 RECEIVED。QAMessage 状态更改为 RECEIVED 时，可以使用缺省删除规则将其删除。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

另请参见

“acknowledgeAll 函数”一节第 422 页

“acknowledgeUntil 函数”一节第 423 页

返回值

当且仅当操作成功时返回 true。

acknowledgeAll 函数

语法

qa_bool **QAManager::acknowledgeAll()**

注释

确认客户端应用程序成功接收了所有未确认的 QAnywhere 消息。

确认 QAMessage 时，其 STATUS 属性将更改为 RECEIVED。QAMessage 状态更改为 RECEIVED 时，可以使用缺省删除规则将其删除。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

另请参见

“acknowledge 函数”一节第 422 页

“acknowledgeUntil 函数”一节第 423 页

返回值

当且仅当操作成功时返回 true。

acknowledgeUntil 函数

语法

```
qa_bool QAManager::acknowledgeUntil(
  QAMessage * msg
)
```

参数

- **msg** 要确认的最后一个消息。还确认所有较早的未确认消息。

注释

对在给定消息之前收到的给定 QAMessage 实例和所有未确认消息进行确认。

确认 QAMessage 时，其 STATUS 属性将更改为 RECEIVED。QAMessage 状态更改为 RECEIVED 时，可以使用缺省删除规则将其删除。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

另请参见

“acknowledge 函数”一节第 422 页

“acknowledgeAll 函数”一节第 422 页

返回值

当且仅当操作成功时返回 true。

open 函数

语法

```
qa_bool QAManager::open(
  qa_short mode
)
```

参数

- **mode** 确认模式。

注释

使用给定的 AcknowledgementMode 值打开 QAManager。

open 函数必须是创建 QAManager 后调用的第一个方法。

如果检测到数据库连接错误，则可以通过调用 close 函数并随后调用 open 函数来重新打开 QAManager。重新打开 QAManager 时，不需要重新创建它、重置属性或重置消息监听器。第一次打开后便无法更改 QAManager 的属性，后续的打开调用必须提供相同的确认模式。

另请参见

“AcknowledgementMode 类”一节第 386 页

“close 函数”一节第 431 页

返回值

如果操作成功，则返回 True；否则返回 false。

recover 函数

语法

```
qa_bool QAManager::recover()
```

注释

强制所有未确认的消息变为未接收状态。

必须使用 getMessage 再次接收这些消息。

返回值

当且仅当操作成功时返回 true。

QAManagerBase 类

语法

public **QAManagerBase**

派生类

- “QAManager 类”一节第 420 页
- “QATransactionalManager 类”一节第 486 页

注释

此类用作 QATransactionalManager 和 QAManager 的基类，它们分别管理事务性和非事务性消息传递。

使用 start 方法可允许 QAManagerBase 实例监听消息。应用程序中的每个线程必须只有一个 QAManagerBase 实例。

使用此类的实例可以创建和管理 QAnywhere 消息。使用 createBinaryMessage 方法和 createTextMessage 方法来创建合适的 QAMessage 实例。QAMessage 实例提供各种设置消息内容和属性的方法。要发送 QAnywhere 消息，请使用 putMessage 将带地址的消息放置于本地消息存储库队列中。此消息由 QAnywhere 代理根据其传输策略传输，或在您调用 triggerSendReceive 时传输。

有关 qaagent 传输策略的详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

使用 close 方法关闭 QAManagerBase 实例时，消息将从内存中释放。

发生 QAException 时，可以使用 getLastError、getLastErrorMessage 和 getLastNativeError 来返回错误信息。QAManagerBase 还提供设置和获取消息存储库属性的方法。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页和“MessageStoreProperties 类”一节第 395 页。

另请参见

“QATransactionalManager 类”一节第 486 页

“QAManager 类”一节第 420 页

成员

QAManagerBase 的所有成员，包括所有继承的成员。

beginEnumStorePropertyNames 函数

语法

qa_store_property_enum_handle **QAManagerBase::beginEnumStorePropertyNames()**

注释

开始枚举消息存储库属性名称。

此方法返回的句柄提供给 nextStorePropertyName 方法。beginEnumStorePropertyNames 方法和 nextStorePropertyName 可用于在调用此方法时枚举消息存储库属性名称。不能在 beginEnumStorePropertyNames 和 endEnumStorePropertyNames 调用之间设置消息存储库属性。

另请参见

“nextStorePropertyName 函数”一节第 444 页

“beginEnumStorePropertyNames 函数”一节第 427 页

“endEnumStorePropertyNames 函数”一节第 433 页

返回值

提供给 nextStorePropertyName 的句柄。

browseClose 函数

语法

void **QAManagerBase::browseClose(**
  qa_browse_handle *handle*
)

参数

- **handle** 由开始浏览操作之一返回的句柄。

注释

释放与浏览操作关联的资源。

browseMessages 函数

语法

qa_browse_handle **QAManagerBase::browseMessages()**

注释

开始浏览消息存储库中排队的消息。

此方法返回的句柄提供给 browseNextMessage。此方法和 browseNextMessage 可用于在调用此方法时枚举消息存储库中的消息。

正在浏览消息，因此无法对这些消息进行确认。使用 getMessage 接收消息，以便可以对这些消息进行确认。

另请参见

“browseNextMessage 函数”一节第 430 页

“browseMessagesByQueue 函数”一节第 429 页

“browseMessagesByID 函数”一节第 428 页

“browseClose 函数”一节第 427 页

返回值

提供给 browseNextMessage 的句柄

browseMessagesByID 函数

语法

qa_browse_handle **QAManagerBase::browseMessagesByID(**
  qa_const_string *msgid*
)

参数

- **msgid**　消息 ID。

注释

开始浏览在消息存储库中排队的具有给定消息 ID 的消息。

此方法返回的句柄提供给 browseNextMessage。此方法和 browseNextMessage 可用于在调用此方法时枚举消息存储库中的消息。

正在浏览消息，因此无法对这些消息进行确认。使用 getMessage 接收消息，以便可以对这些消息进行确认。

另请参见

“browseNextMessage 函数”一节第 430 页

“browseMessagesByQueue 函数”一节第 429 页

“browseMessages 函数”一节第 428 页

“browseClose 函数”一节第 427 页

返回值

提供给 browseNextMessage 的句柄。

browseMessagesByQueue 函数

语法

```
qa_browse_handle QAManagerBase::browseMessagesByQueue(
  qa_const_string address
)
```

参数

- **address** 要浏览的队列。

注释

开始浏览给定队列的消息存储库中排队的消息。

此方法返回的句柄提供给 browseNextMessage。此方法和 browseNextMessage 可用于在调用此方法时枚举消息存储库中的消息。

正在浏览消息，因此无法对这些消息进行确认。使用 getMessage 接收消息，以便可以对这些消息进行确认。

另请参见

“browseNextMessage 函数”一节第 430 页

“browseMessagesByID 函数”一节第 428 页

“browseMessages 函数”一节第 428 页

“browseClose 函数”一节第 427 页

返回值

提供给 browseNextMessage 的句柄。

browseMessagesBySelector 函数

语法

```
qa_browse_handle QAManagerBase::browseMessagesBySelector(
  qa_const_string selector
)
```

参数

- **selector** 选择程序。

注释

开始浏览满足给定选择程序的消息存储库中排队的消息。

此方法返回的句柄提供给 browseNextMessage。此方法和 browseNextMessage 可用于在调用此方法时枚举消息存储库中的消息。

正在浏览消息，因此无法对这些消息进行确认。

使用 getMessage 接收消息，以便可以对这些消息进行确认。

另请参见

“browseNextMessage 函数”一节第 430 页

“browseMessagesByID 函数”一节第 428 页

“browseMessagesByQueue 函数”一节第 429 页

“browseMessages 函数”一节第 428 页

“browseClose 函数”一节第 427 页

返回值

提供给 browseNextMessage 的句柄。

browseNextMessage 函数

语法

```
QAMessage * QAManagerBase::browseNextMessage(
  qa_browse_handle handle
)
```

参数

- **handle** 由开始浏览操作之一返回的句柄。

注释

返回给定浏览操作的下一条消息，如果没有其它消息，则返回空值。

要获取被浏览消息的句柄，请使用允许您按队列或消息 ID 浏览消息的 browseMessages 方法或其它 QAManagerBase 方法。

另请参见

“browseMessages 函数”一节第 428 页

“browseMessagesByQueue 函数”一节第 429 页

“browseMessagesByID 函数”一节第 428 页

“browseClose 函数”一节第 427 页

返回值

返回下一条消息；如果没有其它消息，则返回 qa_null。

cancelMessage 函数

语法

```
qa_bool QAManagerBase::cancelMessage(
  qa_const_string msgid
)
```

参数

- **msgid** 要取消消息的 ID。

注释

取消具有给定消息 ID 的消息。

cancelMessage 方法在消息传输前将其置于已取消状态。当使用 QAnywhere 代理的缺省删除规则时，已取消的消息最终会从消息存储库中删除。

如果消息已经处于最终状态，或已经传输到中央消息传递服务器，则 CancelMessage 方法将失败。

有关删除规则的详细信息，请参见“消息删除规则”一节第 772 页。

返回值

当且仅当操作成功时返回 true。

close 函数

语法

```
qa_bool QAManagerBase::close()
```

注释

关闭到 QAnywhere 消息系统的连接并释放 QAManagerBase 使用的所有资源。

忽略对 close 的后继调用。关闭 QAManagerBase 的实例后，将无法重新打开；这种情况下，必须创建一个新的 QAManagerBase 实例，然后将其打开。

如果检测到数据库连接错误，则可以通过调用 close 函数并随后调用 open 函数来重新打开 QAManager。重新打开 QAManager 时，不需要重新创建它、重置属性或重置消息监听器。第一次打开后便无法更改 QAManager 的属性，后续的打开调用必须提供相同的确认模式。请参见“open 函数”一节第 424 页。

另请参见

“open 函数”一节第 488 页

返回值

当且仅当操作成功时返回 true。

createBinaryMessage 函数

语法

QABinaryMessage * **QAManagerBase::createBinaryMessage()**

注释

创建一个 QABinaryMessage 实例。

QABinaryMessage 实例用于发送包含未解释字节的消息主体的消息。

另请参见

“QABinaryMessage 类”一节第 398 页

返回值

一个新的 QABinaryMessage 实例。请参见“QABinaryMessage 类”一节第 398 页。

createTextMessage 函数

语法

QATextMessage * **QAManagerBase::createTextMessage()**

注释

创建一个 QATextMessage 实例。

QATextMessage 对象用于发送包含字符串消息主体的消息。

另请参见

“QATextMessage 类”一节第 481 页

返回值

一个新的 QATextMessage 实例。

deleteMessage 函数

语法

```
void QAManagerBase::deleteMessage(
  QAMessage * msg
)
```

参数

- **msg** 要删除的消息。

注释

删除一个 QAMessage 对象。

缺省情况下，关闭 QAManagerBase 时自动删除由 createTextMessage 或 createBinaryMessage 创建的消息。此方法允许对何时删除消息进行更多的控制。

endEnumStorePropertyNames 函数

语法

```
void QAManagerBase::endEnumStorePropertyNames(
  qa_store_property_enum_handle h
)
```

参数

- **h** beginEnumStorePropertyNames 返回的句柄。

注释

释放与消息存储库属性名称枚举关联的资源。

另请参见

“beginEnumStorePropertyNames 函数”一节第 427 页

getAllQueueDepth 函数

语法

```
qa_int QAManagerBase::getAllQueueDepth(
  qa_short filter
)
```

参数

- **filter** filter 指示进来的消息、外发的消息或全部消息。

注释

基于给定过滤器，返回所有队列的深度总和。

队列深度是指尚未被接收（如使用 getMessage）的消息的数量。

另请参见

“QueueDepthFilter 类”一节第 490 页.

返回值

消息的数量；如果发生错误，则返回 -1。

getBooleanStoreProperty 函数

语法

```
qa_bool QAManagerBase::getBooleanStoreProperty(
  qa_const_string name,
  qa_bool * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 布尔值的目标。

注释

获取预定义或自定义消息存储库属性的布尔值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getByteStoreProperty 函数

语法

```
qa_bool QAManagerBase::getByteStoreProperty(
  qa_const_string name,
```



```
  qa_byte * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 字节值的目标。

注释

获得预定义或自定义消息存储库属性的字节值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getDoubleStoreProperty 函数

语法

```
qa_bool QAManagerBase::getDoubleStoreProperty(
  qa_const_string name,
  qa_double * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 双精度值的目标。

注释

获取预定义或自定义消息存储库属性的双精度值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getFloatStoreProperty 函数

语法

```
qa_bool QAManagerBase::getFloatStoreProperty(
  qa_const_string name,
  qa_float * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 浮点值的目标。

注释

获取预定义或自定义消息存储库属性的浮点值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getIntStoreProperty 函数

语法

```
qa_bool QAManagerBase::getIntStoreProperty(
  qa_const_string name,
  qa_int * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 整型值的目标。

注释

获取预定义或自定义消息存储库属性的整型值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getLastError 函数

语法

qa_int **QAManagerBase::getLastError()**

注释

与上次执行的 QAManagerBase 方法相关联的错误代码。

0 表示没有错误。

有关值列表，请参见“QAError 类”一节第 411 页。

另请参见

“getLastErrorMsg 函数”一节第 437 页

“QAError 类”一节第 411 页

返回值

错误代码。

getLastErrorMsg 函数

语法

qa_string **QAManagerBase::getLastErrorMsg()**

注释

与上次执行的 QAManagerBase 方法相关联的错误文本。

如果 getLastError 返回 0，则此方法返回空值。可以在捕获 QAError 后检索此属性。

另请参见

“getLastError 函数”一节第 437 页

“QAError 类”一节第 411 页

返回值

错误消息。

getLastNativeError 函数

语法

an_sql_code **QAManagerBase::getLastNativeError()**

注释

与上次执行的 QAManagerBase 方法相关联的错误代码。

-1 表示没有错误。

有关值列表，请参见“QAError 类”一节第 411 页。

另请参见

“getLastError 函数”一节第 437 页

返回值

本地错误代码。

getLongStoreProperty 函数

语法

```
qa_bool QAManagerBase::getLongStoreProperty(
  qa_const_string name,
  qa_long * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 长整型值的目标。

注释

获取预定义或自定义消息存储库属性的长整型值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getMessage 函数

语法

```
QAMessage * QAManagerBase::getMessage(
  qa_const_string address
)
```

参数

- **address** 目标。

注释

返回下一个发送到指定地址的可用 QAMessage。请参见“QAMessage 类”一节第 458 页。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此调用将无限期阻塞直到有可用消息。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMessageBySelector 函数

语法

```
QAMessage * QAManagerBase::getMessageBySelector(
  qa_const_string address,
  qa_const_string selector
)
```

参数

- **address** 目标。
- **selector** 选择程序。

注释

返回下一个发送到指定地址并满足给定选择程序的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此调用将无限期阻塞直到有可用消息。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMessageBySelectorNoWait 函数

语法

```
QAMessage * QAManagerBase::getMessageBySelectorNoWait(
  qa_const_string address,
  qa_const_string selector
)
```

参数

- **address** 目标。
- **selector** 选择程序。

注释

返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此方法立即返回。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

返回值

返回下一条消息；如果没有可用消息，则返回 qa_null。

getMessageBySelectorTimeout 函数

语法

```
QAMessage * QAManagerBase::getMessageBySelectorTimeout(
  qa_const_string address,
  qa_const_string selector,
  qa_long timeout
)
```

参数

- **address** 目标。
- **selector** 选择程序。
- **timeout** 最长等待时间，以毫秒为单位

注释

返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，此方法等待指定的超时时间，然后返回。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMessageNoWait 函数

语法

```
QAMessage * QAManagerBase::getMessageNoWait(
  qa_const_string address
)
```

参数

- **address** 目标。

注释

返回下一个发送到给定地址的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此方法立即返回。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

返回值

返回下一条消息；如果没有可用消息，则返回 qa_null。

getMessageTimeout 函数

语法

```
QAMessage * QAManagerBase::getMessageTimeout(
  qa_const_string address,
  qa_long timeout
)
```

参数

- **address** 目标
- **timeout** 最长等待时间，以毫秒为单位

注释

返回下一个发送到给定地址的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，此方法等待指定的超时时间，然后返回。使用此方法同步接收消息。

有关异步接收消息（使用消息事件处理程序）的详细信息，请参见“异步接收消息”一节第 75 页。

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMode 函数

语法

qa_short **QAManagerBase::getMode()**

注释

返回所接收消息的 QAManager 确认模式。

有关值列表，请参见 AcknowledgementMode 类。

EXPLICIT_ACKNOWLEDGEMENT 和 IMPLICIT_ACKNOWLEDGEMENT 适用于 QAManager 实例；TRANSACTIONAL 是适用于 QATransactionalManager 实例的模式。

另请参见

“AcknowledgementMode 类”一节第 386 页

返回值

确认模式。

getQueueDepth 函数

基于给定过滤器，返回所有队列的深度总和。

语法

```
qa_int QAManagerBase::getQueueDepth(
  qa_const_string queue,
  qa_short filter
)
```

参数

- **queue** 队列名。
- **filter** filter 指示进来的消息、外发的消息或全部消息。

注释

基于给定过滤器返回队列深度（包括未提交的外发消息）。

队列深度是指没有被接收（如使用 getMessage 方法）的消息的数量。

另请参见

“QueueDepthFilter 类”一节第 490 页.

返回值

队列中消息的数量；如果发生错误，则返回 -1。

getShortStoreProperty 函数

语法

```
qa_bool QAManagerBase::getShortStoreProperty(
  qa_const_string name,
  qa_short * value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 短整型值的目标。

注释

获取预定义或自定义消息存储库属性的短整型值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

getStringStoreProperty 函数

语法

```
qa_int QAManagerBase::getStringStoreProperty(
  qa_const_string name,
  qa_string address,
  qa_int maxlen
)
```

参数

- **name** 预定义或自定义属性名称。
- **address** qa_string 值的目标。
- **maxlen** 要复制值的 qa_char 的最大数量，包括空终止符。

注释

获取预定义或自定义消息存储库属性的字符串值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

返回实际复制的非空 qa_char 的数量；如果操作失败，则返回 -1。

nextStorePropertyName 函数

语法

```
qa_int QAManagerBase::nextStorePropertyName(
  qa_store_property_enum_handle h,
  qa_string buffer,
  qa_int bufferLen
)
```

参数

- **h** beginEnumStorePropertyNames 返回的句柄。
- **buffer** 要写入属性名称的缓冲区。
- **bufferLen** 存储属性名称的缓冲区的长度。此长度必须包括用于空终止符的空间。

注释

返回给定枚举的消息存储库属性名称。

如果没有其它属性名称，则返回 -1。

另请参见

“beginEnumStorePropertyNames 函数”一节第 427 页

返回值

属性名称的长度；如果没有其它属性名称，则返回 -1。

putMessage 函数

语法

```
qa_bool QAManagerBase::putMessage(
  qa_const_string address,
  QAMessage * msg
)
```


参数

- **address** 目标。
- **msg** 消息。

注释

将一条消息放入给定目标的队列。

返回值

当且仅当操作成功时返回 true。

putMessageTimeToLive 函数

语法

```
qa_bool QAManagerBase::putMessageTimeToLive(
  qa_const_string address,
  QAMessage * msg,
  qa_long ttl
)
```

参数

- **address** 目标。
- **msg** 消息。
- **ttl** 生存期，以毫秒为单位。

注释

将一条消息放入给定目标的队列，并指定给定的生存期，以毫秒为单位。

返回值

当且仅当操作成功时返回 true。

setBooleanStoreProperty 函数

语法

```
qa_bool QAManagerBase::setBooleanStoreProperty(
  qa_const_string name,
  qa_bool value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_bool 值。

注释

将预定义或自定义消息存储库属性设置为布尔值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setByteStoreProperty 函数

语法

```
qa_bool QAManagerBase::setByteStoreProperty(
  qa_const_string name,
  qa_byte value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_byte 值。

注释

将预定义或自定义消息存储库属性设置为字节值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setDoubleStoreProperty 函数

语法

```
qa_bool QAManagerBase::setDoubleStoreProperty(
  qa_const_string name,
  qa_double value
)
```


参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_double 值。

注释

将预定义或自定义消息存储库属性设置为双精度值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setFloatStoreProperty 函数

语法

```
qa_bool QAManagerBase::setFloatStoreProperty(
  qa_const_string name,
  qa_float value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_float 值。

注释

将预定义或自定义消息存储库属性设置为浮点值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setIntStoreProperty 函数

语法

```
qa_bool QAManagerBase::setIntStoreProperty(
  qa_const_string name,
  qa_int value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_int 值。

注释

将预定义或自定义消息存储库属性设置为整型值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setLongStoreProperty 函数

语法

```
qa_bool QAManagerBase::setLongStoreProperty(
  qa_const_string name,
  qa_long value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_long 值。

注释

将预定义或自定义消息存储库属性设置为长整型值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setMessageListener 函数

语法

```
void QAManagerBase::setMessageListener(
  qa_const_string address,
  QAMessageListener * listener
)
```

参数

- **address** 监听器将应用到的目标地址。
- **listener** 与目标地址相关联的消息监听器。

注释

将消息监听器类设置为异步接收 QAnywhere 消息。

监听器是实现 onMessage（在 QAMessageListener 接口中定义的唯一方法）的类的一个实例。onMessage 可接受单一 QAMessage 参数。

setMessageListener address 参数指定用于接收消息的本地队列名称。只能向给定队列指派一个监听器。

如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 "system" 作为队列名称。使用此方法异步接收消息。

有关详细信息，请参见“异步接收消息”一节第 75 页和“系统队列”一节第 63 页。

setMessageListenerBySelector 函数

语法

```
void QAManagerBase::setMessageListenerBySelector(
  qa_const_string address,
  qa_const_string selector,
  QAMessageListener * listener
)
```

参数

- **address** 监听器将应用到的目标地址。
- **selector** 用于过滤要接收的消息的选择程序。
- **listener** 与目标地址相关联的消息监听器。

注释

使用消息选择程序，将消息监听器类设置为异步接收 QAnywhere 消息。

监听器是实现 onMessage（在 QAMessageListener 接口中定义的唯一方法）的类的一个实例。onMessage 可接受单一 QAMessage 参数。

address 参数指定用于接收消息的本地队列名称。只能向给定队列指派一个监听器。selector 参数指定用于过滤在给定地址上接收的消息的选择程序。

如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 "system" 作为队列名称。使用此方法异步接收消息。

有关详细信息，请参见“异步接收消息”一节第 75 页和“系统队列”一节第 63 页。

setProperty 函数

语法

```
qa_bool QAManagerBase::setProperty(
  qa_const_string name,
  qa_const_string value
)
```

参数

- **name** 预定义或自定义 QAnywhere Manager 配置属性名称。
- **value** QAnywhere Manager 配置属性值。

注释

用于通过编程方式设置 QAnywhere Manager 配置属性。

可以使用此方法通过指定属性名称和值来替换缺省 QAnywhere Manager 配置属性。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

还可以使用属性文件和 createQAManager 方法设置 QAnywhere Manager 配置属性。

有关详细信息，请参见“在文件中设置 QAnywhere Manager 配置属性”一节第 91 页。

注意：调用 open 方法前，必须设置必需的属性。

返回值

当且仅当操作成功时返回 true。

setShortStoreProperty 函数

语法

```
qa_bool QAManagerBase::setShortStoreProperty(
  qa_const_string name,
  qa_short value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_short 值。

注释

将预定义或自定义消息存储库属性设置为短整型值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

setStringStoreProperty 函数

语法

```
qa_bool QAManagerBase::setStringStoreProperty(
  qa_const_string name,
  qa_const_string value
)
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性的 qa_string 值。

注释

将预定义或自定义消息存储库属性设置为字符串值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 类”一节第 395 页。

有关详细信息，请参见“客户端消息存储库属性”一节第 26 页。

返回值

当且仅当操作成功时返回 true。

start 函数

语法

qa_bool **QAManagerBase::start()**

注释

启动 QAManagerBase 来接收消息监听器中进来的消息。

如果未设置消息监听器（即用 GetMessage 方法接收消息），则不需要启动 QAManagerBase。不建议使用 getMessage 方法和消息监听器来接收消息。应该使用异步（消息监听器）或同步 (getMessage) 模型二者中的一种来接收消息。

如果没有使用 stop 调用进行干预，则第一次调用之后对 start 的任何调用都将被忽略。

另请参见

“stop 函数”一节第 452 页

返回值

当且仅当操作成功时返回 true。

stop 函数

语法

qa_bool **QAManagerBase::stop()**

注释

使 QAManagerBase 停止接收进来的消息。

消息并没有丢失。再次启动管理器后才接收消息。如果没有使用 start 调用进行干预，则第一次调用之后对 stop 的任何调用都将被忽略。

另请参见

“start 函数”一节第 452 页

返回值

当且仅当操作成功时返回 true。

triggerSendReceive 函数

语法

qa_bool **QAManagerBase::triggerSendReceive()**

注释

在上载任何发送给其它客户端的消息和下载任何发送给本地客户端的消息时，皆与 QAnywhere 消息服务器保持同步。

对 triggerSendReceive 的调用会使消息在 QAnywhere 代理和中央消息传递服务器之间立即得到同步。手工 triggerSendReceive 调用将导致消息的立即传输，与 QAnywhere 代理传输策略无关。QAnywhere 代理传输策略决定消息传输的方式。例如，当客户端接收推式通知，或者当您调用 putMessage 方法来发送消息时，消息传输将定期自动进行。

有关详细信息，请参见“确定在客户端进行消息传输的时间”一节第 51 页。

另请参见

“putMessage 函数”一节第 444 页

返回值

当且仅当操作成功时返回 true。

QAManagerFactory 类

语法

public **QAManagerFactory**

注释

此类充当用于创建 QATransactionalManager 和 QAManager 对象的工厂类。

只能有一个 QAManagerFactory 实例。

另请参见

“QAManager 类”一节第 420 页

“QATransactionalManager 类”一节第 486 页

成员

QAManagerFactory 的所有成员，包括所有继承的成员。

- “createQAManager 函数”一节第 454 页
- “createQATransactionalManager 函数”一节第 455 页
- “deleteQAManager 函数”一节第 455 页
- “deleteQATransactionalManager 函数”一节第 456 页
- “getLastError 函数”一节第 456 页
- “getLastErrorMsg 函数”一节第 457 页
- “getLastNativeError 函数”一节第 457 页

createQAManager 函数

语法

```
QAManager * QAManagerFactory::createQAManager(
  qa_const_string iniFile
)
```

参数

- **iniFile**　属性文件的路径。

注释

返回带有指定属性的新 QAManager 实例。

如果属性文件参数为空值，则使用缺省属性创建 QAManager。创建 QAManager 后，可以使用 setProperty() 方法通过编程方式设置 QAnywhere Manager 属性。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

另请参见

“QAManager 类”一节第 420 页

返回值

QAManager 实例。

createQATransactionalManager 函数

语法

```
QATransactionalManager * QAManagerFactory::createQATransactionalManager(
  qa_const_string iniFile
)
```

参数

- **iniFile** 属性文件的路径。

注释

返回带有指定属性的新 QATransactionalManager 实例。

如果属性文件参数为空值，则使用缺省属性创建 QATransactionalManager。创建 QATransactionalManager 后，可以使用 setProperty 方法通过编程方式设置 QAnywhere Manager 属性。

有关 QAnywhere Manager 配置属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

另请参见

“QATransactionalManager 类”一节第 486 页

返回值

QATransactionalManager 实例。

deleteQAManager 函数

语法

```
void QAManagerFactory::deleteQAManager(
  QAManager * mgr
)
```

参数

- **mgr** 要取消的 QAManager 实例。

注释

取消一个 QAManager，释放它的资源。

没有必要使用此方法，因为调用 QAnywhereFactory_term() 时将取消创建的所有 QAManager。它是为了您的方便而提供的一种方法，用于及时释放资源。

有关详细信息，请参见“关闭 QAnywhere”一节第 88 页。

deleteQATransactionalManager 函数

语法

```
void QAManagerFactory::deleteQATransactionalManager(
  QATransactionalManager * mgr
)
```

参数

- **mgr** 要取消的 QATransactionalManager 实例。

注释

取消一个 QATransactionalManager 实例，释放它的资源。

没有必要使用此方法，因为调用 QAnywhereFactory_term() 时将取消创建的所有 QATransactionalManager 实例。它是为了您的方便而提供的一种方法，用于及时释放资源。

有关详细信息，请参见“关闭 QAnywhere”一节第 88 页。

getLastError 函数

语法

```
qa_int QAManagerFactory::getLastError()
```

注释

与上次执行的 QAManagerFactory 方法相关联的错误代码。

0 表示没有错误。

有关值列表，请参见“QAError 类”一节第 411 页。

另请参见

“getLastErrorMsg 函数”一节第 457 页

返回值

错误代码。

getLastErrorMsg 函数

语法

qa_string **QAManagerFactory::getLastErrorMsg()**

注释

与上次执行的 QAManagerFactory 方法相关联的错误文本。

如果 getLastError 返回 0，则此方法返回空值。

可以在捕获 QAError 后检索此属性。

另请参见

“getLastError 函数”一节第 456 页

“QAError 类”一节第 411 页

返回值

错误消息。

getLastNativeError 函数

语法

an_sql_code **QAManagerFactory::getLastNativeError()**

注释

与上次执行的 QAManagerFactory 方法相关联的错误代码。

-1 表示没有错误。

有关值列表，请参见“QAError 类”一节第 411 页。

另请参见

“getLastError 函数”一节第 456 页

返回值

本地错误代码。

QAMessage 类

语法

public **QAMessage**

派生类

- “QABinaryMessage 类”一节第 398 页
- “QATextMessage 类”一节第 481 页

注释

QAMessage 提供设置消息属性和标头字段的接口。

派生类 QABinaryMessage 和 QATextMessage 提供读写消息主体的专用方法。可以使用 QAMessage 方法设置预定义或自定义消息属性。

有关预定义属性名称的列表，请参见“MessageProperties 类”一节第 388 页。

有关设置消息属性和标头字段的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

成员

QAMessage 的所有成员，包括所有继承的成员。

DEFAULT_PRIORITY 变量

语法

const qa_int **QAMessage::DEFAULT_PRIORITY**

注释

缺省消息优先级。

此值为 4。这是普通优先级，因为值 0-4 为普通优先级等级，值 5-9 为加速优先级等级。

DEFAULT_TIME_TO_LIVE 变量

语法

const qa_long **QAMessage::DEFAULT_TIME_TO_LIVE**

注释

缺省消息生存期值。

此值为 0，表示消息没到期。

beginEnumPropertyNames 函数

语法

qa_property_enum_handle **QAMessage::beginEnumPropertyNames()**

注释

开始枚举消息属性名称。

此方法返回的句柄提供给 nextPropertyName。此方法和 nextPropertyName 可用于在调用此方法时枚举消息属性名称。在 beginEnumPropertyNames 和 endEnumPropertyNames 之间不能设置消息属性。

返回值

提供给 nextPropertyName 的句柄。

castToBinaryMessage 函数

语法

QABinaryMessage * **QAMessage::castToBinaryMessage()**

注释

将此 QAMessage 转换为 QABinaryMessage。

还可以使用转换运算符将此 QAMessage 转换为 QABinaryMessage。

要使用转换运算符将 QAMessage 转换为 QABinaryMessage，请执行以下操作：

```
QAMessage *msg;
QABinaryMessage *bmsg;
...
bmsg = (QABinaryMessage *)(*msg);
```

返回值

返回一个指向 QABinaryMessage 的指针；如果此消息不是 QABinaryMessage 的实例，则返回空值。

castToTextMessage 函数

语法

QATextMessage * **QAMessage::castToTextMessage()**

注释

将此 QAMessage 转换为 QATextMessage。

还可以使用转换运算符将此 QAMessage 转换为 QATextMessage。

例如，要使用转换运算符将 QAMessage 转换为 QATextMessage，请执行以下操作：

```
QAMessage *msg;
QATextMessage *bmsg;
...
bmsg = (QATextMessage *)(*msg);
```

返回值

返回一个指向 QATextMessage 的指针；如果此消息不是 QATextMessage 的实例，则返回空值。

clearProperties 函数

语法

void **QAMessage::clearProperties()**

注释

清除消息的属性。

注意：不清除消息的标头字段和主体。

endEnumPropertyNames 函数

语法

```
void QAMessage::endEnumPropertyNames(
  qa_property_enum_handle h
)
```

参数

- **h** beginEnumPropertyNames 返回的句柄。

注释

释放与消息属性名称枚举关联的资源。

getAddress 函数

语法

```
qa_const_string QAMessage::getAddress()
```

注释

获取 QAMessage 实例的目标地址。

发送消息时，此字段被忽略。send 方法完成后，此字段保留 putMessage() 中指定的目标地址。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

目标地址。

getBooleanProperty 函数

语法

```
qa_bool QAMessage::getBooleanProperty(
  qa_const_string name,
  qa_bool * value
)
```

参数

- **name** 要获取的属性的名称。
- **value** qa_bool 值的目标。

注释

获取具有指定名称的 qa_bool 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getByteProperty 函数

语法

```
qa_bool QAMessage::getByteProperty(
  qa_const_string name,
  qa_byte * value
)
```

参数

- **name** 要获取的属性的名称。
- **value** qa_byte 值的目标。

注释

获取具有指定名称的 qa_byte 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getDoubleProperty 函数

语法

```
qa_bool QAMessage::getDoubleProperty(
  qa_const_string name,
  qa_double * value
)
```

参数

- **name** 要获取的属性的名称。
- **value** qa_double 值的目标。

注释

获取具有指定名称的 qa_double 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getExpiration 函数

语法

qa_long **QAMessage::getExpiration()**

注释

获取消息的到期时间。

发送消息时，Expiration 标头字段保留未指派状态。发送方法完成后，Expiration 标头保存消息的到期时间。

此属性为只读，因为消息的到期时间是通过将 putMessageTimeToLive 的生存期参数值和当前时间相加来设置的。

到期时间的单位符合平台要求。对于 Windows/PocketPC 平台，到期时间为 SYSTEMTIME，将被转换为 FILETIME 并复制到 qa_long 值中。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

到期时间。

另请参见

“getTimestamp 函数”一节第 470 页

getFloatProperty 函数

语法

```
qa_bool QAMessage::getFloatProperty(
  qa_const_string name,
  qa_float * value
)
```


参数

- **name** 要获取的属性的名称。
- **value** qa_float 值的目标。

注释

获取具有指定名称的 qa_float 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getInReplyToID 函数

语法

qa_const_string **QAMessage::getInReplyToID()**

注释

获取此消息将回复的消息的 ID。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

回复 ID。

getIntProperty 函数

语法

```
qa_bool QAMessage::getIntProperty(
  qa_const_string name,
  qa_int * value
)
```

参数

- **name** 要获取的属性的名称。
- **value** qa_int 值的目标。

注释

获取具有指定名称的 qa_int 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getLongProperty 函数

语法

```
qa_bool QAMessage::getLongProperty(
  qa_const_string name,
  qa_long * value
)
```

参数

- **name** 要获取的属性的名称。
- **value** qa_long 值的目标。

注释

获取具有指定名称的 qa_long 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getMessageID 函数

语法

```
qa_const_string QAMessage::getMessageID()
```

注释

获取消息 ID。

MessageID 标头字段包含一个值，它可唯一地标识 QAnywhere 客户端发送的每条消息。

使用 putMessage 方法发送消息时，MessageID 标头为空值，因此可以被忽略。发送方法返回时，它包含一个指派的值。

MessageID 是一个 qa_string 值，它作为一个唯一键来标识历史存储库中的消息。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

消息 ID。

getPriority 函数

语法

qa_int **QAMessage::getPriority()**

注释

获取消息的优先级。

QAnywhere 客户端 API 定义十种级别的优先级值，0 代表最低优先级，9 代表最高优先级。客户端应将优先级 0-4 视为普通优先级等级，将优先级 5-9 视为加速优先级等级。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

消息优先级。

getPropertyType 函数

语法

```
qa_short QAMessage::getPropertyType(
  qa_const_string name
)
```

参数

- **name** 属性的名称。

注释

返回具有给定名称的属性的类型。

为 PROPERTY_TYPE_BOOLEAN、PROPERTY_TYPE_BYTE、PROPERTY_TYPE_SHORT、PROPERTY_TYPE_INT、PROPERTY_TYPE_LONG、PROPERTY_TYPE_FLOAT、PROPERTY_TYPE_DOUBLE、PROPERTY_TYPE_STRING 和 PROPERTY_TYPE_UNKNOWN 中的一个。

返回值

属性的类型。

getRedelivered 函数

语法

qa_bool **QAMessage::getRedelivered()**

注释

指示消息是否以前已接收但尚未确认。

如果检测到正在被接收的消息以前已接收过，则接收 QAManager 将设置 Redelivered 标头。

例如，应用程序使用以 EXPLICIT_ACKNOWLEDGEMENT 打开的“QAManager 类”一节第 420 页接收消息，并且没有确认此消息就关闭。当应用程序再次启动并接收同一消息时，Redelivered 标头为 true。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

当且仅当重新发送消息时返回 true。

getReplyToAddress 函数

语法

qa_const_string **QAMessage::getReplyToAddress()**

注释

获取对此消息的回复应发送到的地址。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

回复地址。

getShortProperty 函数

语法

```
qa_bool QAMessage::getShortProperty(
  qa_const_string name,
  qa_short * value
)
```


参数

- **name** 要获取的属性的名称。
- **value** qa_short 值的目标。

注释

获取具有指定名称的 qa_short 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

当且仅当操作成功时返回 true。

getStringProperty 函数

语法

```
qa_int QAMessage::getStringProperty(
  qa_const_string name,
  qa_string dest,
  qa_int maxlen
)
```

参数

- **name** 要获取的属性的名称。
- **dest** qa_string 值的目标。
- **maxlen** 要复制值的 qa_char 的最大数量。此值包括空终止符 qa_char。

注释

获取具有指定名称的 qa_string 属性的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

返回实际复制的非空 qa_char 的数量；如果操作失败，则返回 -1。

getStringProperty 函数

语法

```
qa_int QAMessage::getStringProperty(
  qa_const_string name,
  qa_int offset,
  qa_string dest,
  qa_int maxlen
)
```

参数

- **name** 要获取的属性的名称。
- **offset** 要从中复制的属性值的起始偏移。
- **dest** qa_string 值的目标。
- **maxlen** 要复制值的 qa_char 的最大数量。此值包括空终止符 qa_char。

注释

获取具有指定名称的 qa_string 属性的值（从偏移开始）。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

返回值

返回实际复制的非空 qa_char 的数量；如果操作失败，则返回 -1。

getTimestamp 函数

语法

```
qa_long QAMessage::getTimestamp()
```

注释

获取消息时间戳。

此 Timestamp 标头字段包含消息的创建时间。时间为协调通用时间 (UTC)。

它不是消息实际传输的时间，因为事务或其它客户端消息排队可能导致实际发送推迟。它的单位符合平台要求。对于 Windows/PocketPC 平台，时间戳为 SYSTEMTIME，将被转换为 FILETIME 并复制到 qa_long 值中。

要将时间戳 ts 转换为 SYSTEMTIME 以对用户进行显示，请运行以下代码：

```
SYSTEMTIME      stime;
FILETIME        ftime;
```



```
ULARGE_INTEGER  time;
time.QuadPart = ts;
memcpy(&ftime, &time, sizeof(FILETIME));
FileTimeToSystemTime(&ftime, &stime);
```

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

消息时间戳。

getTimestampAsString 函数

语法

qa_int **QAMessage::getTimestampAsString(**
  qa_string *buffer*,
  qa_int *bufferLen*
)

参数

- **buffer** 用于已设置格式的时间戳的缓冲区。
- **bufferLen** 缓冲区大小。

注释

按已设置格式的字符串形式获取消息时间戳。

格式如下："dow, MMM dd, yyyy hh:mm:ss.nnn GMT"。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

返回值

写入到缓冲区的非空 qa_char 的数量。

nextPropertyName 函数

语法

qa_int **QAMessage::nextPropertyName(**
  qa_property_enum_handle *h*,
  qa_string *buffer*,
  qa_int *bufferLen*
)

参数

- **h** beginEnumPropertyNames 返回的句柄。
- **buffer** 要写入属性名称的缓冲区。

- **bufferLen** 存储属性名称的缓冲区的长度。此长度必须包括用于空终止符的空间。

注释

返回给定枚举的消息属性名称；如果没有其它属性名称，则返回 -1。

返回值

属性名称的长度；如果没有其它属性名称，则返回 -1。

propertyExists 函数

语法

```
qa_bool QAMessage::propertyExists(
  qa_const_string name
)
```

参数

- **name** 属性的名称。

注释

指示是否存在属性值。

返回值

当且仅当属性存在时返回 true。

setAddress 函数

语法

```
void QAMessage::setAddress(
  qa_const_string destination
)
```

参数

- **destination** 目标地址。

注释

设置此消息的目标地址。

此方法可用于更改已接收消息的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

setBooleanProperty 函数

语法

```
void QAMessage::setBooleanProperty(
  qa_const_string name,
  qa_bool value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_bool 值。

注释

将具有指定名称的 qa_bool 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

setByteProperty 函数

语法

```
void QAMessage::setByteProperty(
  qa_const_string name,
  qa_byte value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_byte 值。

注释

将具有指定名称的 qa_byte 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页

setDoubleProperty 函数

语法

```
void QAMessage::setDoubleProperty(
  qa_const_string name,
  qa_double value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_double 值。

注释

将具有指定名称的 qa_double 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页.

setFloatProperty 函数

语法

```
void QAMessage::setFloatProperty(
  qa_const_string name,
  qa_float value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_float 值。

注释

将具有指定名称的 qa_float 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页.

setInReplyToID 函数

语法

```
void QAMessage::setInReplyToID(
  qa_const_string id
)
```

参数

- **id** 回复 ID。

注释

设置消息的回复 ID。

客户端可使用 InReplyToID 标头字段，将一条消息与另一条消息相链接。典型应用是将响应消息与其请求消息相链接。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

setIntProperty 函数

语法

```
void QAMessage::setIntProperty(
  qa_const_string name,
  qa_int value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_int 值。

注释

将具有指定名称的 qa_int 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页.

setLongProperty 函数

语法

```
void QAMessage::setLongProperty(
  qa_const_string name,
  qa_long value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_long 值。

注释

将具有指定名称的 qa_long 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页.

setMessageID 函数

语法

```
void QAMessage::setMessageID(
  qa_const_string id
)
```

参数

- **id** 消息 ID。

注释

设置消息 ID。

此方法可用于更改已接收消息的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

setPriority 函数

语法

```
void QAMessage::setPriority(
  qa_int priority
)
```


参数

- **priority** 消息优先级。

注释

设置此消息的优先级级别。

此方法可用于更改已接收消息的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

setRedelivered 函数

语法

```
void QAMessage::setRedelivered(
  qa_bool redelivered
)
```

参数

- **redelivered** 重新发送的指示。

注释

设置此消息是否重新发送的指示。

此方法可用于更改已接收消息的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

setReplyToAddress 函数

语法

```
void QAMessage::setReplyToAddress(
  qa_const_string replyTo
)
```

参数

- **replyTo** 回复地址。

注释

设置对此消息的回复应发送到的地址。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

setShortProperty 函数

语法

```
void QAMessage::setShortProperty(
  qa_const_string name,
  qa_short value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_short 值。

注释

将具有指定名称的 qa_short 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页.

setStringProperty 函数

语法

```
void QAMessage::setStringProperty(
  qa_const_string name,
  qa_const_string value
)
```

参数

- **name** 要设置的属性的名称。
- **value** 属性的 qa_string 值。

注释

将具有指定名称的 qa_string 属性设置为指定值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“MessageProperties 类”一节第 388 页.

setTimestamp 函数

语法

```
void QAMessage::setTimestamp(
  qa_long timestamp
)
```

参数

- **timestamp** 消息时间戳，为协调通用时间 (UTC)。

注释

设置消息时间戳。

此方法可用于更改已接收消息的值。

有关获取和设置消息标头和属性的详细信息，请参见“QAnywhere 消息简介”一节第 14 页。

另请参见

“getTimestamp 函数”一节第 470 页

QAMessageListener 类

语法

public **QAMessageListener**

注释

QAMessageListener 对象用于接收异步发送的消息。

成员

QAMessageListener 的所有成员，包括所有继承的成员。

- “onMessage 函数”一节第 480 页
- “~QAMessageListener 函数”一节第 480 页

onMessage 函数

语法

void **QAMessageListener::onMessage(**
  QAMessage * *message*
)

参数

- **message**　传递给监听器的消息。

注释

向监听器传递一条消息。

~QAMessageListener 函数

语法

virtual **QAMessageListener::~QAMessageListener()**

注释

虚析构函数。

QATextMessage 类

语法

public **QATextMessage**

基类

- “QAMessage 类”一节第 458 页

注释

QATextMessage 继承自 QAMessage 类，并添加了文本消息主体。

QATextMessage 提供读取和写入文本消息主体的方法。

首次创建消息时，消息的主体处于只写模式。消息发送后，发送消息的客户端可保留和修改该消息，而不会影响已发送的消息。可以多次发送同一消息对象。

接收到消息时，提供程序已调用 reset，因此消息主体处于只读模式并且从消息主体的开头开始读取值。如果客户端尝试在只读模式下写入消息，则设置 COMMON_MSG_NOT_WRITEABLE_ERROR。

另请参见

“QABinaryMessage 类”一节第 398 页

成员

QATextMessage 的所有成员，包括所有继承的成员。

- “setShortProperty 函数”一节第 478 页
- “setStringProperty 函数”一节第 478 页
- “setText 函数”一节第 484 页
- “setTimestamp 函数”一节第 479 页
- “writeText 函数”一节第 485 页
- “~QATextMessage 函数”一节第 485 页

getText 函数

语法

qa_string **QATextMessage::getText()**

注释

获取包含此消息数据的字符串。

缺省值为空。

如果消息超出 MAX_IN_MEMORY_MESSAGE_SIZE 属性指定的最大大小，则此函数返回空值。在这种情况下，使用 readText 方法读取文本。

有关 QAManager 属性的详细信息，请参见“QAnywhere Manager 配置属性”一节第 90 页。

返回值

包含消息数据的字符串。

getTextLength 函数

语法

qa_long **QATextMessage::getTextLength()**

注释

返回文本长度。

注意：如果文本长度为非零值，且 getText() 返回 qa_null，则文本无法装入内存，必须使用 readText 分段读取。

返回值

文本长度。

readText 函数

语法

```
qa_int QATextMessage::readText(
  qa_string string,
  qa_int length
)
```

参数

- **string** 文本的目标。
- **length** 要读取到目标缓冲区的 qa_char 的最大数量，包括空终止符。

注释

从当前文本位置将请求长度的文本读取到缓冲区。

返回值

返回读取的非空 qa_char 的实际数量；如果已读取整个文本流，则返回 -1。

reset 函数

语法

```
void QATextMessage::reset()
```

注释

将当前文本位置重新定位为开头。

setText 函数

语法

```
void QATextMessage::setText(
  qa_const_string string
)
```

参数

- **string** 包含要设置的消息数据的字符串。

注释

设置包含此消息数据的字符串。

writeText 函数

语法

void **QATextMessage::writeText(**
  qa_const_string *string*,
  qa_int *offset*,
  qa_int *length*
)

参数

- **string** 要连接的源文本。
- **offset** 从中开始读取的源文本偏移。
- **length** 要读取的源文本的 qa_char 数量。

注释

将文本连接到当前文本。

~QATextMessage 函数

语法

virtual **QATextMessage::~QATextMessage()**

注释

虚析构函数。

QATransactionalManager 类

语法

public **QATransactionalManager**

基类

- “QAManagerBase 类”一节第 425 页

注释

此类是用于事务性消息传递的管理器。

QATransactionalManager 类从 QAManagerBase 派生并且管理事务性 QAnywhere 消息传递操作。

有关派生行为的详细说明，请参见“QAManagerBase 类”一节第 425 页。

QATransactionalManager 只能用于事务性确认。使用 commit 方法提交全部 putMessage 和 getMessage 调用。

有关详细信息，请参见“实现事务性消息传递”一节第 67 页。

另请参见

“QATransactionalManager 类”一节第 486 页.

成员

QATransactionalManager 的所有成员，包括所有继承的成员。

commit 函数

语法

qa_bool **QATransactionalManager::commit()**

注释

提交当前事务并开始新的事务。

此方法提交全部 putMessage() 和 getMessage() 调用。

第一个事务以对 open 方法的调用开始。

另请参见

“QATransactionalManager 类”一节第 486 页

返回值

当且仅当提交操作成功时返回 true。

open 函数

语法

qa_bool **QATransactionalManager::open()**

注释

打开一个 QATransactionalManager 实例。

open 方法必须是创建管理器后第一个调用的方法。

另请参见

“QATransactionalManager 类”一节第 486 页

返回值

当且仅当操作成功时返回 true。

rollback 函数

语法

qa_bool **QATransactionalManager::rollback()**

注释

回退当前事务并开始新的事务。

此方法会回退全部未提交的 putMessage 和 getMessage 调用。

另请参见

“QATransactionalManager 类”一节第 486 页

返回值

当且仅当打开操作成功时返回 true。

~QATransactionalManager 函数

语法

virtual **QATransactionalManager::~QATransactionalManager()**

注释

虚析构函数。

QueueDepthFilter 类

语法

public **QueueDepthFilter**

注释

QAManagerBase 的队列深度方法的 QueueDepthFilter 值。

成员

QueueDepthFilter 的所有成员，包括所有继承的成员。

- “ALL 变量”一节第 490 页
- “INCOMING 变量”一节第 490 页
- “OUTGOING 变量”一节第 491 页

ALL 变量

语法

const qa_short **QueueDepthFilter::ALL**

注释

对进来的消息和外发的消息进行计数。

系统消息和到期消息不包含在队列深度计数中。

INCOMING 变量

语法

const qa_short **QueueDepthFilter::INCOMING**

注释

只对进来的消息进行计数。

进来的消息被定义为其发出方不同于消息存储库的代理 ID 的消息。

LOCAL 变量

语法

const qa_short **QueueDepthFilter::LOCAL**

注释

如果使用 LOCAL 过滤器调用 getQueueDepth 且指定了队列，则它将返回发送到该队列的未接收本地消息数。如果未指定队列，它将返回消息存储库中未接收的本地消息的总数，但不包括系统消息。

OUTGOING 变量

语法

const qa_short **QueueDepthFilter::OUTGOING**

注释

只对外发的消息进行计数。

外发的消息被定义为其发出方是消息存储库的代理 ID，而其目标不是消息存储库的代理 ID 的消息。

StatusCodes 类

语法

public **StatusCodes**

注释

此接口为消息状态定义了一组代码。

成员

StatusCodes 的所有成员，包括所有继承的成员。

CANCELLED 变量

语法

const qa_int **StatusCodes::CANCELLED**

注释

消息已取消。

此代码具有值 40。此代码适用于 STATUS。

EXPIRED 变量

语法

const qa_int **StatusCodes::EXPIRED**

注释

消息已到期，即消息在达到到期时间之前未被接收。

此代码具有值 30。此代码适用于 STATUS。

FINAL 变量

语法

const qa_int **StatusCodes::FINAL**

注释

此常量用于确定消息是否已达到最终状态。

当且仅当消息状态大于此常量时，消息达到最终状态。

此代码具有值 20。此代码适用于 STATUS。

LOCAL 变量

语法

const qa_int **StatusCodes::LOCAL**

注释

此消息被发送到本地消息存储库，没有传输到服务器。

此代码具有值 2。此代码适用于 TRANSMISSION_STATUS。

PENDING 变量

语法

const qa_int **StatusCodes::PENDING**

注释

消息已发送但未被接收。

此代码具有值 1。此代码适用于 STATUS。

RECEIVED 变量

语法

const qa_int **StatusCodes::RECEIVED**

注释

此消息已由接收方接收并确认。

此代码具有值 60。此代码适用于 STATUS。

RECEIVING 变量

语法

const qa_int **StatusCodes::RECEIVING**

注释

消息正处于接收过程中，或者已接收但未确认。

此代码具有值 10。此代码适用于 STATUS。

TRANSMITTED 变量

语法

const qa_int **StatusCodes::TRANSMITTED**

注释

消息已经传输到服务器。

此代码具有值 1。此代码适用于 TRANSMISSION_STATUS。

TRANSMITTING 变量

语法

const qa_int **StatusCodes::TRANSMITTING**

注释

此消息正处于传输到服务器的过程中。

此代码具有值 3。此代码适用于 TRANSMISSION_STATUS。

UNRECEIVABLE 变量

语法

const qa_int **StatusCodes::UNRECEIVABLE**

注释

此消息已标记为无法接收。

消息格式错误或尝试发送时失败次数过多。

此代码具有值 50。此代码适用于 STATUS。

UNTRANSMITTED 变量

语法

const qa_int **StatusCodes::UNTRANSMITTED**

注释

消息未传输到服务器。

此代码具有值 0。此代码适用于 TRANSMISSION_STATUS。

QAnywhere Java API 参考

目录

用于客户端的 QAnywhere Java API

包

- ianywhere.qanywhere.client（用于常规客户端）
- ianywhere.qanywhere.standaloneclient（用于独立客户端）

AcknowledgementMode 接口

语法

public **AcknowledgementMode**

注释

指出 QAnywhere 客户端应用程序应如何确认消息。

隐式和显式确认模式被指派给使用 QAManager.open(short) 方法的 QAManager 实例。

使用隐式确认时，客户端应用程序在收到消息时会进行确认。使用显式确认时，必须调用其中一个 QAManager 确认方法。服务器在客户端之间传播全部状态变化。

另请参见

“QAManager 接口”一节第 535 页

“QATransactionalManager 接口”一节第 601 页

“QAManagerBase 接口”一节第 541 页

成员

ianywhere.qanywhere.client.AcknowledgementMode 的所有成员，其中包括所有继承的成员。

- “EXPLICIT_ACKNOWLEDGEMENT 变量”一节第 498 页
- “IMPLICIT_ACKNOWLEDGEMENT 变量”一节第 499 页
- “TRANSACTIONAL 变量”一节第 499 页

EXPLICIT_ACKNOWLEDGEMENT 变量

语法

final short **AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT**

注释

表示使用其中一个 QAManager 确认方法确认已接收到的消息。

另请参见

“QAManager 接口”一节第 535 页

IMPLICIT_ACKNOWLEDGEMENT 变量

语法

final short **AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT**

注释

表示客户端应用程序在接收到每条消息时都会进行确认。

如果同步接收消息，则返回 QAManagerBase.getMessage(string) 方法时将确认消息。如果异步接收消息，则事件处理方法返回时将确认消息。

另请参见

“getMessage 方法”一节第 551 页

TRANSACTIONAL 变量

语法

final short **AcknowledgementMode.TRANSACTIONAL**

注释

此模式表示消息仅作为进行中事务的一部分进行确认。

此模式自动指派给 QATransactionalManager 实例。

另请参见

“QATransactionalManager 接口”一节第 601 页

MessageProperties 接口

语法

public **MessageProperties**

注释

提供存储标准消息属性名称的字段。

MessageProperties 类提供标准消息属性名称。您可以将 MessageProperties 字段传递给用于获取和设置消息属性的 QAMessage 方法。

```
QAMessage msg = mgr.createTextMessage();
```

以下示例使用 QAMessage.getIntProperty(String) 方法获取与 MessageProperties.MSG_TYPE 相对应的值。MessageType 枚举将整数结果映射到合适的消息类型。

```
int msg_type = t_msg.getIntProperty(MessageProperties.MSG_TYPE);
```

以下示例显示 onSystemMessage(QAMessage) 方法，此方法用于处理 QAnywhere 系统消息。

消息类型可通过 MessageProperties.MSG_TYPE 变量和 QAMessage.getIntProperty(String) 方法来计算。

```
private void onSystemMessage(QAMessage msg) {
    QATextMessage    t_msg;
    int              msg_type;
    String           network_adapters;
    String           network_names;
    String           network_info;

    t_msg = (QATextMessage) msg;
    if (t_msg != null) {
        // Evaluate the message type.
        msg_type = (MessageType)
t_msg.getIntProperty(MessageProperties.MSG_TYPE);

        if (msg_type == MessageType.NETWORK_STATUS_NOTIFICATION) {
            // Handle network status notification.
            network_info = "";
            network_adapters =
t_msg.getStringProperty(MessageProperties.ADAPTERS);
            if (network_adapters != null && network_adapters.length > 0) {
                network_info += network_adapters;
            }
            network_names =
t_msg.getStringProperty(MessageProperties.RASNAMES);
            //...
        }
    }
 }
```

成员

ianywhere.qanywhere.client.MessageProperties 的所有成员，其中包括所有继承的成员。

- “ADAPTER 变量”一节第 501 页
- “ADAPTERS 变量”一节第 501 页
- “DELIVERY_COUNT 变量”一节第 501 页
- “IP 变量”一节第 502 页
- “MAC 变量”一节第 502 页
- “MSG_TYPE 变量”一节第 502 页
- “NETWORK_STATUS 变量”一节第 503 页
- “ORIGINATOR 变量”一节第 503 页
- “RAS 变量”一节第 503 页
- “RASNAMES 变量”一节第 504 页
- “STATUS 变量”一节第 504 页
- “STATUS_TIME 变量”一节第 505 页
- “TRANSMISSION_STATUS 变量”一节第 505 页

ADAPTER 变量

语法

final String **MessageProperties.ADAPTER**

注释

用于 "system" 队列消息，指用来连接 QAnywhere 服务器的网络适配器。

此字段的值为 "ias_Network.Adapter"。

您可以在 QAMessage.getStringProperty(String) 方法中传递 MessageProperties.ADAPTER 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getStringProperty 方法”一节第 586 页

ADAPTERS 变量

语法

final String **MessageProperties.ADAPTERS**

注释

此属性名称是指一个可用于连接 QAnywhere 服务器的网络适配器的分隔列表。

它用于系统队列消息。

您可以在 QAMessage.getStringProperty(String) 方法中传递 MessageProperties.ADAPTERS 以访问相关属性。此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getStringProperty 方法”一节第 586 页

DELIVERY_COUNT 变量

语法

final String **MessageProperties.DELIVERY_COUNT**

注释

此属性名称指目前已尝试传送消息的次数。

IP 变量

语法

final String **MessageProperties.IP**

注释

用于 "system" 队列消息，指用来连接 QAnywhere 服务器的网络适配器的 IP 地址。

此字段的值为 "ias_Network.IP"。

您可以在 QAMessage.getStringProperty(String) 方法中传递 MessageProperties.IP 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getStringProperty 方法”一节第 586 页

MAC 变量

语法

final String **MessageProperties.MAC**

注释

用于 "system" 队列消息，指用来连接 QAnywhere 服务器的网络适配器的 MAC 地址。

此字段的值为 "ias_Network.MAC"。

您可以在 QAMessage.getStringProperty(String) 方法中传递 MessageProperties.MAC 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getStringProperty 方法”一节第 586 页

MSG_TYPE 变量

语法

final String **MessageProperties.MSG_TYPE**

注释

此属性名称是指与 QAnywhere 消息相关的 MessageType 枚举值。

此字段的值为 "ias_MessageType"。

您可以在 QAMessage.getIntProperty(String) 方法中传递 MessageProperties.MSG_TYPE 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getIntProperty 方法”一节第 581 页

NETWORK_STATUS 变量

语法

final String **MessageProperties.NETWORK_STATUS**

注释

此属性名称是指网络连接的状态。

如果网络可访问，则值为 1，否则值为 0。网络状态用于 system 队列消息（如网络状态更改）。

您可以在 QAMessage.getIntProperty(String) 方法中传递 MessageProperties.NETWORK_STATUS 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getIntProperty 方法”一节第 581 页

ORIGINATOR 变量

语法

final String **MessageProperties.ORIGINATOR**

注释

此属性名称是指消息发出方的消息存储库 ID。

RAS 变量

语法

final String **MessageProperties.RAS**

注释

用于 "system" 队列消息，是指用来连接 QAnywhere 服务器的 RAS 条目名。

此字段的值为 "ias_Network.RAS"。

您可以在 QAMessage.getStringProperty(String) 方法中传递 MessageProperties.RAS 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getStringProperty 方法”一节第 586 页

RASNAMES 变量

语法

final String **MessageProperties.RASNAMES**

注释

用于 "system" 队列消息，是指可用来连接 QAnywhere 服务器的 RAS 条目名的分隔列表。

此字段的值为 "ias_RASNames"。

您可以在 QAMessage.getStringProperty(String) 方法中传递 MessageProperties.RASNAMES 以访问相关属性。

此属性是只读属性。

另请参见

“MessageProperties 接口”一节第 499 页

“getStringProperty 方法”一节第 586 页

STATUS 变量

语法

final String **MessageProperties.STATUS**

注释

此属性名称是指消息的当前状态。

另请参见

“StatusCodes 接口”一节第 605 页

STATUS_TIME 变量

语法

final String **MessageProperties.STATUS_TIME**

注释

此属性名称是指消息呈现当前状态的时间。

如果将 MessageProperties.STATUS_TIME 传递到 QAMessage.getProperty 方法，它将返回一个 java.util.Date 实例。

另请参见

“getProperty 方法”一节第 583 页

TRANSMISSION_STATUS 变量

语法

final String **MessageProperties.TRANSMISSION_STATUS**

注释

此属性名称是指消息的当前传输状态。

另请参见

“StatusCodes 接口”一节第 605 页

MessageStoreProperties 接口

语法

public **MessageStoreProperties**

注释

此类为有用的消息存储库属性名称定义常量值。

MessageStoreProperties 类提供标准消息属性名称。您可以将 MessageStoreProperties 字段传递给用于获取和设置预定义或自定义消息存储库属性的 QAManagerBase 方法。

另请参见

“QAManagerBase 接口”一节第 541 页

成员

ianywhere.qanywhere.client.MessageStoreProperties 的所有成员，其中包括所有继承的成员。

- “MAX_DELIVERY_ATTEMPTS 变量”一节第 506 页

MAX_DELIVERY_ATTEMPTS 变量

语法

final String **MessageStoreProperties.MAX_DELIVERY_ATTEMPTS**

注释

此属性名称是指在消息的状态被设置为 StatusCodes.UNRECEIVABLE 之前，在不进行确认的情况下，能够接收消息的最大次数。

另请参见

“UNRECEIVABLE 变量”一节第 609 页

MessageType 接口

语法

public **MessageType**

注释

为消息属性 MessageProperties.MSG_TYPE 定义常量值。

以下示例显示 onSystemMessage(QAMessage) 方法，此方法用于处理 QAnywhere 系统消息。消息类型将与 MessageType.NETWORK_STATUS_NOTIFICATION 进行比较。

```
private void onSystemMessage(QAMessage msg) {
    QATextMessage    t_msg;
    int              msg_type;
    String           network_adapters;
    String           network_names;
    String           network_info;

    t_msg = (QATextMessage) msg;
    if (t_msg != null) {
        // Evaluate message type.
        msg_type = t_msg.getIntProperty(MessageProperties.MSG_TYPE);
        if (msg_type == MessageType.NETWORK_STATUS_NOTIFICATION) {
            // Handle network status notification.
        }
    }
}
```


成员

ianywhere.qanywhere.client.MessageType 的所有成员，其中包括所有继承的成员。

- “NETWORK_STATUS_NOTIFICATION 变量”一节第 507 页
- “PUSH_NOTIFICATION 变量”一节第 507 页
- “REGULAR 变量”一节第 507 页

NETWORK_STATUS_NOTIFICATION 变量

语法

final int **MessageType.NETWORK_STATUS_NOTIFICATION**

注释

标识用于通知 QAnywhere 客户端应用程序网络状态更改的 QAnywhere 系统消息。

网络状态更改适用于接收系统消息的设备。使用 MessageProperties.ADAPTER、MessageProperties.NETWORK 和 MessageProperties.NETWORK_STATUS 字段来标识新的网络状态信息。

PUSH_NOTIFICATION 变量

语法

final int **MessageType.PUSH_NOTIFICATION**

注释

标识用于通知 QAnywhere 客户端应用程序推式通知的 QAnywhere 系统消息。

如果您使用 [要求时] QAnywhere 代理策略，一个典型响应就是调用 QAManagerBase.triggerSendReceive() 方法以接收随中央消息服务器一同等待的消息。

REGULAR 变量

语法

final int **MessageType.REGULAR**

注释

如果不存在任何消息类型属性，则假定消息类型为 REGULAR。

消息系统不会特别处理此类型消息。

PropertyType 接口

语法

public **PropertyType**

注释

QAMessage 属性类型枚举，自然对应 Java 类型。

另请参见

“QAMessage 接口”一节第 575 页

成员

ianywhere.qanywhere.client.PropertyType 的所有成员，其中包括所有继承的成员。

- “PROPERTY_TYPE_BOOLEAN 变量”一节第 508 页
- “PROPERTY_TYPE_BYTE 变量”一节第 508 页
- “PROPERTY_TYPE_DOUBLE 变量”一节第 509 页
- “PROPERTY_TYPE_FLOAT 变量”一节第 509 页
- “PROPERTY_TYPE_INT 变量”一节第 509 页
- “PROPERTY_TYPE_LONG 变量”一节第 509 页
- “PROPERTY_TYPE_SHORT 变量”一节第 509 页
- “PROPERTY_TYPE_STRING 变量”一节第 510 页
- “PROPERTY_TYPE_UNKNOWN 变量”一节第 510 页

PROPERTY_TYPE_BOOLEAN 变量

语法

final short **PropertyType.PROPERTY_TYPE_BOOLEAN**

注释

表示布尔型属性。

PROPERTY_TYPE_BYTE 变量

语法

final short **PropertyType.PROPERTY_TYPE_BYTE**

注释

表示有符号字节型属性。

PROPERTY_TYPE_DOUBLE 变量

语法

final short **PropertyType.PROPERTY_TYPE_DOUBLE**

注释

表示双精度型属性。

PROPERTY_TYPE_FLOAT 变量

语法

final short **PropertyType.PROPERTY_TYPE_FLOAT**

注释

表示浮点型属性。

PROPERTY_TYPE_INT 变量

语法

final short **PropertyType.PROPERTY_TYPE_INT**

注释

表示整型属性。

PROPERTY_TYPE_LONG 变量

语法

final short **PropertyType.PROPERTY_TYPE_LONG**

注释

表示长整型属性。

PROPERTY_TYPE_SHORT 变量

语法

final short **PropertyType.PROPERTY_TYPE_SHORT**

注释

表示短整型属性。

PROPERTY_TYPE_STRING 变量

语法

final short **PropertyType.PROPERTY_TYPE_STRING**

注释

表示字符串型属性。

PROPERTY_TYPE_UNKNOWN 变量

语法

final short **PropertyType.PROPERTY_TYPE_UNKNOWN**

注释

表示未知属性类型，原因通常是属性未知。

QABinaryMessage 接口

语法

public **QABinaryMessage**

基类

- “QAMessage 接口”一节第 575 页

注释

QABinaryMessage 对象用于发送包含未解释字节流的消息。

QABinaryMessage 继承自类 QAMessage，并添加了字节消息主体。QABinaryMessage 提供了读取和写入字节消息主体的各种函数。

首次创建消息时，消息的主体处于只写模式。消息发送后，发送消息的客户端可保留和修改该消息，而不会影响已发送的消息。可以多次发送同一消息对象。

接收到消息时，提供程序已调用 QABinaryMessage.reset()，因此消息主体处于只读模式并且从消息主体的开头开始读取值。

以下示例使用 QABinaryMessage.writeString(String) 将其后是字符串 "Anywhere" 的字符串 "Q" 写入 QABinaryMessage 实例的消息主体。

```
// Create a binary message instance.
QABinaryMessage binary_message;
binary_message = qa_manager.createBinaryMessage();

// Set optional message properties.
binary_message.setReplyToAddress("my-queue-name");
```



```
// Write to the message body.
binary_message.writeString("Q");
binary_message.writeString("Anywhere");

// Put the message in the local database, ready for sending.
try {
    qa_manager.putMessage("store-id\\queue-name", binary_message);
}
catch (QAException e) {
    handleError();
}
```

在接收端，第一次 QABinaryMessage.readString() 调用返回 "Q"，而下一次 QABinaryMessage.readString() 调用则返回 "Anywhere"。

由 QAnywhere 代理发送消息。

另请参见

“QAMessage 接口”一节第 575 页

“readString 方法”一节第 519 页

成员

ianywhere.qanywhere.client.QABinaryMessage 的所有成员，其中包括所有继承的成员。

- “setPriority 方法”一节第 591 页
- “setProperty 方法”一节第 591 页
- “setReplyToAddress 方法”一节第 592 页
- “setShortProperty 方法”一节第 592 页
- “setStringProperty 方法”一节第 593 页
- “writeBinary 方法”一节第 519 页
- “writeBinary 方法”一节第 520 页
- “writeBinary 方法”一节第 520 页
- “writeBoolean 方法”一节第 521 页
- “writeByte 方法”一节第 521 页
- “writeChar 方法”一节第 522 页
- “writeDouble 方法”一节第 522 页
- “writeFloat 方法”一节第 523 页
- “writeInt 方法”一节第 523 页
- “writeLong 方法”一节第 524 页
- “writeShort 方法”一节第 524 页
- “writeString 方法”一节第 525 页

getBodyLength 方法

语法

long **QABinaryMessage.getBodyLength()**
throws **QAException**

抛出条件

- 如果在检索消息主体大小时存在问题，则抛出。

注释

返回消息主体的大小（以字节为单位）.

返回值

消息主体的大小（以字节为单位）。

readBinary 方法

语法

int **QABinaryMessage.readBinary(**
  byte[] *dest*
)
throws **QAException**

参数

- **dest**　保存读取字节的字节数组。

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节。

另请参见

“writeBinary 方法”一节第 519 页

返回值

从消息主体读取的字节数。

readBinary 方法

语法

```
int QABinaryMessage.readBinary(
  byte[] dest,
  int length
)
throws QAException
```

参数

- **dest** 保存读取字节的字节数组。
- **length** 要读取的最大字节数。

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节。

另请参见

“writeBinary 方法”一节第 519 页

返回值

从消息主体读取的字节数。

readBinary 方法

语法

```
int QABinaryMessage.readBinary(
  byte[] dest,
```



```
  int offset,
  int length
)
throws QAException
```

参数

- **dest** 保存读取字节的字节数组。
- **offset** 目标数组的起始偏移。
- **length** 要读取的最大字节数。

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 实例主体的未读部分开始读取指定数量的字节并将其存储到从 dest[offset] 开始的数组目标中。

另请参见

“writeBinary 方法”一节第 519 页

返回值

从消息主体读取的字节数。

readBoolean 方法

语法

```
boolean QABinaryMessage.readBoolean()
throws QAException
```

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 实例的消息主体的未读部分开始读取布尔值。

另请参见

“writeBoolean 方法”一节第 521 页

返回值

从消息主体读取的布尔值。

readByte 方法

语法

byte **QABinaryMessage.readByte()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取有符号字节值。

另请参见

“writeByte 方法”一节第 521 页

返回值

从消息主体读取的有符号字节值。

readChar 方法

语法

char **QABinaryMessage.readChar()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取字符值。

另请参见

“writeChar 方法”一节第 522 页

返回值

从消息主体读取的字符值。

readDouble 方法

语法

double **QABinaryMessage.readDouble()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取双精度值。

另请参见

“writeDouble 方法”一节第 522 页

返回值

从消息主体读取的双精度值。

readFloat 方法

语法

float **QABinaryMessage.readFloat()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取浮点值。

另请参见

“writeFloat 方法”一节第 523 页

返回值

从消息主体读取的浮点值。

readInt 方法

语法

int **QABinaryMessage.readInt()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取整数值。

另请参见

“writeInt 方法”一节第 523 页

返回值

从消息主体读取的整型值。

readLong 方法

语法

long **QABinaryMessage.readLong()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取长整型值。

另请参见

“writeLong 方法”一节第 524 页

返回值

从消息主体读取的长整型值。

readShort 方法

语法

short **QABinaryMessage.readShort()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取短整型值。

另请参见

“writeShort 方法”一节第 524 页

返回值

从消息主体读取的短整型值。

readString 方法

语法

String **QABinaryMessage.readString()**
throws **QAException**

抛出条件

- 如果读取值时发生转换错误或没有其它输入，则抛出。

注释

从 QABinaryMessage 消息主体的未读部分开始读取字符串值。

另请参见

“writeString 方法”一节第 525 页

返回值

从消息主体读取的字符串值。

reset 方法

语法

void **QABinaryMessage.reset()**
throws **QAException**

抛出条件

- 如果重置消息时存在问题，则抛出。

注释

重置消息以便从消息主体的开头开始读取值。

reset 方法还将 QABinaryMessage 消息主体置于只读模式。

writeBinary 方法

语法

void **QABinaryMessage.writeBinary(**
byte[] *val*
)
throws **QAException**

参数

- **val** 写入消息主体的字节数组值。

抛出条件

- 如果将字节数组附加到消息主体时存在问题，则抛出。

注释

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

另请参见

“readBinary 方法”一节第 513 页

writeBinary 方法

语法

```
void QABinaryMessage.writeBinary(
  byte[] val,
  int len
)
throws QAException
```

参数

- **val** 写入消息主体的字节数组值。
- **len** 要写入的字节数。

抛出条件

- 如果将字节数组附加到消息主体时存在问题，则抛出。

注释

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

另请参见

“readBinary 方法”一节第 513 页

writeBinary 方法

语法

```
void QABinaryMessage.writeBinary(
  byte[] val,
  int offset,
  int len
)
throws QAException
```

参数

- **val** 写入消息主体的字节数组值。

- **offset** 要开始写入的字节数组内的偏移。
- **len** 要写入的字节数。

抛出条件

- 如果将字节数组附加到消息主体时存在问题，则抛出。

注释

将字节数组值附加到 QABinaryMessage 实例的消息主体中。

另请参见

“readBinary 方法”一节第 513 页

writeBoolean 方法

语法

```
void QABinaryMessage.writeBoolean(
  boolean val
)
throws QAException
```

参数

- **val** 要写入消息主体的布尔值。

抛出条件

- 如果将布尔值附加到消息主体时存在问题，则抛出。

注释

将布尔值附加到 QABinaryMessage 实例的消息主体中。

布尔值由一个单字节值表示。1 代表 true；0 代表 false。

另请参见

“readBoolean 方法”一节第 515 页

writeByte 方法

语法

```
void QABinaryMessage.writeByte(
  byte val
)
throws QAException
```

参数

- **val** 要写入消息主体的有符号字节值。

抛出条件

- 如果将有符号字节值附加到消息主体时存在问题，则抛出。

注释

将有符号字节值附加到 QABinaryMessage 实例的消息主体中。

有符号字节由一个单字节值表示。

另请参见

“readByte 方法”一节第 516 页

writeChar 方法

语法

```
void QABinaryMessage.writeChar(
  char val
)
throws QAException
```

参数

- **val** 要写入消息主体的字符值。

抛出条件

- 如果将字符值附加到消息主体时存在问题，则抛出。

注释

将字符值附加到 QABinaryMessage 实例的消息主体中。

字符由一个双字节值表示，首先附加高位字节。

另请参见

“readChar 方法”一节第 516 页

writeDouble 方法

语法

```
void QABinaryMessage.writeDouble(
  double val
)
throws QAException
```


参数

- **val** 要写入消息主体的双精度值。

抛出条件

- 如果将双精度值附加到消息主体时存在问题，则抛出。

注释

将双精度值附加到 QABinaryMessage 实例的消息主体中。

双精度值可转换为 8 字节长整型的表示形式，首先附加高位字节。

另请参见

“readDouble 方法”一节第 516 页

writeFloat 方法

语法

```
void QABinaryMessage.writeFloat(
  float val
)
throws QAException
```

参数

- **val** 要写入消息主体的浮点值。

抛出条件

- 如果将浮点值附加到消息主体时存在问题，则抛出。

注释

将浮点值附加到 QABinaryMessage 实例的消息主体中。

浮点值可转换为 4 字节整型表示形式，首先附加高位字节。

另请参见

“readFloat 方法”一节第 517 页

writeInt 方法

语法

```
void QABinaryMessage.writeInt(
  int val
)
throws QAException
```

参数

- **val** 要写入消息主体的整型值。

抛出条件

- 如果将整数值附加到消息主体时存在问题，则抛出。

注释

将整数值附加到 QABinaryMessage 实例的消息主体中。

整数参数由一个 4 字节值表示，首先附加高位字节。

另请参见

“readInt 方法”一节第 517 页

writeLong 方法

语法

```
void QABinaryMessage.writeLong(
  long val
)
throws QAException
```

参数

- **val** 要写入消息主体的长整型值。

抛出条件

- 如果将长整型值附加到消息主体时存在问题，则抛出。

注释

将长整型值附加到 QABinaryMessage 实例的消息主体中。

长整型参数用 8 字节值表示，首先附加高位字节。

另请参见

“readLong 方法”一节第 518 页

writeShort 方法

语法

```
void QABinaryMessage.writeShort(
  short val
)
throws QAException
```


参数

- **val** 要写入消息主体的短整型值。

抛出条件

- 如果将短整型值附加到消息主体时存在问题，则抛出。

注释

将短整型值附加到 QABinaryMessage 实例的消息主体中。

短整型参数由一个双字节值表示，首先附加高位字节。

另请参见

“readShort 方法”一节第 518 页

writeString 方法

语法

```
void QABinaryMessage.writeString(
  String val
)
throws QAException
```

参数

- **val** 要写入消息主体的字符串值。

抛出条件

- 如果将字符串值附加到消息主体时存在问题，则抛出。

注释

将字符串值附加到 QABinaryMessage 实例的消息主体中。

注意：接收应用程序需要为每个 writeString 调用而调用 QABinaryMessage.readString。

注意：要写入的字符串的 UTF-8 表示形式最多可为 32767 个字节。

另请参见

“readString 方法”一节第 519 页

QAException 类

语法

```
public QAException
```

注释

封装 QAnywhere 客户端应用程序异常。

您可以使用 QAException 类捕获 QAnywhere 异常。

```
try {
    _qaManager = QAManagerFactory.getInstance().CreateQAManager();
    _qaManager.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
    _qaManager.start();
}
catch(QAException e) {
    // Handle exception.
    System.err.println("Error code: " + e.getErrorCode());
    System.err.println("Error message: " + e.getMessage());
    System.err.println("Native error code: " + e.getNativeErrorCode());
    System.err.println("Detailed error message: " + e.getDetailedMessage());
}
```

成员

ianywhere.qanywhere.client.QAException 的所有成员，其中包括所有继承的成员。

COMMON_ALREADY_OPEN_ERROR 变量

语法

final int **QAException.COMMON_ALREADY_OPEN_ERROR**

注释

QAManager 已打开。

另请参见

“QAManager 接口”一节第 535 页

COMMON_GETQUEUEDEPTH_ERROR 变量

语法

final int **QAException.COMMON_GETQUEUEDEPTH_ERROR**

注释

获取队列深度时出错。

另请参见

“getQueueDepth 方法”一节第 556 页

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 变量

语法

final int **QAException.COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG**

注释

当过滤器为 ALL 时，无法在给定目标上使用 QAManagerBase.getQueueDepth。

另请参见

“getQueueDepth 方法”一节第 556 页

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 变量

语法

final int **QAException.COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID**

注释

在未设置消息存储库 ID 时，无法使用 QAManagerBase.getQueueDepth。

另请参见

“getQueueDepth 方法”一节第 556 页

COMMON_GET_INIT_FILE_ERROR 变量

语法

final int **QAException.COMMON_GET_INIT_FILE_ERROR**

注释

无法访问客户端属性文件。

COMMON_GET_PROPERTY_ERROR 变量

语法

final int **QAException.COMMON_GET_PROPERTY_ERROR**

注释

从消息存储库中检索属性时出错。

COMMON_INIT_ERROR 变量

语法

final int **QAException.COMMON_INIT_ERROR**

注释

初始化错误。

COMMON_INIT_THREAD_ERROR 变量

语法

final int **QAException.COMMON_INIT_THREAD_ERROR**

注释

初始化后台线程时出错。

COMMON_INVALID_PROPERTY 变量

语法

final int **QAException.COMMON_INVALID_PROPERTY**

注释

客户端属性文件中有一个无效的属性。

COMMON_MSG_ACKNOWLEDGE_ERROR 变量

语法

final int **QAException.COMMON_MSG_ACKNOWLEDGE_ERROR**

注释

确认消息时出错。

COMMON_MSG_CANCEL_ERROR 变量

语法

final int **QAException.COMMON_MSG_CANCEL_ERROR**

注释

取消消息时出错。

COMMON_MSG_CANCEL_ERROR_SENT 变量

语法

final int **QAException.COMMON_MSG_CANCEL_ERROR_SENT**

注释

取消消息时出错。

不能取消已发送的消息。

COMMON_MSG_NOT_WRITEABLE_ERROR 变量

语法

final int **QAException.COMMON_MSG_NOT_WRITEABLE_ERROR**

注释

不能写入到处于只读模式的消息。

COMMON_MSG_RETRIEVE_ERROR 变量

语法

final int **QAException.COMMON_MSG_RETRIEVE_ERROR**

注释

从客户端消息存储库中检索消息时出错。

COMMON_MSG_STORE_ERROR 变量

语法

final int **QAException.COMMON_MSG_STORE_ERROR**

注释

在客户端消息存储库中存储消息时出错。

COMMON_MSG_STORE_NOT_INITIALIZED 变量

语法

final int **QAException.COMMON_MSG_STORE_NOT_INITIALIZED**

注释

尚未为进行消息传递而初始化消息存储库。

COMMON_MSG_STORE_TOO_LARGE 变量

语法

final int **QAException.COMMON_MSG_STORE_TOO_LARGE**

注释

消息存储库相对于设备上的可用磁盘空间过大。

COMMON_NOT_OPEN_ERROR 变量

语法

final int **QAException.COMMON_NOT_OPEN_ERROR**

注释

QAManager 没有打开。

另请参见

“QAManager 接口”一节第 535 页

COMMON_NO_DEST_ERROR 变量

语法

final int **QAException.COMMON_NO_DEST_ERROR**

注释

没有目标。

COMMON_NO_IMPLEMENTATION 变量

语法

final int **QAException.COMMON_NO_IMPLEMENTATION**

注释

该方法未实现。

COMMON_OPEN_ERROR 变量

语法

final int **QAException.COMMON_OPEN_ERROR**

注释

打开消息存储库连接时出错。

COMMON_OPEN_LOG_FILE_ERROR 变量

语法

final int **QAException.COMMON_OPEN_LOG_FILE_ERROR**

注释

打开日志文件时出错。

COMMON_OPEN_MAXTHREADS_ERROR 变量

语法

final int **QAException.COMMON_OPEN_MAXTHREADS_ERROR**

注释

无法打开 QAManager，因为并发服务器请求的最大数不够大。请参见“-gn 服务器选项”一节《SQL Anywhere 服务器 - 数据库管理》。

COMMON_SELECTOR_SYNTAX_ERROR 变量

语法

final int **QAException.COMMON_SELECTOR_SYNTAX_ERROR**

注释

给定选择程序存在一个语法错误。

COMMON_SET_PROPERTY_ERROR 变量

语法

final int **QAException.COMMON_SET_PROPERTY_ERROR**

注释

将属性存储到消息存储库中时出错。

COMMON_TERMINATE_ERROR 变量

语法

final int **QAException.COMMON_TERMINATE_ERROR**

注释

终止错误。

COMMON_UNEXPECTED_EOM_ERROR 变量

语法

final int **QAException.COMMON_UNEXPECTED_EOM_ERROR**

注释

遇到了意外的消息结尾。

COMMON_UNREPRESENTABLE_TIMESTAMP 变量

语法

final int **QAException.COMMON_UNREPRESENTABLE_TIMESTAMP**

注释

时间戳在可接受范围以外。

getDetailedMessage 方法

语法

string **QAException.getDetailedMessage()**

注释

返回对异常的详细说明。

返回值

异常的错误消息。

getErrorCode 方法

语法

int **QAException.getErrorCode()**

注释

返回异常的错误代码。

返回值

异常的错误代码。

getNativeErrorCode 方法

语法

int **QAException.getNativeErrorCode()**

注释

返回异常的本地错误代码。

返回值

异常的本地错误代码。

QA_NO_ERROR 变量

语法

final int **QAException.QA_NO_ERROR**

注释

无错误。

QAException 方法

语法

```
QAException.QAException(
 String message,
 int errorCode
)
```

参数

- **message** 异常的文本描述。
- **errorCode** 错误代码。

注释

利用提供的错误代码和错误消息文本创建一个 QAException 实例。

QAManager 接口

语法

public **QAManager**

基类

- “QAManagerBase 接口”一节第 541 页

注释

QAManager 管理非事务性的 QAnywhere 消息传递操作。

它由 QAManagerBase 派生。

有关派生行为的详细说明，请参见“QAManagerBase 接口”一节第 541 页。

可以如 AcknowledgementMode 类中定义的那样对 QAManager 实例进行配置，以进行隐式或显式确认。要将消息作为事务的一部分进行确认，请使用 QATransactionalManager。

使用 QAManagerFactory 类创建 QAManager 和 QATransactionalManager 对象。

另请参见

“AcknowledgementMode 接口”一节第 498 页

“QAManagerFactory 类”一节第 571 页

“QATransactionalManager 接口”一节第 601 页

成员

ianywhere.qanywhere.client.QAManager 的所有成员，其中包括所有继承的成员。

- “setShortStoreProperty 方法”一节第 568 页
- “setStoreProperty 方法”一节第 568 页
- “setStringStoreProperty 方法”一节第 569 页
- “start 方法”一节第 570 页
- “stop 方法”一节第 570 页
- “triggerSendReceive 方法”一节第 570 页

acknowledge 方法

语法

```
void QAManager.acknowledge(
  QAMessage msg
)
throws QAException
```

参数

- **msg** 要确认的消息。

抛出条件

- 如果确认消息时存在问题，则抛出。

注释

确认客户端应用程序成功接收了一条 QAnywhere 消息。

当 QAMessage 得到确认后，它的状态属性会更改为 StatusCodes.RECEIVED。这时可以使用缺省删除规则将其删除。

另请参见

“RECEIVED 变量”一节第 608 页

“acknowledgeUntil 方法”一节第 539 页

“acknowledgeAll 方法”一节第 538 页

acknowledgeAll 方法

语法

```
void QAManager.acknowledgeAll()
throws QAException
```

抛出条件

- 如果确认消息时存在问题，则抛出。

注释

确认客户端应用程序成功接收了 QAnywhere 消息。

确认所有未确认的消息。

当 QAMessage 得到确认后，它的状态属性会更改为 StatusCodes.RECEIVED。这时可以使用缺省删除规则将其删除。

另请参见

“RECEIVED 变量”一节第 608 页

“acknowledge 方法”一节第 538 页

“acknowledgeUntil 方法”一节第 539 页

acknowledgeUntil 方法

语法

```
void QAManager.acknowledgeUntil(
  QAMessage msg
)
throws QAException
```

参数

- **msg** 要确认的最后一个消息。还确认所有较早的未确认消息。

抛出条件

- 如果确认消息时存在问题，则抛出。

注释

对在给定消息之前收到的给定 QAMessage 实例和所有未确认消息进行确认。

当 QAMessage 得到确认后，它的状态属性会更改为 StatusCodes.RECEIVED。这时可以使用缺省删除规则将其删除。

另请参见

“QAMessage 接口”一节第 575 页

“RECEIVED 变量”一节第 608 页

“acknowledge 方法”一节第 538 页

“acknowledgeAll 方法”一节第 538 页

open 方法

语法

```
void QAManager.open(
  short mode
)
throws QAException
```

参数

- **mode** 确认模式，AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 或 AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT 中的一个。

抛出条件

- 如果打开 QAManager 实例时存在问题，则抛出。

注释

使用给定的 AcknowledgementMode 值打开 QAManager。

open 方法必须是创建 QAManager 后调用的第一个方法。

如果检测到数据库连接错误，则可以通过调用 close 方法随后调用 open 方法来重新打开 QAManager。重新打开 QAManager 时，不需要重新创建它、重置属性或重置消息监听器。第一次打开后便无法更改 QAManager 的属性，后续的打开调用必须提供相同的确认模式。

另请参见

“AcknowledgementMode 接口”一节第 498 页

“EXPLICIT_ACKNOWLEDGEMENT 变量”一节第 498 页

“IMPLICIT_ACKNOWLEDGEMENT 变量”一节第 499 页

“close 方法”一节第 546 页

recover 方法

语法

```
void QAManager.recover()
throws QAException
```

抛出条件

- 如果恢复时存在问题，则抛出。

注释

强制将所有未确认的消息转为未接收状态。

必须使用 QAManagerBase.getMessage(String) 再次接收这些消息。

另请参见

“getMessage 方法”一节第 551 页

QAManagerBase 接口

语法

public **QAManagerBase**

派生类

- “QAManager 接口”一节第 535 页
- “QATransactionalManager 接口”一节第 601 页

注释

此类用作 QATransactionalManager 和 QAManager 的基类，它们分别管理事务性和非事务性消息传递。

使用 QAManagerBase.start() 方法可允许 QAManagerBase 实例监听消息。QAManagerBase 实例只能由创建它的线程使用。

您可以使用此类的实例来创建和管理 QAnywhere 消息。使用 QAManagerBase.createBinaryMessage() 和 QAManagerBase.createTextMessage() 方法来创建合适的 QAMessage 实例。QAMessage 实例提供设置消息内容和属性的各种方法。要发送 QAnywhere 消息，可使用 QAManagerBase.putMessage(String, QAMessage) 方法，将已指定地址的消息放置在本地消息存储库队列中。此消息由 QAnywhere 代理根据其传输策略传输，或在调用 QAManagerBase.triggerSendReceive() 时传输。

QAManagerBase 还提供设置和获取消息存储库属性的方法。

另请参见

“QATransactionalManager 接口”一节第 601 页

“QAManager 接口”一节第 535 页

成员

ianywhere.qanywhere.client.QAManagerBase 的所有成员，其中包括所有继承的成员。

- “start 方法”一节第 570 页
- “stop 方法”一节第 570 页
- “triggerSendReceive 方法”一节第 570 页

browseMessages 方法

语法

java.util.Enumeration **QAManagerBase.browseMessages()**
throws **QAException**

抛出条件

- 如果浏览消息时存在问题，则抛出。

注释

浏览消息存储库中的全部可用消息。

正在浏览消息，因此无法对这些消息进行确认。

使用 QAManagerBase.getMessage(String) 方法接收消息，以便可以对这些消息进行确认。

另请参见

“browseMessagesByQueue 方法”一节第 544 页

“browseMessagesByID 方法”一节第 543 页

“getMessage 方法”一节第 551 页

返回值

可用消息的枚举器。

browseMessagesByID 方法

语法

java.util.Enumeration **QAManagerBase.browseMessagesByID(**
String *id*
)
throws **QAException**

参数

- **id** 消息的消息 ID。

抛出条件

- 如果浏览消息时存在问题，则抛出。

注释

浏览具有给定消息 ID 的消息。

正在浏览消息，因此无法对其进行确认。使用 QAManagerBase.getMessage(String) 接收消息，以便可以对这些消息进行确认。

另请参见

“browseMessagesByQueue 方法”一节第 544 页

“browseMessages 方法”一节第 543 页

“getMessage 方法”一节第 551 页

返回值

包含 0 或 1 消息的枚举器。

browseMessagesByQueue 方法

语法

```
java.util.Enumeration QAManagerBase.browseMessagesByQueue(
  String address
)
throws QAException
```

参数

- **address** 消息的地址。

抛出条件

- 如果浏览消息时存在问题，则抛出。

注释

浏览正在等待的已发送到给定地址的可用消息。

正在浏览消息，因此无法对这些消息进行确认。

使用 QAManagerBase.getMessage(String) 方法接收消息，以便可以对这些消息进行确认。

另请参见

“browseMessagesByID 方法”一节第 543 页

“browseMessages 方法”一节第 543 页

“getMessage 方法”一节第 551 页

返回值

可用消息的枚举器。

browseMessagesBySelector 方法

语法

java.util.Enumeration **QAManagerBase.browseMessagesBySelector(**
  String *selector*
)
throws **QAException**

参数

- **selector** 选择程序。

抛出条件

- 如果浏览消息时存在问题，则抛出。

注释

浏览消息存储库中满足给定选择程序的排队消息。

正在浏览消息，因此无法对其进行确认。使用 QAManagerBase.getMessage(String) 接收消息，以便可以对这些消息进行确认。

另请参见

“browseMessagesByQueue 方法”一节第 544 页

“browseMessages 方法”一节第 543 页

“browseMessagesByID 方法”一节第 543 页

“getMessage 方法”一节第 551 页

返回值

可用消息的枚举器。

cancelMessage 方法

语法

boolean **QAManagerBase.cancelMessage(**
  String *id*
)
throws **QAException**

参数

- **id** 要取消的消息的消息 ID。

抛出条件

- 如果取消消息时存在问题，则抛出。

注释

取消具有给定消息 ID 的消息。

在消息传输前将其置于已取消状态。

当使用 QAnywhere 代理的缺省删除规则时，已取消的消息最终会从消息存储库中删除。

如果消息已经处于最终状态，或已经将消息传输到中央消息传递服务器，此操作将失败。

close 方法

语法

void **QAManagerBase.close()**
throws **QAException**

抛出条件

- 如果关闭 QAManagerBase 实例时存在问题，则抛出。

注释

关闭到 QAnywhere 消息系统的连接并释放 QAManagerBase 使用的所有资源。

除对 close() 的第一次调用外，其余对 close() 的调用都将被忽略。随后的任何对 QAManagerBase 方法（close() 除外）的调用都会导致出现 QAException。在这种情况下，您必须创建并打开一个新的 QAManagerBase 实例。

如果检测到数据库连接错误，则可以通过调用 close 函数并随后调用 open 函数来重新打开 QAManager。重新打开 QAManager 时，不需要重新创建它、重置属性或重置消息监听器。第一次打开后便无法更改 QAManager 的属性，后续的打开调用必须提供相同的确认模式。

另请参见

- “open 方法”一节第 540 页

createBinaryMessage 方法

语法

QABinaryMessage **QAManagerBase.createBinaryMessage()**
throws **QAException**

抛出条件

- 如果创建消息时存在问题，则抛出。

注释

创建一个 QABinaryMessage 对象。

QABinaryMessage 对象用于发送包含未解释字节的消息主体的消息。

另请参见

“QABinaryMessage 接口”一节第 510 页

返回值

一个新的 QABinaryMessage 实例。

createTextMessage 方法

语法

QATextMessage **QAManagerBase.createTextMessage()**
throws **QAException**

抛出条件

- 如果创建消息时存在问题，则抛出。

注释

创建一个 QATextMessage 对象。

QATextMessage 对象用于发送包含字符串消息主体的消息。

另请参见

“QATextMessage 接口”一节第 596 页

返回值

一个新的 QATextMessage 实例。

getBooleanStoreProperty 方法

语法

boolean **QAManagerBase.getBooleanStoreProperty(**
String *name*
)
throws **QAException**

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的布尔值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

布尔属性值。

getByteStoreProperty method

语法

```
byte QAManagerBase.getByteStoreProperty(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的有符号字节值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

有符号字节属性值。

getDoubleStoreProperty 方法

语法

```
double QAManagerBase.getDoubleStoreProperty(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的双精度值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

双精度属性值。

getFloatStoreProperty 方法

语法

```
float QAManagerBase.getFloatStoreProperty(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的浮点值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

浮点属性值。

getIntStoreProperty 方法

语法

```
int QAManagerBase.getIntStoreProperty(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的整型值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

整型属性值。

getLongStoreProperty 方法

语法

```
long QAManagerBase.getLongStoreProperty(
  String name
)
throws QAException
```


参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的长整型值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

长整型属性值。

getMessage 方法

语法

```
QAMessage QAManagerBase.getMessage(
  String address
)
throws QAException
```

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。

抛出条件

- 如果获取消息时存在问题，则抛出。

注释

返回下一个发送到指定地址的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此调用将无限期阻塞直到有可用消息。使用此方法同步接收消息。

另请参见

“QAMessage 接口”一节第 575 页

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMessageBySelector 方法

语法

QAMessage **QAManagerBase.getMessageBySelector(**
  String *address*,
  String *selector*
)
throws **QAException**

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。
- **selector** 选择程序。

抛出条件

- 如果获取消息时存在问题，则抛出。

注释

返回下一个发送到指定地址并满足给定选择程序的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此调用将无限期阻塞直到有可用消息。

使用此方法同步接收消息。

另请参见

“QAMessage 接口”一节第 575 页

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMessageBySelectorNoWait 方法

语法

QAMessage **QAManagerBase.getMessageBySelectorNoWait(**
  String *address*,
  String *selector*
)
throws **QAException**

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。
- **selector** 选择程序。

抛出条件

- 如果获取消息时存在问题，则抛出。

注释

返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此方法立即返回。

使用此方法同步接收消息。

另请参见

“QAMessage 接口”一节第 575 页

返回值

返回下一个可用 QAMessage；如果没有可用消息，则返回空值。

getMessageBySelectorTimeout 方法

语法

```
QAMessage QAManagerBase.getMessageBySelectorTimeout(
  String address,
  String selector,
  long timeout
)
throws QAException
```

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。
- **selector** 选择程序。
- **timeout** 等待消息可用的时间（以毫秒为单位）。

抛出条件

- 如果获取消息时存在问题，则抛出。

注释

返回下一个发送到给定地址并满足给定选择程序的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，此方法等待指定的超时时间，然后返回。

使用此方法同步接收消息。

另请参见

“QAMessage 接口”一节第 575 页

返回值

返回下一条可用 QAMessage；如果没有可用消息，则返回空值。

getMessageNoWait 方法

语法

```
QAMessage QAManagerBase.getMessageNoWait(
  String address
)
throws QAException
```

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。

抛出条件

- 如果获取消息时存在问题，则抛出。

注释

返回下一个发送到给定地址的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，则此方法立即返回。

使用此方法同步接收消息。

另请参见

“QAMessage 接口”一节第 575 页

返回值

返回下一个可用 QAMessage；如果没有可用消息，则返回空值。

getMessageTimeout 方法

语法

```
QAMessage QAManagerBase.getMessageTimeout(
  String address,
  long timeout
)
throws QAException
```

参数

- **address** 此地址指定 QAnywhere 客户端用于接收消息的队列名称。
- **timeout** 等待消息可用的时间（以毫秒为单位）。

抛出条件

- 如果获取消息时存在问题，则抛出。

注释

返回下一个发送到给定地址的可用 QAMessage。

address 参数指定本地队列名称。地址可以采用 'store-id\queue-name' 或 'queue-name' 的形式。如果没有可用消息，此方法等待指定的超时时间，然后返回。使用此方法同步接收消息。

另请参见

“QAMessage 接口”一节第 575 页

返回值

返回下一条 QAMessage，如果没有可用消息，则返回空值。

getMode 方法

语法

short **QAManagerBase.getMode()**
throws **QAException**

抛出条件

- 如果检索 QAManager 确认模式时存在问题，则抛出。

注释

返回所接收消息的 QAManager 确认模式。

有关返回值的列表，请参见“AcknowledgementMode 接口”一节第 498 页。

AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 和 AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT 适用于 QAManager 实例。AcknowledgementMode.TRANSACTIONAL 是适用于 QATransactionalManager 实例的模式。

另请参见

“EXPLICIT_ACKNOWLEDGEMENT 变量”一节第 498 页

“IMPLICIT_ACKNOWLEDGEMENT 变量”一节第 499 页

“QAManager 接口”一节第 535 页

“QATransactionalManager 接口”一节第 601 页

返回值

所接收消息的 QAManager 确认模式。

getQueueDepth 方法

基于给定过滤器，返回所有队列的深度总和。

语法

```
int QAManagerBase.getQueueDepth(
  short filter
)
```

参数

- **filter** filter 指示进来的消息、外发的消息或全部消息。

抛出条件

- “QAException 类”一节第 525 页.

注释

基于给定过滤器，返回所有队列的深度总和。

队列的深度是指没有被接收（如使用 QAManagerBase.getMessage 方法）的消息的数量，其中包括了未提交的外发消息。

有关可能的过滤器值的列表，请参见“QueueDepthFilter 接口”一节第 604 页。

另请参见

“getMessage 方法”一节第 551 页

返回值

INTEGER - 给定过滤器的所有队列中消息的数量。

getQueueDepth 方法

基于给定过滤器，返回所有队列的深度总和。

语法

```
int QAManagerBase.getQueueDepth(
  String queue,
  short filter
)
```

参数

- **queue** 队列名。
- **filter** filter 指示进来的消息、外发的消息或全部消息。

抛出条件

- “QAException 类”一节第 525 页.

注释

基于给定过滤器，返回队列深度。

队列的深度是指没有被接收（如使用 QAManagerBase.getMessage 方法）的消息的数量，其中包括了未提交的外发消息。

有关可能的过滤器值的列表，请参见“QueueDepthFilter 接口”一节第 604 页。

返回值

INTEGER - 未接收到的消息的数量。

getShortStoreProperty 方法

语法

```
short QAManagerBase.getShortStoreProperty(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取预定义或自定义消息存储库属性的短整型值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

短整型属性值。

getStoreProperty 方法

语法

```
Object QAManagerBase.getStoreProperty(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取表示消息存储库属性的对象。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

返回值

属性值。

getStorePropertyNames 方法

语法

java.util.Enumeration **QAManagerBase.getStorePropertyNames()**
throws **QAException**

抛出条件

- 如果检索枚举器时存在问题，则抛出。

注释

获取消息存储库属性名称的枚举器。

返回值

消息存储库属性名称的枚举器。

getStringStoreProperty 方法

语法

String **QAManagerBase.getStringStoreProperty(**
  String *name*
)
throws **QAException**

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果检索字符串值时存在问题，则抛出。

注释

获取预定义或自定义消息存储库属性的字符串值。

可以使用此方法访问预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

返回值

返回字符串属性值；如果属性不存在，则返回空值。

另请参见

“MessageStoreProperties 接口”一节第 505 页

propertyExists 方法

语法

```
boolean QAManagerBase.propertyExists(
  String name
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。

抛出条件

- 如果检索属性值时存在问题，则抛出。

注释

测试给定属性是否有值。

可以使用该方法确定给定的属性名当前是否有通过消息存储库映射到它的值。

另请参见

“MessageStoreProperties 接口”一节第 505 页

putMessage 方法

语法

```
void QAManagerBase.putMessage(
  String address,
  QAMessage msg
)
throws QAException
```

参数

- **address** 指定目标队列名称的消息的地址。
- **msg** 放入本地消息存储库以进行传输的消息。

抛出条件

- 如果放入消息时存在问题，则抛出。

注释

准备将发送给另一 QAnywhere 客户端的消息。

此方法将消息和目标地址插入本地消息存储库中。消息传输的时间取决于 QAnywhere 代理的传输策略。

地址采用 'id\queue-name' 的形式，其中 'id' 是目标消息存储库 ID，而 'queue-name' 则用于标识目标 QAnywhere 客户端用于监听或接收消息的队列。

另请参见

“putMessageTimeToLive 方法”一节第 560 页

putMessageTimeToLive 方法

语法

```
void QAManagerBase.putMessageTimeToLive(
  String address,
  QAMessage msg,
  long ttl
)
throws QAException
```

参数

- **address** 指定目标队列名称的消息的地址。
- **msg** 要放入的消息。
- **ttl** 延迟时间（以毫秒为单位），如果此时间前还没有发送消息，则消息将到期。值为 0 表示消息并未到期。

抛出条件

- 如果放入消息时存在问题，则抛出。

注释

准备将发送给另一 QAnywhere 客户端的消息。

此方法将消息和目标地址插入本地消息存储库中。消息传输的时间取决于 QAnywhere 代理的传输策略。但是，如果下一条消息的传输时间超出给定生存期值，消息将到期。

地址采用 'id\queue-name' 的形式，其中 'id' 是目标消息存储库 ID，而 'queue-name' 则用于标识目标 QAnywhere 客户端用于监听或接收消息的队列。

setBooleanStoreProperty 方法

语法

```
void QAManagerBase.setBooleanStoreProperty(
  String name,
  boolean value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 布尔属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为布尔值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setByteStoreProperty 方法

语法

```
void QAManagerBase.setByteStoreProperty(
  String name,
  byte value
```

```
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** sbyte 属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为 sbyte 值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setDoubleStoreProperty 方法

语法

```
void QAManagerBase.setDoubleStoreProperty(
  String name,
  double value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 双精度属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为双精度值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setFloatStoreProperty 方法

语法

```
void QAManagerBase.setFloatStoreProperty(
  String name,
  float value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 浮点属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为浮点值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setIntStoreProperty 方法

语法

```
void QAManagerBase.setIntStoreProperty(
  String name,
  int value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 整型属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为整型值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setLongStoreProperty 方法

语法

```
void QAManagerBase.setLongStoreProperty(
  String name,
  long value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 长整型属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为长整型值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setMessageListener 方法

语法

```
void QAManagerBase.setMessageListener(
  String address,
  QAMessageListener listener
)
throws QAException
```

参数

- **address** 用来接收消息的本地队列名称的地址，或监听 QAnywhere 系统消息的系统。

- **listener** 监听器。

抛出条件

- 如果注册 QAMessageListener 对象时存在问题，则抛出。

注释

注册一个 QAMessageListener 对象以异步接收 QAnywhere 消息。

address 参数指定用于接收消息的本地队列名称。只能向给定队列指派一个监听器对象。如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 "system" 作为队列名称。

使用此方法异步接收消息。

另请参见

“QAMessageListener 接口”一节第 593 页

setMessageListener2 方法

语法

```
void QAManagerBase.setMessageListener2(
  String address,
  QAMessageListener2 listener
)
throws QAException
```

参数

- **address** 用来接收消息的本地队列名称的地址，或监听 QAnywhere 系统消息的系统。
- **listener** 监听器。

抛出条件

- 如果注册 QAMessageListener2 对象时存在问题，则抛出。

注释

注册一个 QAMessageListener2 对象以异步接收 QAnywhere 消息。

address 参数指定用于接收消息的本地队列名称。只能向给定队列指派一个监听器对象。如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 "system" 作为队列名称。

使用此方法异步接收消息。

另请参见

“QAMessageListener2 接口”一节第 594 页

setMessageListenerBySelector 方法

语法

```
void QAManagerBase.setMessageListenerBySelector(
  String address,
  String selector,
  QAMessageListener listener
)
throws QAException
```

参数

- **address** 用来接收消息的本地队列名称的地址，或监听 QAnywhere 系统消息的系统。
- **selector** 用于过滤要接收的消息的选择程序。
- **listener** 监听器。

抛出条件

- 如果注册 QAMessageListener 对象时存在问题（如因为已有一个指派给给定队列的监听器对象），则抛出。

注释

注册 QAMessageListener 对象以使用消息选择程序异步接收 QAnywhere 消息。

address 参数指定用于接收消息的本地队列名称。只能向给定队列指派一个监听器对象。selector 参数指定用于过滤在给定地址上接收的消息的选择程序。如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 "system" 作为队列名称。

使用此方法异步接收消息。

setMessageListenerBySelector2 方法

语法

```
void QAManagerBase.setMessageListenerBySelector2(
  String address,
  String selector,
  QAMessageListener2 listener
)
throws QAException
```

参数

- **address** 用来接收消息的本地队列名称的地址，或监听 QAnywhere 系统消息的系统。
- **selector** 用于过滤要接收的消息的选择程序。
- **listener** 监听器。

抛出条件

- 如果注册 QAMessageListener2 对象时存在问题，则抛出。

注释

注册 QAMessageListener2 对象以使用消息选择程序异步接收 QAnywhere 消息。

address 参数指定用于接收消息的本地队列名称。只能向给定队列指派一个监听器对象。selector 参数指定用于过滤在给定地址上接收的消息的选择程序。如果想要监听 QAnywhere 系统消息（包括推式通知和网络状态更改），请指定 "system" 作为队列名称。

使用此方法异步接收消息。

另请参见

“QAMessageListener2 接口”一节第 594 页

setProperty 方法

设置 QAnywhere Manager 配置属性。

语法

```
void QAManagerBase.setProperty(
  string name,
  string val
)
  throws QAException
```

参数

- **name** QAnywhere Manager 配置属性名。
- **val** QAnywhere Manager 配置属性值。

注释

可以使用此方法通过指定属性名称和值来替换缺省 QAnywhere Manager 配置属性。有关属性的列表，请参见“QAnywhere Manager 配置属性”一节第 90 页。

还可以使用属性文件和 QAManagerFactory.CreateQAManager 方法设置 QAnywhere Manager 配置属性。

有关详细信息，请参见“在文件中设置 QAnywhere Manager 配置属性”一节第 91 页。

> **注意**
> 调用 QAManager.Open 或 QATransactionalManager.Open() 前，必须设置必需的属性。

抛出条件

- “QAException 类”一节第 525 页

另请参见

- “QAManagerBase 接口”一节第 541 页
- “open 方法”一节第 540 页
- “open 方法”一节第 603 页

setShortStoreProperty 方法

语法

```
void QAManagerBase.setShortStoreProperty(
  String name,
  short value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 短整型属性值。

抛出条件

- 如果设置消息存储库属性时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为短整型值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setStoreProperty 方法

语法

```
void QAManagerBase.setStoreProperty(
  String name,
  Object value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 属性值。

抛出条件

- 如果将消息存储库属性设置为此值时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为 System.Object 值。

属性类型必须与可接受的原始数据类型之一相对应，或为字符串型。可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

setStringStoreProperty 方法

语法

```
void QAManagerBase.setStringStoreProperty(
  String name,
  String value
)
throws QAException
```

参数

- **name** 预定义或自定义属性名称。
- **value** 字符串型属性值。

抛出条件

- 如果将消息存储库属性设置为字符串值时存在问题，则抛出。

注释

将预定义或自定义消息存储库属性设置为字符串值。

可以使用此方法设置预定义或用户定义的客户端存储库属性。

有关预定义属性的列表，请参见“MessageStoreProperties 接口”一节第 505 页。

另请参见

“MessageStoreProperties 接口”一节第 505 页

start 方法

语法

void **QAManagerBase.start()**
throws **QAException**

抛出条件

- 如果启动 QAManagerBase 实例时存在问题，则抛出。

注释

启动 QAManagerBase 以接收进来的消息。

如果没有使用 QAManagerBase.stop() 调用进行干预，则除第一次调用外，对此方法的任何其它调用都将被忽略。

另请参见

“stop 方法”一节第 570 页

stop 方法

语法

void **QAManagerBase.stop()**
throws **QAException**

抛出条件

- 如果停止 QAManagerBase 实例时存在问题，则抛出。

注释

使 QAManagerBase 暂停接收进来的消息。

消息并没有丢失。再次启动管理器后才接收消息。如果没有使用 QAManagerBase.start() 调用进行干预，则除第一次调用外，对 stop() 的任何其它调用都将被忽略。

另请参见

“start 方法”一节第 570 页

triggerSendReceive 方法

语法

void **QAManagerBase.triggerSendReceive()**
throws **QAException**

抛出条件

- 如果触发发送/接收时存在问题，则抛出。

注释

在上载任何发送给其它客户端的消息和下载任何发送给本地客户端的消息时，皆与 QAnywhere 消息服务器保持同步。

对此方法的调用会使消息在 QAnywhere 代理和中央消息传递服务器之间立即保持同步。手工 triggerSendReceive() 调用将导致消息的立即传输，与 QAnywhere 代理传输策略无关。

QAnywhere 代理传输策略决定消息传输的方式。例如，当客户端接收到推式通知，或者当您调用 QAManagerBase.putMessage() 方法来发送消息时，消息传输将定期自动进行。

另请参见

“putMessage 方法”一节第 560 页

QAManagerFactory 类

语法

public **QAManagerFactory**

注释

此类充当用于创建 QATransactionalManager 和 QAManager 对象的工厂类。

只能有一个 QAManagerFactory 实例。

另请参见

“QAManager 接口”一节第 535 页

“QATransactionalManager 接口”一节第 601 页

成员

ianywhere.qanywhere.client.QAManagerFactory 的所有成员，其中包括所有继承的成员。

- “createQAManager 方法”一节第 572 页
- “createQAManager 方法”一节第 572 页
- “createQAManager 方法”一节第 573 页
- “createQATransactionalManager 方法”一节第 573 页
- “createQATransactionalManager 方法”一节第 574 页
- “createQATransactionalManager 方法”一节第 574 页
- “getInstance 方法”一节第 575 页

createQAManager 方法

语法

```
abstract QAManager QAManagerFactory.createQAManager(
  String iniFile
)
throws QAException
```

参数

- **iniFile** 用于配置 QAManager 实例的属性文件，若为空值则使用缺省属性创建 QAManager 实例。

抛出条件

- 如果创建管理器时存在问题，则抛出。

注释

返回带有指定属性的新 QAManager 实例。

如果 iniFile 参数为空值，则使用缺省属性创建 QAManager。在创建实例后，可以使用 QAManagerBase setproperty 方法以编程方式设置 QAManager 属性。

另请参见

“QAManager 接口”一节第 535 页

返回值

一个新的 QAManager 实例。

createQAManager 方法

语法

```
abstract QAManager QAManagerFactory.createQAManager(
  java.util.Hashtable properties
)
throws QAException
```

参数

- **properties** 用于配置 QAManager 实例的 Hashtable。

抛出条件

- 如果创建管理器时存在问题，则抛出。

注释

返回以指定属性作为 Hashtable 的新 QAManager 实例。

另请参见

“QAManager 接口”一节第 535 页

返回值

一个新的 QAManager 实例。

createQAManager 方法

语法

```
abstract QAManager QAManagerFactory.createQAManager()
throws QAException
```

抛出条件

- 如果创建管理器时存在问题，则抛出。

注释

返回带有缺省属性的新 QAManager 实例。

另请参见

“QAManager 接口”一节第 535 页

返回值

一个新的 QAManager 实例。

createQATransactionalManager 方法

语法

```
abstract QATransactionalManager QAManagerFactory.createQATransactionalManager(
  String iniFile
)
throws QAException
```

参数

- **iniFile** 用于配置 QATransactionalManager 实例的属性文件。

抛出条件

- 如果创建管理器时存在问题，则抛出。

注释

返回带有指定属性的新 QATransactionalManager 实例。

如果 iniFile 参数为空值，则使用缺省属性创建 QATransactionalManager。在创建该实例后，可以使用 QAManagerBase setproperty 方法以编程方式设置 QATransactionalManager 属性。

另请参见

“QATransactionalManager 接口”一节第 601 页

返回值

配置后的 QATransactionalManager。

createQATransactionalManager 方法

语法

```
abstract QATransactionalManager QAManagerFactory.createQATransactionalManager(
  java.util.Hashtable properties
)
throws QAException
```

参数

- **properties** 用于配置 QATransactionalManager 实例的 Hashtable。

抛出条件

- 如果创建管理器时存在问题，则抛出。

注释

返回带有指定属性的新 QATransactionalManager 实例。

另请参见

“QATransactionalManager 接口”一节第 601 页

返回值

配置后的 QATransactionalManager。

createQATransactionalManager 方法

语法

```
abstract QATransactionalManager QAManagerFactory.createQATransactionalManager()
throws QAException
```

抛出条件

- 如果创建管理器时存在问题，则抛出。

注释

返回带有缺省属性的新 QATransactionalManager 实例。

另请参见

“QATransactionalManager 接口”一节第 601 页

返回值

一个新的 QATransactionalManager。

getInstance 方法

语法

QAManagerFactory **QAManagerFactory.getInstance()**
throws **QAException**

抛出条件

- 如果创建管理器工厂时存在问题，则抛出。

注释

返回单个 QAManagerFactory 实例。

返回值

单个 QAManagerFactory 实例。

QAMessage 接口

语法

public **QAMessage**

派生类

- “QABinaryMessage 接口”一节第 510 页
- “QATextMessage 接口”一节第 596 页

注释

QAMessage 提供设置消息属性和标头字段的接口。

派生类 QABinaryMessage 和 QATextMessage 提供读写消息主体的专用函数。可以使用 QAMessage 函数设置预定义或自定义消息属性。

有关预定义属性名称的列表，请参见“MessageProperties 接口”一节第 499 页。

另请参见

“QABinaryMessage 接口”一节第 510 页

“QATextMessage 接口”一节第 596 页

成员

ianywhere.qanywhere.client.QAMessage 的所有成员，其中包括所有继承的成员。

DEFAULT_PRIORITY 变量

语法

final int **QAMessage.DEFAULT_PRIORITY**

注释

缺省消息优先级。

DEFAULT_TIME_TO_LIVE 变量

语法

final long **QAMessage.DEFAULT_TIME_TO_LIVE**

注释

缺省生存期值。

clearProperties 方法

语法

void **QAMessage.clearProperties()**
throws **QAException**

抛出条件

- 如果清除消息属性时存在问题，则抛出。

注释

清除消息的所有属性。

getAddress 方法

语法

String **QAMessage.getAddress()**
throws **QAException**

抛出条件

- 如果检索目标地址时存在问题，则抛出。

注释

返回 QAMessage 实例的目标地址。

发送消息时，此字段被忽略。发送操作完成后，此字段保留 QAManagerBase.putMessage(String, QAMessage) 中指定的目标地址。

返回值

QAMessage 实例的目标地址。

getBooleanProperty 方法

语法

```
boolean QAMessage.getBooleanProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取布尔型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getByteProperty 方法

语法

```
byte QAMessage.getByteProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取有符号字节型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getDoubleProperty 方法

语法

```
double QAMessage.getDoubleProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取双精度型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getExpiration 方法

语法

```
java.util.Date QAMessage.getExpiration()
throws QAException
```

抛出条件

- 如果获取有效期时存在问题，则抛出。

注释

返回消息的有效期值，或在消息没有到期或尚未发送的情况下返回空值。

发送消息时，有效期保持未指派状态。发送操作完成后，它用于保存消息的到期时间。

它是一个只读属性，因为消息的到期时间是通过将 QAManagerBase.putMessageTimeToLive(String, QAMessage, long) 的生存期参数值与当前时间相加而设置的。

另请参见

“putMessageTimeToLive 方法”一节第 560 页

返回值

消息的有效期值，或在消息没有到期或尚未发送的情况下返回空值。

getFloatProperty 方法

语法

```
float QAMessage.getFloatProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取浮点型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getInReplyToID 方法

语法

```
String QAMessage.getInReplyToID()
throws QAException
```


抛出条件

- 如果获取此消息所回复的消息的消息 ID 时存在问题，则抛出。

注释

返回此消息所回复的消息的消息 ID。

返回值

返回此消息所回复的消息的消息 ID，或在此消息不回复时返回空值。

getIntProperty 方法

语法

```
int QAMessage.getIntProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取整型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getLongProperty 方法

语法

```
long QAMessage.getLongProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取长整型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getMessageID 方法

语法

String **QAMessage.getMessageID()**
throws **QAException**

抛出条件

- 如果获取消息 ID 时存在问题，则抛出。

注释

返回消息的全局唯一消息 ID。

此属性在放入消息前一直为空值。

使用 QAManagerBase.putMessage(String, QAMessage) 发送消息时，消息 ID 为空值并且可以被忽略。发送方法返回时，它包含一个指派的值。

另请参见

“putMessage 方法”一节第 560 页

返回值

消息的消息 ID，或在尚未放入消息时为空值。

getPriority 方法

语法

int **QAMessage.getPriority()**
throws **QAException**

抛出条件

- 如果获取消息优先级时存在问题，则抛出。

注释

返回消息的优先级（从 0 到 9）。

返回值

消息的优先级。

getProperty 方法

语法

```
Object QAMessage.getProperty(
  String name
)
throws QAException
```

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误，则抛出。

注释

获取消息属性。

返回值

返回属性值；如果属性不存在，则返回空值。

getPropertyNames 方法

语法

```
java.util.Enumeration QAMessage.getPropertyNames()
throws QAException
```

抛出条件

- 如果获取消息属性名称的枚举器时存在问题，则抛出。

注释

获取消息属性名称的枚举器。

返回值

消息属性名称的枚举器。

getPropertyType 方法

语法

short **QAMessage.getPropertyType(**
String *name*
)
throws **QAException**

参数

- **name** 属性名称。

抛出条件

- 如果检索属性类型时存在问题，则抛出。

注释

返回给定属性的属性类型。

另请参见

“PropertyType 接口”一节第 508 页

返回值

属性类型。

getRedelivered 方法

语法

boolean **QAMessage.getRedelivered()**
throws **QAException**

抛出条件

- 如果检索重新发送的状态时存在问题，则抛出。

注释

指示消息是否以前已接收但尚未确认。

如果接收方 QAManager 检测到以前曾收到过正在接收的消息，则设置 Redelivered。

例如，应用程序使用以 AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 打开的 QAManager 接收消息，并且没有确认此消息就关闭。当应用程序再次启动并接收同一消息时，此消息被标记为 redelivered。

返回值

如果以前收到过此消息但未确认，则返回 True。

getReplyToAddress 方法

语法

String **QAMessage.getReplyToAddress()**
throws **QAException**

抛出条件

- 如果检索回复地址时存在问题，则抛出。

注释

返回此消息的回复地址。

返回值

此消息的回复地址，或在此消息不存在时返回空值。

getShortProperty 方法

语法

short **QAMessage.getShortProperty(**
String *name*
)
throws **QAException**

参数

- **name** 属性名称。

抛出条件

- 如果获取属性值时发生转换错误或此属性不存在，则抛出。

注释

获取短整型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

属性值。

getStringProperty 方法

语法

String **QAMessage.getStringProperty(**
  String *name*
)
throws **QAException**

参数

- **name** 属性名称。

抛出条件

- 如果检索消息属性时存在问题，则抛出。

注释

获取字符串型消息属性。

另请参见

“MessageProperties 接口”一节第 499 页

返回值

返回属性值；如果属性不存在，则返回空值。

getTimestamp 方法

语法

java.util.Date **QAMessage.getTimestamp()**
throws **QAException**

抛出条件

- 如果检索消息时间戳时存在问题，则抛出。

注释

返回消息时间戳，即创建消息的时间。

返回值

消息时间戳。

propertyExists 方法

语法

```
boolean QAMessage.propertyExists(
  String name
)
throws QAException
```

参数

- **name** 属性名称

抛出条件

- 如果检查是否已设置此属性时存在问题，则抛出。

注释

指示是否已为此消息设置了给定属性。

返回值

当属性存在时返回 true。

setBooleanProperty 方法

语法

```
void QAMessage.setBooleanProperty(
  String name,
  boolean value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置布尔型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setByteProperty 方法

语法

```
void QAMessage.setByteProperty(
  String name,
  byte value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置有符号字节型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setDoubleProperty 方法

语法

```
void QAMessage.setDoubleProperty(
  String name,
  double value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置双精度型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setFloatProperty 方法

语法

```
void QAMessage.setFloatProperty(
  String name,
  float value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置浮点型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setInReplyToID 方法

语法

```
void QAMessage.setInReplyToID(
  String id
)
throws QAException
```

参数

- **id** 此消息将回复的消息的 ID。

抛出条件

- 如果设置回复 ID 时存在问题，则抛出。

注释

设置回复 ID，它用来标识此消息将回复的消息。

setIntProperty 方法

语法

```
void QAMessage.setIntProperty(
  String name,
  int value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置整型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setLongProperty 方法

语法

```
void QAMessage.setLongProperty(
  String name,
  long value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置长整型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setPriority 方法

语法

void **QAMessage.setPriority(**
  int *priority*
)
throws **QAException**

参数

- **priority** 消息的优先级。

抛出条件

- 如果设置优先级时存在问题，则抛出。

注释

设置消息的优先级（从 0 到 9）。

setProperty 方法

语法

void **QAMessage.setProperty(**
  String *name*,
  Object *value*
)
throws **QAException**

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置属性。

属性类型必须与可接受的原始数据类型之一相对应，或为字符串型。

另请参见

“MessageProperties 接口”一节第 499 页

setReplyToAddress 方法

语法

```
void QAMessage.setReplyToAddress(
  String address
)
throws QAException
```

参数

- **address** 回复地址。

抛出条件

- 如果设置回复地址时存在问题，则抛出。

注释

设置回复地址。

setShortProperty 方法

语法

```
void QAMessage.setShortProperty(
  String name,
  short value
)
throws QAException
```

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置短整型属性。

另请参见

“MessageProperties 接口”一节第 499 页

setStringProperty 方法

语法

void **QAMessage.setStringProperty(**
String *name*,
String *value*
)
throws **QAException**

参数

- **name** 属性名称。
- **value** 属性值。

抛出条件

- 如果设置属性时存在问题，则抛出。

注释

设置字符串型属性。

另请参见

“MessageProperties 接口”一节第 499 页

QAMessageListener 接口

语法

public **QAMessageListener**

注释

若要监听消息，则应通过调用 QAManagerBase.setMessageListener(String,QAMessageListener) 来实现此接口并注册实现。

另请参见

“setMessageListener 方法”一节第 564 页

成员

ianywhere.qanywhere.client.QAMessageListener 的所有成员，其中包括所有继承的成员。

- “onException 方法”一节第 594 页
- “onMessage 方法”一节第 594 页

onException 方法

语法

void **QAMessageListener.onException(**
  QAException *exception*,
  QAMessage *message*
)

参数

- **exception** 发生的异常。
- **message** 在将消息传递到 onMessage(QAMessage) 后发生异常的情况下，处理的消息。否则，为空值。

注释

只要在监听消息时发生了异常，就调用此方法。

请注意，此方法无法用于自动关闭 QAManagerBase 实例，因为 QAManagerBase.close() 方法在所有消息监听器完成处理之前一直阻塞。

另请参见

“QAManagerBase 接口”一节第 541 页

“close 方法”一节第 546 页

onMessage 方法

语法

void **QAMessageListener.onMessage(**
  QAMessage *message*
)

参数

- **message** 接收到的消息。

注释

只要收到消息，就调用此方法。

QAMessageListener2 接口

语法

public **QAMessageListener2**

注释

若要监听消息，则应通过调用 QAManagerBase 来实现此接口并注册这一实现。

setMessageListener2(String,QAMessageListener2)。

另请参见

“setMessageListener2 方法”一节第 565 页

成员

ianywhere.qanywhere.client.QAMessageListener2 的所有成员，其中包括所有继承的成员。

- “onException 方法”一节第 595 页
- “onMessage 方法”一节第 596 页

onException 方法

语法

```
void QAMessageListener2.onException(
  QAManagerBase mgr,
  QAException exception,
  QAMessage message
)
```

参数

- **mgr** 处理消息的 QAManagerBase。
- **exception** 发生的异常。
- **message** 在将消息传递到 onMessage(QAMessage) 后发生异常的情况下，处理的消息。否则，为空值。

注释

只要在监听消息时发生了异常，就调用此方法。

请注意，此方法无法用于自动关闭 QAManagerBase 实例，因为 QAManagerBase.close() 方法在所有消息监听器完成处理之前一直阻塞。

另请参见

“QAManagerBase 接口”一节第 541 页

“close 方法”一节第 546 页

“onMessage 方法”一节第 594 页

onMessage 方法

语法

```
void QAMessageListener2.onMessage(
  QAManagerBase mgr,
  QAMessage message
)
```

参数

- **mgr** 接收消息的 QAManagerBase。
- **message** 接收到的消息。

注释

只要收到消息，就调用此方法。

另请参见

“QAManagerBase 接口”一节第 541 页

QATextMessage 接口

语法

public **QATextMessage**

基类

- “QAMessage 接口”一节第 575 页

注释

QATextMessage 继承自 QAMessage 类，并添加了文本消息主体以及读写文本消息主体的方法。

首次创建消息时，消息的主体处于只写模式。消息发送后，发送消息的客户端可保留和修改该消息，而不会影响已发送的消息。可以多次发送同一消息对象。

接收到消息时，提供程序已调用 QATextMessage.reset()，因此消息主体处于只读模式并且从消息主体的开头开始读取值。

另请参见

“onMessage 方法”一节第 594 页

成员

ianywhere.qanywhere.client.QATextMessage 的所有成员，其中包括所有继承的成员。

getText 方法

语法

String **QATextMessage.getText()**
throws **QAException**

抛出条件

- 如果检索消息文本时存在问题，则抛出。

注释

返回消息文本。

如果消息文本超出 QAManager.MAX_IN_MEMORY_MESSAGE_SIZE 属性指定的最大大小，则此方法返回空值。在这种情况下，使用 QATextMessage.readText(int) 方法读取文本。

另请参见

“readText 方法”一节第 598 页

返回值

消息文本或空值。

getTextLength 方法

语法

long **QATextMessage.getTextLength()**
throws **QAException**

抛出条件

- 如果检索消息长度时存在问题，则抛出。

注释

返回消息长度（以字符数表示）。

返回值

消息长度（以字符数表示）。

readText 方法

语法

String **QATextMessage.readText(**
int *maxLength*

)
throws **QAException**

参数

- **maxLength** 要读取的最大字符数。

抛出条件

- 如果检索未读文本时存在问题，则抛出。

注释

返回消息的未读文本。

任何其它未读文本必须使用此方法的后续调用读取。从任何未读文本的开头读取文本。

返回值

文本。

reset 方法

语法

void **QATextMessage.reset()**
throws **QAException**

抛出条件

- 如果重置消息的文本位置时存在问题，则抛出。

注释

将消息的文本位置重置为开头。

setText 方法

语法

void **QATextMessage.setText(**
String *value*
)
throws **QAException**

参数

- **value** 要写入消息主体的文本。

抛出条件

- 如果覆盖消息文本时存在问题，则抛出。

注释

覆盖消息文本。

writeText 方法

语法

```
void QATextMessage.writeText(
  String value
)
throws QAException
```

参数

- **value** 要附加的文本。

抛出条件

- 如果附加消息文本时存在问题，则抛出。

注释

将文本附加到消息文本中。

writeText 方法

语法

```
void QATextMessage.writeText(
  String value,
  int length
)
throws QAException
```

参数

- **value** 要附加的文本。
- **length** 要附加的文本的字符数。

抛出条件

- 如果附加消息文本时存在问题，则抛出。

注释

将文本附加到消息文本中。

writeText 方法

语法

```
void QATextMessage.writeText(
  String value,
  int offset,
  int length
)
throws QAException
```

参数

- **value** 要附加的文本。
- **offset** 对要附加的文本值的偏移。
- **length** 要附加的文本的字符数。

抛出条件

- 如果附加消息文本时存在问题，则抛出。

注释

将文本附加到消息文本中。

QATransactionalManager 接口

语法

public **QATransactionalManager**

基类

- “QAManagerBase 接口”一节第 541 页

注释

QATransactionalManager 类从 QAManagerBase 派生并且管理事务性 QAnywhere 消息传递操作。

有关派生行为的详细说明，请参见“QAManagerBase 接口”一节第 541 页。

QATransactionalManager 实例只能用于事务性确认。使用 QATransactionalManager.commit() 方法提交全部 QAManagerBase.putMessage(String, QAMessage) 和 QAManagerBase.getMessage(String) 调用。

另请参见

“commit 方法”一节第 603 页

“putMessage 方法”一节第 560 页

“getMessage 方法”一节第 551 页

成员

ianywhere.qanywhere.client.QATransactionalManager 的所有成员，其中包括所有继承的成员。

- “setStringStoreProperty 方法”一节第 569 页
- “start 方法”一节第 570 页
- “stop 方法”一节第 570 页
- “triggerSendReceive 方法”一节第 570 页

commit 方法

语法

void **QATransactionalManager.commit()**
throws **QAException**

抛出条件

- 如果提交时存在问题，则抛出。

注释

提交当前事务并开始新的事务。

此方法会提交全部 QAManagerBase.putMessage(String, QAMessage) 和 QAManagerBase.getMessage(String) 调用。

第一个事务以调用 QATransactionalManager.open() 开始。

open 方法

语法

void **QATransactionalManager.open()**
throws **QAException**

抛出条件

- 如果打开管理器时存在问题，则抛出。

注释

打开一个 QATransactionalManager 实例。

此方法必须是创建管理器后调用的第一个方法。

rollback 方法

语法

void **QATransactionalManager.rollback()**
throws **QAException**

抛出条件

- 如果回退时存在问题，则抛出。

注释

回退当前事务并开始新的事务。

此方法会回退全部未提交的 QAManagerBase.putMessage(String, QAMessage) 和 QAManagerBase.getMessage(String) 调用。

QueueDepthFilter 接口

语法

public **QueueDepthFilter**

注释

为 QAManagerBase.getQueueDepth(short) 和 QAManagerBase.getQueueDepth(String, short) 提供队列深度过滤器值。

另请参见

“getQueueDepth 方法”一节第 556 页

“getQueueDepth 方法”一节第 556 页

成员

ianywhere.qanywhere.client.QueueDepthFilter 的所有成员，其中包括所有继承的成员。

- “ALL 变量”一节第 604 页
- “INCOMING 变量”一节第 605 页
- “LOCAL 变量”一节第 605 页
- “OUTGOING 变量”一节第 605 页

ALL 变量

语法

final short **QueueDepthFilter.ALL**

注释

此过滤器指定进来的消息和外发消息。

系统消息和到期消息不包含在队列深度计数中。

INCOMING 变量

语法

final short **QueueDepthFilter.INCOMING**

注释

此过滤器只指定进来的消息。

进来的消息被定义为其发出方不同于消息存储库的代理 ID 的消息。

LOCAL 变量

语法

final short **QueueDepthFilter.LOCAL**

注释

如果使用 LOCAL 过滤器调用 getQueueDepth 且指定了队列，则该变量返回发送到该队列的未接收本地消息数。如果未指定队列，LOCAL 返回消息存储库中未接收的本地消息的总数，但不包括系统消息。

OUTGOING 变量

语法

final short **QueueDepthFilter.OUTGOING**

注释

此过滤器只指定外发消息。

外发的消息被定义为其发出方是消息存储库的代理 ID，而其目标不是消息存储库的代理 ID 的消息。

StatusCodes 接口

语法

public **StatusCodes**

注释

此接口为消息状态定义了一组代码。

成员

ianywhere.qanywhere.client.StatusCodes 的所有成员，其中包括所有继承的成员。

CANCELLED 变量

语法

final int **StatusCodes.CANCELLED**

注释

消息已取消。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

EXPIRED 变量

语法

final int **StatusCodes.EXPIRED**

注释

消息已到期；消息在达到到期时间之前未被接收。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

FINAL 变量

语法

final int **StatusCodes.FINAL**

注释

此常量用于确定消息是否已达到最终状态。

当且仅当消息状态大于此常量时，消息达到最终状态。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

LOCAL 变量

语法

final int **StatusCodes.LOCAL**

注释

此消息被发送到本地消息存储库，没有传输到服务器。

此代码适用于 MessageProperties.TRANSMISSION_STATUS。

另请参见

“TRANSMISSION_STATUS 变量”一节第 505 页

PENDING 变量

语法

final int **StatusCodes.PENDING**

注释

消息已发送但未被接收。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

RECEIVED 变量

语法

final int **StatusCodes.RECEIVED**

注释

此消息已由接收方接收并确认。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

RECEIVING 变量

语法

final int **StatusCodes.RECEIVING**

注释

消息正处于接收过程中，或者已接收但未确认。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

TRANSMITTED 变量

语法

final int **StatusCodes.TRANSMITTED**

注释

消息已经传输到服务器。

此代码适用于 MessageProperties.TRANSMISSION_STATUS。

另请参见

“TRANSMISSION_STATUS 变量”一节第 505 页

TRANSMITTING 变量

语法

final int **StatusCodes.TRANSMITTING**

注释

此消息正处于传输到服务器的过程中。

此代码适用于 MessageProperties.TRANSMISSION_STATUS。

另请参见

“TRANSMISSION_STATUS 变量”一节第 505 页

UNRECEIVABLE 变量

语法

final int **StatusCodes.UNRECEIVABLE**

注释

此消息已标记为无法接收。

消息格式错误或尝试发送时失败次数过多。

此代码适用于 MessageProperties.STATUS。

另请参见

“STATUS 变量”一节第 504 页

UNTRANSMITTED 变量

语法

final int **StatusCodes.UNTRANSMITTED**

注释

消息未传输到服务器。

此代码适用于 MessageProperties.TRANSMISSION_STATUS。

另请参见

“TRANSMISSION_STATUS 变量”一节第 505 页

用于 Web 服务的 QAnywhere Java API

包

ianywhere.qanywhere.ws

WSBase 类

语法

public **WSBase**

注释

它是由移动 Web 服务编译器生成的主要 Web 服务代理类的基类。

成员

ianywhere.qanywhere.ws.WSBase 的所有成员，其中包括所有继承的成员。

- “clearRequestProperties 方法”一节第 611 页
- “getResult 方法”一节第 611 页
- “getServiceID 方法”一节第 612 页
- “setListener 方法”一节第 612 页
- “setListener 方法”一节第 613 页
- “setProperty 方法”一节第 613 页
- “setQAManager 方法”一节第 614 页
- “setRequestProperty 方法”一节第 614 页
- “setServiceID 方法”一节第 614 页
- “WSBase 方法”一节第 611 页
- “WSBase 方法”一节第 610 页

WSBase 方法

语法

WSBase.WSBase()
throws **WSException**

抛出条件

- 如果构造 WSBase 时存在问题，则抛出。

注释

构造函数。

WSBase 方法

语法

```
WSBase.WSBase(
  String iniFile
)
throws WSException
```

参数

- **iniFile** 包含配置属性的文件。

抛出条件

- 如果构造 WSBase 时存在问题，则抛出。

注释

配置属性文件的构造函数。

有效配置属性有：

- **LOG_FILE** 记录运行时信息的文件。
- **LOG_LEVEL** 介于 0 到 6 之间的值，用于控制所记录信息的详细程度，6 表示详细程度最高。
- **WS_CONNECTOR_ADDRESS** MobiLink 服务器中 Web 服务连接器的地址。缺省 WS_CONNECTOR_ADDRESS 是 "ianywhere.connector.webservices\\"。

clearRequestProperties 方法

语法

```
void WSBase.clearRequestProperties()
```

注释

清除为此 WSBase 设置的全部请求属性。

getResult 方法

语法

```
WSResult WSBase.getResult(
  String requestID
)
```

参数

- **requestID** Web 服务请求的 ID。

注释

获取代表 Web 服务请求结果的 WSResult 对象。

返回值

代表 Web 服务请求结果的一个 WSResult 实例。

另请参见

“WSStatus 类”一节第 642 页

getServiceID 方法

语法

String **WSBase.getServiceID()**

注释

获取此 WSBase 实例的服务 ID。

返回值

服务 ID。

setListener 方法

语法

```
void WSBase.setListener(
  String requestID,
  WSListener listener
)
```

参数

- **requestID** 要监听结果的 Web 服务请求的 ID。
- **listener** 给定 Web 服务请求的结果可用时调用的监听器对象。

注释

为给定 Web 服务请求的结果设置监听器。

监听器通常用于获取此服务的 asyncXYZ 方法的结果。

要删除监听器，可在调用 setListener 时将 listener 参数设置为空值。

注意：此方法会替换由先前的任意一次 SetListener 调用所设置的监听器。

setListener 方法

语法

```
void WSBase.setListener(
  WSListener listener
)
```

参数

- **listener** Web 服务请求的结果可用时调用的监听器对象。

注释

为此 WSBase 实例发出的所有 Web 服务请求的结果设置监听器。

监听器通常用于获取此服务的 asyncXYZ 方法的结果。

要删除监听器，可在调用 setListener 时将 listener 参数设置为空值。

此方法会替换由先前的任意一次 SetListener 调用所设置的监听器。

setProperty 方法

语法

```
void WSBase.setProperty(
  String property,
  String val
)
```

参数

- **property** 要设置的属性名称。
- **val** 属性值。

注释

为此 WSBase 实例设置配置属性。

在发出任何异步或同步 Web 服务请求前，必须设置配置属性；晚于此时此方法将不生效。

有效配置属性有：

- **LOG_FILE** 记录运行时信息的文件。
- **LOG_LEVEL** 介于 0 到 6 之间的值，用于控制所记录信息的详细程度，6 表示详细程度最高。
- **WS_CONNECTOR_ADDRESS** MobiLink 服务器中 Web 服务连接器的地址。缺省 WS_CONNECTOR_ADDRESS 是 "ianywhere.connector.webservices\\"。

setQAManager 方法

语法

```
void WSBase.setQAManager(
  QAManagerBase mgr
)
```

参数

- **mgr** 要使用的 QAManagerBase。

注释

设置此 Web 服务客户端用于发出 Web 服务请求的 QAManagerBase。

如果使用 EXPLICIT_ACKNOWLEDGEMENT QAManager，可以通过调用 WSResult 的 acknowledge() 方法确认异步 Web 服务请求的结果。即使是使用 EXPLICIT_ACKNOWLEDGEMENT QAManager，也会自动确认同步 Web 服务请求的结果。如果使用 IMPLICIT_ACKNOWLEDGEMENT QAManager，则可以自动确认任意 Web 服务请求的结果。

setRequestProperty 方法

语法

```
void WSBase.setRequestProperty(
  String name,
  Object value
)
```

参数

- **name** 要设置的属性名称。
- **value** 属性值。

注释

为 WSBase 的此实例发出的 Web 服务请求设置请求属性。

对由此 WSBase 发送的每个 QAMessage 都设置一个请求属性，直到此属性被清除。请求属性可通过将其设置为空值来清除。消息属性的类型由值参数的类来确定。例如，如果值是一个整数实例，则 SetIntProperty 用于设置 QAMessage 的属性。

setServiceID 方法

语法

```
void WSBase.setServiceID(
  String serviceID
)
```


参数

- **serviceID** 服务 ID。

注释

为此 WSBase 实例设置用户定义 ID。

服务 ID 应设置为此 WSBase 实例独有的值。它在内部使用，用来构成发送和接收 Web 服务请求的队列名称。服务 ID 应在应用程序会话间持久存在，以便检索之前会话中发出的 Web 服务请求的结果。

WSException 类

语法

public **WSException**

派生类

- “WSFaultException 类”一节第 617 页

注释

此类表示处理 Web 服务请求期间出现的异常。

成员

ianywhere.qanywhere.ws.WSException 的所有成员，其中包括所有继承的成员。

- “getErrorCode 方法”一节第 616 页
- “WSException 方法”一节第 615 页
- “WSException 方法”一节第 616 页
- “WSException 方法”一节第 616 页

WSException 方法

语法

```
WSException.WSException(
 String msg
)
```

参数

- **msg** 错误消息。

注释

使用指定的错误消息构造新异常。

WSException 方法

语法

```
WSException.WSException(
 String msg,
 int errorCode
)
```

参数

- **msg** 错误消息。
- **errorCode** 错误代码。

注释

使用指定的错误消息和错误代码构造新异常。

WSException 方法

语法

```
WSException.WSException(
 Exception exception
)
```

参数

- **exception** 异常。

注释

构造一个新异常。

getErrorCode 方法

语法

int **WSException.getErrorCode()**

注释

获取与此异常相关联的错误代码。

返回值

与此异常相关联的错误代码。

WSFaultException 类

语法

public **WSFaultException**

基类

- “WSException 类”一节第 615 页

注释

此类表示来自 Web 服务连接器的 SOAP 错误异常。

成员

ianywhere.qanywhere.ws.WSFaultException 的所有成员，其中包括所有继承的成员。

- “getErrorCode 方法”一节第 616 页
- “WSException 方法”一节第 615 页
- “WSException 方法”一节第 616 页
- “WSException 方法”一节第 616 页
- “WSFaultException 方法”一节第 617 页

WSFaultException 方法

语法

WSFaultException.WSFaultException(
String *msg*
)

参数

- **msg** 错误消息。

注释

使用指定的错误消息构造新异常。

WSListener 接口

语法

public **WSListener**

注释

此类表示用于监听 Web 服务请求结果的监听器。

成员

ianywhere.qanywhere.ws.WSListener 的所有成员，其中包括所有继承的成员。

- “onException 方法”一节第 618 页
- “onResult 方法”一节第 618 页

onException 方法

语法

```
void WSListener.onException(
  WSException e,
  WSResult wsResult
)
```

参数

- **e** 处理结果期间发生的 WSException。
- **WSResult** 可从中获取请求 ID 的 WSResult。此 WSResult 的值未定义。

注释

在处理异步 Web 服务请求的结果期间出现异常时调用。

另请参见

“WSException 类”一节第 615 页

“WSResult 类”一节第 619 页

onResult 方法

语法

```
void WSListener.onResult(
  WSResult wsResult
)
```

参数

- **WSResult** 描述 Web 服务请求的结果的 WSResult。

注释

使用异步 Web 服务请求的结果调用。

另请参见

“WSResult 类”一节第 619 页

WSResult 类

语法

public **WSResult**

注释

此类表示 Web 服务请求的结果。

- 它被传递到 WSListener.onResult。
- 它由编译器生成的服务代理的 asyncXYZ 方法返回。
- 通过调用具有指定请求 ID 的 WSBase.getResult 获取。

可通过以下三种方式获取 WSResult 对象：

成员

ianywhere.qanywhere.ws.WSResult 的所有成员，其中包括所有继承的成员。

- “getStringValue 方法”一节第 641 页
- “getValue 方法”一节第 642 页

acknowledge 方法

语法

void **WSResult.acknowledge()**

注释

确认此 WSResult 已处理。

仅在使用 EXPLICIT_ACKNOWLEDGEMENT QAManager 时，此方法才有用。

getArrayValue 方法

语法

```
WSSerializable[] WSResult.getArrayValue(
  String parentName
)
throws WSException
```

参数

- **parentName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一组复杂类型的值。

返回值

值。

getBigDecimalArrayValue 方法

语法

```
BigDecimal[] WSResult.getBigDecimalArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 BigDecimal 数组值。

返回值

值。

getBigDecimalValue 方法

语法

```
BigDecimal WSResult.getBigDecimalValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 BigDecimal 值。

返回值

值。

getBigIntegerArrayValue 方法

语法

```
BigInteger[] WSResult.getBigIntegerArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 BigInteger 数组值。

返回值

值。

getBigIntegerValue 方法

语法

```
BigInteger WSResult.getBigIntegerValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 BigInteger 值。

返回值

值。

getBooleanArrayValue 方法

语法

```
Boolean[] WSResult.getBooleanArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Boolean 数组值。

返回值

值。

getBooleanValue 方法

语法

```
Boolean WSResult.getBooleanValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Boolean 值。

返回值

值。

getByteArrayValue 方法

语法

```
Byte[] WSResult.getByteArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Byte 数组值。

返回值

值。

getByteValue 方法

语法

```
Byte WSResult.getByteValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Byte 值。

返回值

值。

getCharacterArrayValue 方法

语法

```
Character[] WSResult.getCharacterArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Character 数组值。

返回值

值。

getCharacterValue 方法

语法

Character **WSResult.getCharacterValue(**
  String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Character 值。

返回值

值。

getDoubleArrayValue 方法

语法

Double[] **WSResult.getDoubleArrayValue(**
  String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Double 数组值。

返回值

值。

getDoubleValue 方法

语法

Double **WSResult.getDoubleValue(**
  String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Double 值。

返回值

值。

getErrorMessage 方法

语法

String **WSResult.getErrorMessage()**

注释

获取错误消息。

返回值

错误消息。

getFloatArrayValue 方法

语法

Float[] **WSResult.getFloatArrayValue(**
  String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Float 数组值。

返回值

值。

getFloatValue 方法

语法

```
Float WSResult.getFloatValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Float 值。

返回值

值。

getIntegerArrayValue 方法

语法

```
Integer[] WSResult.getIntegerArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Integer 数组值。

返回值

值。

getIntegerValue 方法

语法

```
Integer WSResult.getIntegerValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Integer 值。

返回值

值。

getLongArrayValue 方法

语法

```
Long[] WSResult.getLongArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Long 数组值。

返回值

值。

getLongValue 方法

语法

```
Long WSResult.getLongValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Long 值。

返回值

值。

getObjectArrayValue 方法

语法

```
Object[] WSResult.getObjectArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一组复杂类型的值。

返回值

值。

getObjectValue 方法

语法

```
Object WSResult.getObjectValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取复杂类型值。

返回值

值。

getPrimitiveBooleanArrayValue 方法

语法

```
boolean[] WSResult.getPrimitiveBooleanArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取布尔数组值。

返回值

值。

getPrimitiveBooleanValue 方法

语法

```
boolean WSResult.getPrimitiveBooleanValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取布尔值。

返回值

值。

getPrimitiveByteArrayValue 方法

语法

```
byte[] WSResult.getPrimitiveByteArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取字节数组值。

返回值

值。

getPrimitiveByteValue 方法

语法

byte **WSResult.getPrimitiveByteValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个字节值。

返回值

值。

getPrimitiveCharArrayValue 方法

语法

char[] **WSResult.getPrimitiveCharArrayValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取字符数组值。

返回值

值。

getPrimitiveCharValue 方法

语法

```
char WSResult.getPrimitiveCharValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个字符值。

返回值

值。

getPrimitiveDoubleArrayValue 方法

语法

```
double[] WSResult.getPrimitiveDoubleArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取双精度数组值。

返回值

值。

getPrimitiveDoubleValue 方法

语法

```
double WSResult.getPrimitiveDoubleValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个双精度值。

返回值

值。

getPrimitiveFloatArrayValue 方法

语法

```
float[] WSResult.getPrimitiveFloatArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取浮点数组值。

返回值

值。

getPrimitiveFloatValue 方法

语法

float **WSResult.getPrimitiveFloatValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个浮点值。

返回值

值。

getPrimitiveIntArrayValue 方法

语法

int[] **WSResult.getPrimitiveIntArrayValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取整型数组值。

返回值

值。

getPrimitiveIntValue 方法

语法

```
int WSResult.getPrimitiveIntValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个整型值。

返回值

值。

getPrimitiveLongArrayValue 方法

语法

```
long[] WSResult.getPrimitiveLongArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取长整型数组值。

返回值

值。

getPrimitiveLongValue 方法

语法

long **WSResult.getPrimitiveLongValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个长整型值。

返回值

值。

getPrimitiveShortArrayValue 方法

语法

short[] **WSResult.getPrimitiveShortArrayValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取短整型数组值。

返回值

值。

getPrimitiveShortValue 方法

语法

short **WSResult.getPrimitiveShortValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取一个短整型值。

返回值

值。

getRequestID 方法

语法

String **WSResult.getRequestID()**

注释

获取此 WSResult 代表的请求 ID。

如果想要在所运行的应用程序与发出请求时所运行的应用程序不同的情况下获取与 Web 服务请求相对应的 WSResult，则此请求 ID 在应用程序的运行间隔期应一直保留。

返回值

请求 ID。

getShortArrayValue 方法

语法

Short[] **WSResult.getShortArrayValue(**
String *elementName*
)
throws **WSException**

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Short 数组值。

返回值

值。

getShortValue 方法

语法

```
Short WSResult.getShortValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取 java.lang.Short 值。

返回值

值。

getStatus 方法

语法

```
int WSResult.getStatus()
```

注释

获取此 WSResult 的状态。

返回值

状态代码。

另请参见

“WSStatus 类”一节第 642 页

getStringArrayValue 方法

语法

```
String[] WSResult.getStringArrayValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取字符串数组值。

返回值

值。

getStringValue 方法

语法

```
String WSResult.getStringValue(
  String elementName
)
throws WSException
```

参数

- **elementName** 此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取字符串值。

返回值

值。

getValue 方法

语法

Object **WSResult.getValue(**
  String *elementName*
)
throws **WSException**

参数

- **elementName**　此值在 WSDL 文档中的元素名称。

抛出条件

- 如果获取值时存在问题，则抛出。

注释

从此 WSResult 获取复杂类型值。

返回值

值。

WSStatus 类

语法

public **WSStatus**

注释

此类定义 Web 服务请求状态的代码。

成员

ianywhere.qanywhere.ws.WSStatus 的所有成员，其中包括所有继承的成员。

- “STATUS_ERROR 变量”一节第 642 页
- “STATUS_QUEUED 变量”一节第 643 页
- “STATUS_RESULT_AVAILABLE 变量”一节第 643 页
- “STATUS_SUCCESS 变量”一节第 643 页

STATUS_ERROR 变量

语法

final int **WSStatus.STATUS_ERROR**

注释

处理请求时存在错误。

STATUS_QUEUED 变量

语法

final int **WSStatus.STATUS_QUEUED**

注释

已将此请求放入队列中等待发送给服务器。

STATUS_RESULT_AVAILABLE 变量

语法

final int **WSStatus.STATUS_RESULT_AVAILABLE**

注释

请求的结果可用。

STATUS_SUCCESS 变量

语法

final int **WSStatus.STATUS_SUCCESS**

注释

请求成功。

QAnywhere SQL API 参考

目录

消息属性、标头和内容

本节介绍了帮助设置消息标头、消息内容和消息属性的 QAnywhere SQL 存储过程。

消息标头

可使用下面的存储过程获取和设置消息标头信息。

请参见“消息标头”一节第 684 页。

ml_qa_getaddress

返回消息的 QAnywhere 地址。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

VARCHAR(128) 形式的 QAnywhere 消息地址。QAnywhere 消息地址的格式为 *id\queue-name*。

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “QAnywhere 消息地址”一节第 63 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到一条消息，并将其地址输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @addr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @addr = ml_qa_getaddress( @msgid );
    message 'message to address ' || @addr || ' received';
    commit;
end
```

ml_qa_getexpiration

返回消息的到期时间。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

TIMESTAMP 形式的到期时间。如果没有到期时间，则返回空值。

注释

ml_qa_putmessage 完成后，如果在指定时间内预期接收者未收到消息，则消息到期。然后，可使用缺省的 QAnywhere 删除规则删除此消息。

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “消息删除规则”一节第 772 页
- “发送 QAnywhere 消息”一节第 66 页
- “ml_qa_setexpiration”一节第 652 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到一条消息，并将消息到期时间输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @expires timestamp;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @expires = ml_qa_getexpiration( @msgid );
    message 'message would have expired at ' || @expires || ' if it had not
been received';
    commit;
end
```

ml_qa_getinreplytoid

返回消息的回复 ID。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

VARCHAR(128) 形式的回复 ID。

注释

客户端可使用 InReplyToID 标头字段，将一条消息与另一条消息相链接。典型应用是将响应消息与其请求消息相链接。

回复 ID 是此消息将回复的消息的 ID。

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setinreplytoid”一节第 652 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到一条消息，并将消息的回复 ID 输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @inreplytoid varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @inreplytoid = ml_qa_getinreplytoid( @msgid );
    message 'message is likely a reply to the message with id ' ||
@inreplytoid;
    commit;
end
```

ml_qa_getpriority

返回消息的优先级级别。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

INTEGER 形式的优先级级别。

注释

QAnywhere API 定义十种级别的优先级值，0 代表最低优先级，9 代表最高优先级。客户端应将优先级 0-4 视为普通优先级等级，将优先级 5-9 视为加速优先级等级。

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setpriority”一节第 653 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到一条消息，并将消息优先级输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @priority integer;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @priority = ml_qa_getpriority( @msgid );
    message 'a message with priority ' || @priority || ' has been received';
    commit;
end
```

ml_qa_getredelivered

返回一个指明此消息是否以前接收过但未进行确认的值。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

以 BIT 表示重新传送的值。值 **1** 表示正在重新传送消息；值 **0** 表示现在未重新传送消息。

注释

如果某消息以前被接收过但未进行确认，则可以重新传送该消息。例如，该消息已被接收，但接收该消息的应用程序还未处理完消息内容就崩溃了。这种情况下，QAnywhere 将消息标记为重新传送，以警告接收方注意该消息可能只处理了一部分。

例如，假定消息的接收过程分三个步骤完成：

1. 由使用非事务性 QAnywhere Manager 的应用程序接收消息。
2. 该应用程序将消息内容和消息 ID 写入名为 T1 的数据库表，然后提交更改。
3. 应用程序确认消息。

如果应用程序在第 1 步和第 2 步之间或第 2 步和第 3 步之间发生故障，则消息将在应用程序重新启动时重新传送。

如果在第 1 步和第 2 步之间发生故障，则应通过运行第 2 步和第 3 步来处理重新传送的消息。如果在第 2 步和第 3 步之间发生故障，则消息已得到处理，仅需您确认。

要确定应用程序发生故障时发生了何种情况，可让应用程序调用 ml_qa_getredelivered，以检查以前是否重新传送过此消息。只有重新传送的消息需要在表 T1 中查找。比起让应用程序访问已接收消息的消息 ID 以查看此消息是否在表 T1 中，这样做效率更高，原因是应用程序故障很少发生。

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到一条消息；如果该消息以前传送过但未收到，则将消息 ID 输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @redelivered bit;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @redelivered = ml_qa_getredelivered( @msgid );
    if @redelivered = 1 then
        message 'message with message ID ' || @msgid || ' has been
redelivered';
    end if;
    commit;
end
```

ml_qa_getreplytoaddress

返回此消息的回复地址。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

VARCHAR(128) 形式的回复地址。

注释

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setreplytoaddress”一节第 654 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，如果接收的消息有回复地址，则将内容为 '消息已收到' 的消息发送到回复地址：

```
begin
    declare @msgid varchar(128);
    declare @rmsgid varchar(128);
    declare @replytoaddr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @replytoaddr = ml_qa_getreplytoaddress( @msgid );
    if @replytoaddr is not null then
        set @rmsgid = ml_qa_createmessage();
        call ml_qa_settextcontent( @rmsgid, 'message received' );
        call ml_qa_putmessage( @rmsgid, @replytoaddr );
    end if;
    commit;
end
```

ml_qa_gettimestamp

返回消息创建时间。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

TIMESTAMP 形式的消息创建时间。

注释

Timestamp 标头字段包含消息的创建时间。时间为协调通用时间 (UTC)。它不是消息实际传输的时间，因为事务或其它客户端消息排队可能导致实际发送推迟。

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到一条消息，并将消息创建时间输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @ts timestamp;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @ts = ml_qa_gettimestamp( @msgid );
    message 'message received with create time: ' || @ts ;
    commit;
end
```

ml_qa_setexpiration

设置消息到期时间。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	到期时间	TIMESTAMP

注释

接收消息后就可以读取此标头，一直到执行回退或提交；执行回退或提交后就不能再读取此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getexpiration”一节第 646 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，创建一条消息，如果在随后的 3 天内未传送，则该消息到期：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setexpiration( @msgid, dateadd( day, 3, current timestamp ) );
    call ml_qa_settextcontent( @msgid, 'time-limited offer' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setinreplytoid

设置此消息的回复 ID。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	回复 ID	VARCHAR(128)

注释

此回复 ID 与电子邮件系统跟踪回复所使用的回复 ID 类似。

通常将回复 ID 设置为此消息将回复的消息的消息 ID（如果有）。

客户端可使用 InReplyToID 标头字段，将一条消息与另一条消息相链接。典型应用是将响应消息与其请求消息相链接。

消息发送后就不能再更改此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getinreplytoid”一节第 647 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，收到包含回复地址的消息时，创建和发送一条在回复 ID 中包含消息 ID 的回复消息：

```
begin
    declare @msgid varchar(128);
    declare @rmsgid varchar(128);
    declare @replyaddr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @replyaddr = ml_qa_getreplyaddress( @msgid );
    if @replyaddr is not null then
        set @rmsgid = ml_qa_createmessage();
        call ml_qa_settextcontent( @rmsgid, 'message received' );
        call ml_qa_setinreplytoid( @rmsgid, @msgid );
        call ml_qa_putmessage( @rmsgid, @replyaddr );
    end if;
    commit;
end
```

ml_qa_setpriority

设置消息优先级。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	优先级	INTEGER

注释

QAnywhere API 定义十种级别的优先级值，0 代表最低优先级，9 代表最高优先级。客户端应将优先级 0-4 视为普通优先级等级，将优先级 5-9 视为加速优先级等级。

消息发送后就不能再更改此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getpriority”一节第 648 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

下面的示例发送了一条高优先级消息：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setpriority( @msgid, 9 );
    call ml_qa_settextcontent( @msgid, 'priority content' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setreplytoaddress

设置消息回复地址。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	回复地址	VARCHAR(128)

注释

消息发送后就不能再更改此标头。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getreplytoaddress”一节第 650 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，将回复地址添加到某消息。消息接收者随后可以使用该回复地址创建一个回复。

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setreplytoaddress( @msgid, 'myaddress' );
    call ml_qa_settextcontent( @msgid, 'some content' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

消息属性

可使用以下存储过程获取和设置自定义消息属性，或获取预定义消息属性。

请参见“消息属性”一节第 687 页。

ml_qa_getbooleanproperty

返回 SQL BIT 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

BIT 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setbooleanproperty”一节第 662 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将布尔型属性 mybooleanproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop bit;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getbooleanproperty( @msgid, 'mybooleanproperty' );
    message 'message property mybooleanproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getbyteproperty

返回 SQL TINYINT 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

TINYINT 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setbyteproperty”一节第 663 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将字节型属性 mybyteproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop tinyint;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getbyteproperty( @msgid, 'mybyteproperty' );
    message 'message property mybyteproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getdoubleproperty

返回 SQL DOUBLE 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

DOUBLE 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setdoubleproperty”一节第 664 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将双精度型属性 mydoubleproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop double;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getdoubleproperty( @msgid, 'mydoubleproperty' );
```



```
    message 'message property mydoubleproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getfloatproperty

返回 SQL FLOAT 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

FLOAT 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setfloatproperty”一节第 665 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将浮点型属性 myfloatproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop float;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getfloatproperty( @msgid, 'myfloatproperty' );
    message 'message property myfloatproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getintproperty

返回 SQL INTEGER 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

INTEGER 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setintproperty”一节第 666 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将整型属性 myintproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop integer;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getintproperty( @msgid, 'myintproperty' );
    message 'message property myintproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getlongproperty

返回 SQL BIGINT 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

BIGINT 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setlongproperty”一节第 666 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

ml_qa_getpropertynames

检索指定消息的属性名称。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

注释

此存储过程将打开一个涉及指定消息所有属性名称的结果集。消息 ID 参数必须是已接收消息的消息 ID。

该结果集是一个类型为 VARCHAR(128) 的单列，其中每行都包含一个消息属性的名称。将不返回 QAnywhere 保留属性名称（前缀为 "ias_" 或 "QA" 的属性名称）。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

下面的示例对消息 ID 为 msgid 的消息的整个属性名称结果集声明一个游标。然后获取地址为 clientid \queuename 的消息；接着打开一个游标访问该消息的属性名称；最后读取下一个属性名称。

```
begin
 declare prop_name_cursor cursor for
```



```
 call ml_qa_getpropertynames( @msgid );
declare @msgid varchar(128);
declare @name varchar(128);

set @msgid = ml_qa_getmessage( 'clientid\queuename' );
open prop_name_cursor;
lp: loop
 fetch next prop_name_cursor into name;
 if sqlcode <> 0 then leave lp end if;
 ...
end loop;
close prop_name_cursor;
end
```

ml_qa_getshortproperty

返回 SQL SMALLINT 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

SMALLINT 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setshortproperty”一节第 667 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将短整型属性 myshortproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop smallint;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getshortproperty( @msgid, 'myshortproperty' );
```

```
    message 'message property myshortproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getstringproperty

返回 SQL LONG VARCHAR 数据类型形式的指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)

返回值

LONG VARCHAR 形式的属性值。

注释

如果消息属性值超出范围，则会出现 SQL 错误 SQLSTATE 22003。

接收消息后就可以读取此属性，一直到执行回退或提交；执行回退或提交后就不能再读取此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setstringproperty”一节第 668 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在下面的示例中，收到一条消息，并将字符串型属性 mystringproperty 的值输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @prop long varchar;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getstringproperty( @msgid, 'mystringproperty' );
    message 'message property mystringproperty is set to ' || @prop;
    commit;
end
```

ml_qa_setbooleanproperty

用 SQL BIT 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	BIT

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getbooleanproperty”一节第 655 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置布尔型属性 mybooleanproperty1 和 mybooleanproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty1', 0 );
    call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty2', 1 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setbyteproperty

用 SQL TINYINT 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	TINYINT

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getbyteproperty”一节第 656 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置字节型属性 mybyteproperty1 和 mybyteproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbyteproperty( @msgid, 'mybyteproperty1', 0 );
    call ml_qa_setbyteproperty( @msgid, 'mybyteproperty2', 255 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setdoubleproperty

用 SQL DOUBLE 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	DOUBLE

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getdoubleproperty”一节第 657 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置双精度型属性 mydoubleproperty1 和 mydoubleproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty1', -12.34e-56 );
    call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty2', 12.34e56 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setfloatproperty

用 SQL FLOAT 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	FLOAT

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getfloatproperty”一节第 658 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置浮点型属性 myfloatproperty1 和 myfloatproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
    call ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```


ml_qa_setintproperty

用 SQL INTEGER 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	INTEGER

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getintproperty”一节第 658 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置整数型属性 myintproperty1 和 myintproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setintproperty( @msgid, 'myintproperty1', -1234567890 );
    call ml_qa_setintproperty( @msgid, 'myintproperty2',  1234567890 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setlongproperty

用 SQL BIGINT 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

项	说明	注释
2	属性名称	VARCHAR(128)
3	属性值	BIGINT

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getlongproperty”一节第 659 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置长整型属性 mylongproperty1 和 mylongproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setlongproperty( @msgid, 'mylongproperty1',
-1234567890987654321 );
    call ml_qa_setlongproperty( @msgid, 'mylongproperty2',
1234567890987654321 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setshortproperty

用 SQL SMALLINT 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	SMALLINT

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getshortproperty”一节第 661 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置短整型属性 myshortproperty1 和 myshortproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setshortproperty( @msgid, 'myshortproperty1', -12345 );
    call ml_qa_setshortproperty( @msgid, 'myshortproperty2',  12345 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setstringproperty

用 SQL LONG VARCHAR 数据类型设置指定消息属性。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	属性名称	VARCHAR(128)
3	属性值	LONG VARCHAR

注释

消息发送后就不能再更改此属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getstringproperty”一节第 662 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “自定义消息属性”一节第 689 页

示例

在以下示例中，创建一条消息，设置字符串型属性 mystringproperty1 和 mystringproperty2，然后将消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setstringproperty( @msgid, 'mystringproperty1', 'c:\\temp' );
    call ml_qa_setstringproperty( @msgid, 'mystringproperty2',  'first line
\nsecond line' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

消息内容

可使用下面的存储过程获取和设置消息内容。

ml_qa_getbinarycontent

返回二进制消息的消息内容。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

LONG BINARY 形式的消息内容。

如果消息有文本内容，而非二进制内容，则此存储过程返回空值。

接收消息后就可以读取此内容，一直到执行回退或提交；执行回退或提交后就不能再读取此内容。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setbinarycontent”一节第 671 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “ml_qa_getcontentclass”一节第 670 页

示例

在下面的示例中，将消息的加密内容解密并输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @content long binary;
```

```
    declare @plaintext long varchar;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @content = ml_qa_getbinarycontent( @msgid );
    set @plaintext = decrypt( @content, 'mykey' );
    message 'message content decrypted: ' || @plaintext;
    commit;
end
```

ml_qa_getcontentclass

返回消息类型（文本或二进制）。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

INTEGER 形式的内容分类。

返回值可以是：

- **1** 指明消息内容是二进制内容，应使用存储过程 ml_qa_getbinarycontent 读取。
- **2** 指明消息内容是文本内容，应使用存储过程 ml_qa_gettextcontent 读取。

注释

接收消息后就可以读取此内容，一直到执行回退或提交；执行回退或提交后就不能再读取此内容。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “ml_qa_getbinarycontent”一节第 669 页
- “ml_qa_gettextcontent”一节第 671 页

示例

在下面的示例中，收到一条消息，并将其内容输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @contentclass integer;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @contentclass = ml_qa_getcontentclass( @msgid );
    if @contentclass = 1 then
        message 'message  binary is ' || ml_qa_getbinarycontent( @msgid );
    elseif @contentclass = 2 then
        message 'message  text is ' || ml_qa_gettextcontent( @msgid );
    end if;
```



```
    commit;
end
```

ml_qa_gettextcontent

返回文本消息的消息内容。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。

返回值

LONG VARCHAR 形式的文本内容。

如果消息有二进制内容，而非文本内容，则此存储过程返回空值。

注释

接收消息后就可以读取此内容，一直到执行回退或提交；执行回退或提交后就不能再读取此内容。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_settextcontent”一节第 672 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页
- “ml_qa_getcontentclass”一节第 670 页

示例

在下面的示例中，将消息内容输出到数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    declare @content long binary;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @content = ml_qa_gettextcontent( @msgid );
    message 'message content: ' || @content ;
    commit;
end
```

ml_qa_setbinarycontent

设置消息的二进制内容。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	内容	LONG BINARY

消息发送后就不能再更改此内容。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getbinarycontent”一节第 669 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，创建并发送一条含有加密内容的消息：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbinarycontent( @msgid, encrypt( 'my secret message',
'mykey' ) );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_settextcontent

设置消息的文本内容。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	内容	LONG VARCHAR

注释

消息发送后就不能再更改此内容。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_gettextcontent”一节第 671 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

在下面的示例中，创建一条消息，然后用给定内容进行设置：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'my simple message' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

消息存储库属性

可使用下面的存储过程获取和设置客户端消息存储库属性。

有关消息存储库属性的详细信息，请参见“客户端消息存储库属性”一节第 26 页。

ml_qa_getstoreproperty

返回客户端消息存储库属性。

参数

项	说明	注释
1	属性名称	VARCHAR(128)

返回值

LONG VARCHAR 形式的属性值。

注释

可从此客户端消息存储库的每个连接读取其属性。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_setstoreproperty”一节第 674 页

示例

下面的示例获取了此消息存储库的当前同步策略并将其输出到数据库服务器消息窗口：

```
begin
    declare @policy varchar(128);
    set @policy = ml_qa_getstoreproperty( 'policy' );
    message 'the current policy for synchronizing this message store is ' ||
@policy;
end
```

ml_qa_setstoreproperty

设置客户端消息存储库属性。

参数

项	说明	注释
1	属性名称	VARCHAR(128)

项	说明	注释
2	属性值	SMALLINT

注释

可从此客户端消息存储库的每个连接读取其属性。这些值将被同步到服务器，在服务器上同样可以将这些值用于传输规则。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getstoreproperty”一节第 674 页

示例

下面的示例将消息存储库的同步策略设置为 [自动]：

```
begin
    call ml_qa_setstoreproperty( 'policy', 'automatic' );
    commit;
end
```

消息管理

可使用下面的存储过程管理 QAnywhere 客户端事务。

ml_qa_createmessage

返回新消息的消息 ID。

返回值

新消息的消息 ID。

注释

此存储过程用于创建消息。消息创建后，即可将内容、属性和标头与此消息关联，然后将其发送。

可使用以 ml_qa_set 开头的任何 QAnywhere 存储过程来关联内容、属性和标头。例如，使用 ml_qa_setbinarycontent（或 ml_qa_settextcontent）创建二进制（或文本）消息。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “消息标头”一节第 646 页
- “消息属性”一节第 655 页
- “消息内容”一节第 669 页

示例

以下示例创建一条消息，设置该消息内容，然后将该消息发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'some content' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_getmessage

返回排队等候给定地址的下一条消息的消息 ID，在有消息排队之前会一直阻塞。

参数

项	说明	注释
1	地址	VARCHAR(128)

返回值

VARCHAR(128) 形式的消息 ID。

如果没有消息排队等候此地址，则返回空值。

注释

此存储过程用于同步检查是否有消息在等候指定的 QAnywhere 消息地址。如果想在有消息等候指定的 QAnywhere 地址时异步调用 SQL 过程，则使用监听器。

在有消息排队之前，此存储过程将一直阻塞。

有关避免阻塞的信息，请参见“ml_qa_getmessagenowait”一节第 677 页
或“ml_qa_getmessagetimeout”一节第 678 页。

在当前事务提交之前，与返回的消息 ID 对应的消息将不被视为已收到。一旦接收被提交，此 QAnywhere API 或其它任何 QAnywhere API 就不能再次收到该消息。同样，对当前事务的回退表示未收到该消息，这样，对 ml_qa_getmessage 的后续调用可能会返回同一个消息 ID。

在对当前事务执行提交或回退之前，可由各种 ml_qa_get 存储过程读取已接收消息的属性和内容。一旦对当前事务执行了提交或回退，就不能再读取消息数据了。在提交之前，应以表格形式或在 SQL 变量中存储您所需要的任何消息数据。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getmessagenowait”一节第 677 页
- “ml_qa_getmessagetimeout”一节第 678 页
- “消息标头”一节第 646 页
- “消息属性”一节第 655 页
- “消息内容”一节第 669 页

示例

下面的示例显示了发送到地址 myaddress 的所有消息的内容：

```
begin
    declare @msgid varchar(128);
    loop
        set @msgid = ml_qa_getmessage( 'myaddress' );
        message 'a message with content ' || ml_qa_gettextcontent( @msgid )
|| ' has been received';
        commit;
    end loop;
end
```

ml_qa_getmessagenowait

返回当前排队等候给定地址的下一条消息的消息 ID。

参数

项	说明	注释
1	地址	VARCHAR(128)

返回值

VARCHAR(128) 形式的消息 ID。

返回排队等候给定地址的下一条消息的消息 ID。如果没有消息排队等候此地址，则返回空值。

注释

此存储过程用于同步检查是否有消息在等候指定的 QAnywhere 消息地址。如果想在有消息等候指定的 QAnywhere 地址时异步调用 SQL 过程，则使用监听器。

有关在有消息可用之前一直阻塞的信息，请参见“ml_qa_getmessage”一节第 676 页和“ml_qa_getmessagetimeout”一节第 678 页。

在当前事务提交之前，与返回的消息对应的消息将不被视为已收到。一旦接收被提交，此 QAnywhere API 或其它任何 QAnywhere API 就不能再次收到该消息。同样，对当前事务的回退表示未收到该消息，这样，对 ml_qa_getmessage 的后续调用可能会返回同一个消息 ID。

在对当前事务执行提交或回退之前，可由各种 ml_qa_get 存储过程读取已接收消息的属性和内容。一旦对当前事务执行了提交或回退，就不能再读取消息数据了。在提交之前，应以表格形式或在 SQL 变量中存储您所需要的任何消息数据。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “QAnywhere 消息地址”一节第 63 页
- “监听器”一节 《MobiLink - 服务器启动的同步》
- “ml_qa_getmessagetimeout”一节第 678 页
- “消息标头”一节第 646 页
- “消息属性”一节第 655 页
- “消息内容”一节第 669 页

示例

下面的示例显示了在地址 myaddress 处等候的所有消息直到都被读取之前的内容（读取最后一条消息后提交通常比读取每条消息后提交的效率更高）：

```
begin
    declare @msgid varchar(128);
    loop
        set @msgid = ml_qa_getmessagenowait( 'myaddress' );
        if @msgid is null then leave end if;
        message 'a message with content ' || ml_qa_gettextcontent( @msgid )
|| ' has been received';
    end loop;
    commit;
end
```

ml_qa_getmessagetimeout

等待指定超时期过后，返回排队等候给定地址的下一条消息的消息 ID。

参数

项	说明	注释
1	地址	VARCHAR(128)
2	以毫秒为单位的超时时间	INTEGER

返回值

VARCHAR(128) 形式的消息 ID。

如果在超时期内没有消息排队等候此地址，则返回空值。

注释

此存储过程用于同步检查是否有消息在等候指定的 QAnywhere 消息地址。如果想在有消息等候指定的 QAnywhere 地址时异步调用 SQL 过程，则使用监听器。

在当前事务提交之前，与返回的消息对应的消息将不被视为已收到。一旦接收被提交，此 QAnywhere API 或其它任何 QAnywhere API 就不能再次收到该消息。同样，对当前事务的回退表示未收到该消息，这样，对 ml_qa_getmessage 的后续调用可能会返回同一个消息 ID。

在对当前事务执行提交或回退之前，可由各种 ml_qa_get 存储过程读取已接收消息的属性和内容。一旦对当前事务执行了提交或回退，就不能再读取消息数据了。在提交之前，应以表格形式或在 SQL 变量中存储您所需要的任何消息数据。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_getmessage”一节第 676 页
- “ml_qa_getmessagenowait”一节第 677 页

示例

下面的示例将发送到地址 myaddress 的所有消息的内容输出到数据库服务器消息窗口；如果未收到任何消息，则每隔 10 秒钟更新一次数据库服务器消息窗口：

```
begin
    declare @msgid varchar(128);
    loop
        set @msgid = ml_qa_getmessagetimeout( 'myaddress', 10000 );
        if @msgid is null then
            message 'waiting for a message...';
        else
          message 'a message with content ' || ml_qa_gettextcontent( @msgid )
|| ' has been received';
        commit;
        end if;
    end loop;
end
```

ml_qa_grant_messaging_permissions

授予其他用户使用 QAnywhere 存储过程的权限。

参数

项	说明	注释
1	数据库用户 ID	VARCHAR(128)

注释

只有具备 DBA 特权的用户才自动具备执行 QAnywhere 存储过程的权限。其他用户必须通过具备 DBA 特权的用户运行此存储过程来授权。

此过程将用户添加到名为 ml_qa_message_group 的组，然后授予他们对所有 QAnywhere 存储过程的执行权限。

另请参见

- “设置 SQL 应用程序”一节第 61 页

示例

例如，要向数据库 ID 为 user1 的用户授予消息传递权限，可执行下面的 SQL 代码：

```
call dbo.ml_qa_grant_messaging_permissions( 'user1' )
```

ml_qa_listener_queue

创建名为 **ml_qa_listener_*queue*** （其中 *queue* 是消息队列名）的存储过程以异步接收消息。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 QAnywhere 监听器获取消息 ID。

注释

注意
此存储过程与所有其它 QAnywhere 存储过程的不同之处在于此过程并非事先提供。如果您创建一个名为 **ml_qa_listener_*queue*** 的存储过程（其中 *queue* 是消息队列），QAnywhere 随后即会使用它。

尽管可在连接上同步接收各消息，但异步接收消息通常更方便。可创建一个在有消息在特定地址上排队等候时便被调用的存储过程。此过程的名称必须为 **ml_qa_listener_*queue***，其中 *queue* 是消息队列。如果已有此过程，则只要有消息在给定地址上排队，便会调用此过程。

此过程从另一个不同的连接调用。只要执行此过程时不出现 SQL 错误，就会自动确认和提交消息。

在此过程中不要执行提交或回退。

队列名是 QAnywhere 地址的一部分。有关详细信息，请参见“QAnywhere 消息地址”一节第 63 页。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “异步接收消息”一节第 75 页
- “同步接收消息”一节第 74 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

下面的示例创建一个过程，只要有消息在名为 executesql 的地址上排队，便会调用此过程。在本示例中，此过程假设消息内容是可对当前数据库执行的 SQL 语句。

```
CREATE PROCEDURE ml_qa_listener_executesql(IN @msgid VARCHAR(128))
begin
 DECLARE @execstr LONG VARCHAR;
 SET @execstr = ml_qa_gettextcontent( @msgid );
 EXECUTE IMMEDIATE @execstr;
end
```

ml_qa_putmessage

发送消息。

参数

项	说明	注释
1	消息 ID	VARCHAR(128)。可从 ml_qa_createmessage 或 ml_qa_getmessage 获取消息 ID。
2	地址	VARCHAR(128)

注释

指定的消息 ID 必须是先前使用 ml_qa_createmessage 创建的消息 ID。只有在调用 ml_qa_putmessage 之前与该消息 ID 关联的内容、属性和标头才能随该消息发送。调用 ml_qa_putmessage 之后添加的信息均被忽略。

消息必须在提交之后才能实际排队等候发送。

另请参见

- “设置 SQL 应用程序”一节第 61 页
- “ml_qa_createmessage”一节第 676 页
- “ml_qa_getmessage”一节第 676 页

示例

下例中，创建一条含有内容 'a simple message' 的消息，然后将其发送到地址 clientid\queuename：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'a simple message' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_triggersendreceive

触发与 MobiLink 服务器之间的消息同步。

注释

消息同步通常由 QAnywhere 代理处理。但如果同步策略是 [要求时]，则由应用程序负责触发消息同步。可使用此存储过程完成该任务。在提交当前事务之前，该触发器不生效。

另请参见

- “设置 SQL 应用程序”一节第 61 页

示例

在下面的示例中，发送一条消息，随后立即启动对该消息的传输：

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'my simple message' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    call ml_qa_triggersendreceive();
    commit;
end
```


消息标头和属性

目录

消息标头

所有 QAnywhere 消息都支持相同的标头字段集。标头字段包含客户端和提供程序用来标识和路由消息的值。

以下消息标头是预定义的。其使用方法取决于您所拥有的客户端应用程序的类型。

- **Message ID** 只读。新消息的消息 ID。仅在发送消息后，此标头才会有值。请参见：
  - .NET API： “MessageID 属性”一节第 308 页
  - C++ API： “getMessageID 函数”一节第 466 页 和 “setMessageID 函数”一节第 476 页
  - Java API： “getMessageID 方法”一节第 582 页
  - SQL API： “ml_qa_createmessage”一节第 676 页 和 “ml_qa_getmessage”一节第 676 页
- **Message creation timestamp** 只读。Timestamp 标头字段包含消息的创建时间。时间为协调通用时间 (UTC)。它不是消息实际传输的时间，因为事务或其它客户端消息排队可能导致实际发送推迟。可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。请参见：
  - .NET API： “Timestamp 属性”一节第 310 页
  - C++ API： “getTimestamp 函数”一节第 470 页 和 “setTimestamp 函数”一节第 479 页
  - Java API： “getTimestamp 方法”一节第 586 页
  - SQL API： “ml_qa_gettimestamp”一节第 651 页
- **Reply-to address** 读写。如果回复地址不存在，则以 VARCHAR(128) 或空值表示。可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。请参见：
  - .NET API： “ReplyToAddress 属性”一节第 310 页
  - C++ API： “getReplyToAddress 函数”一节第 468 页 和 “setReplyToAddress 函数”一节第 477 页
  - Java API： “getReplyToAddress 方法”一节第 585 页 和 “setReplyToAddress 方法”一节第 592 页
  - SQL API： “ml_qa_getreplytoaddress”一节第 650 页 和 “ml_qa_setreplytoaddress”一节第 654 页
- **Message address** 只读。以 VARCHAR(128) 表示 QAnywhere 消息地址。QAnywhere 消息地址的形式为 *id\queue-name*。可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。请参见：
  - .NET API： “Address 属性”一节第 307 页
  - C++ API： “getAddress 函数”一节第 462 页
  - Java API： “getAddress 方法”一节第 577 页
  - SQL API： “ml_qa_getaddress”一节第 646 页
- **Redelivered state of message** 只读。以 BIT 表示重新传送的值。值 1 表示正在重新传送消息；值 0 表示现在未重新传送消息。

  如果某消息以前被接收过但未进行确认，则可以重新传送该消息。例如，消息虽已收到，但接收消息的应用程序在崩溃前没有完成对消息内容的处理。这种情况下，QAnywhere 将消息标记为重新传送，以警告接收方注意消息可能只处理了一部分。

  例如，假定消息的接收过程分为三个步骤：

1. 由使用非事务性 QAnywhere Manager 的应用程序接收消息。
2. 该应用程序将消息内容和消息 ID 写入名为 T1 的数据库表，然后提交更改。
3. 应用程序确认此消息。

如果应用程序在步骤 1 和 2 之间或步骤 2 和 3 之间发生故障，则消息将在应用程序重新启动时重新传送。

如果在第 1 步和第 2 步之间发生故障，则应通过运行第 2 步和第 3 步来处理重新传送的消息。如果在第 2 步和第 3 步之间发生故障，则消息已得到处理，仅需您确认。

要确定应用程序发生故障时发生了何种情况，可让应用程序调用 ml_qa_getredelivered，以检查以前是否重新传送过此消息。只有重新传送的消息需要在表 T1 中查找。比起让应用程序访问已接收消息的消息 ID 以查看此消息是否在表 T1 中，这样做效率更高，原因是应用程序故障很少发生。

可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。

请参见：

- .NET API： “Redelivered 属性”一节第 309 页
- C++ API： “getRedelivered 函数”一节第 468 页 和 “setRedelivered 函数”一节第 477 页
- Java API： “getRedelivered 方法”一节第 584 页
- SQL API： “ml_qa_getredelivered”一节第 649 页

- **Expiration of message** 只读（只有在 SQL API 中是读写）。以 TIMESTAMP 表示的到期时间。如果没有到期时间，则返回空值。如果消息未在指定时间内由预期接收者接收，则它将过期。然后，可使用缺省的 QAnywhere 删除规则删除此消息。可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。请参见：

  - .NET API： “Expiration 属性”一节第 307 页
  - C++ API： “getExpiration 函数”一节第 464 页
  - Java API： “getExpiration 方法”一节第 579 页
  - SQL API： “ml_qa_getexpiration”一节第 646 页 和 “ml_qa_setexpiration”一节第 652 页

- **Priority of message** 读写。QAnywhere API 定义十种级别的优先级值，0 代表最低优先级，9 代表最高优先级。客户端应将优先级 0-4 视为普通优先级等级，将优先级 5-9 视为加速优先级等级。可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。请参见：

  - .NET API： “Priority 属性”一节第 309 页
  - C++ API： “getPriority 函数”一节第 467 页
  - Java API： “getPriority 方法”一节第 582 页
  - SQL API： “ml_qa_getpriority”一节第 648 页

- **Message ID of a message for which this message is a reply** 读写。以 VARCHAR(128) 表示的回复 ID。客户端可使用 InReplyToID 标头字段，将一条消息与另一条消息相链接。典型应用

是将响应消息与其请求消息相链接。回复 ID 是作为此消息回复对象的消息的 ID。可以在收到消息后以及发生回退或提交之前读取此标头；之后则无法读取。请参见：

- .NET API： “InReplyToID 属性”一节第 308 页
- C++ API： “getInReplyToID 函数”一节第 465 页
- Java API： “getInReplyToID 方法”一节第 580 页
- SQL API： “ml_qa_getinreplytoid”一节第 647 页

某些消息标头可以在传输规则中使用。请参见“规则引擎定义的变量”一节第 767 页。

另请参见

- .NET API： “QAMessage 成员”一节第 305 页
- C++ API： “QAMessage 类”一节第 458 页
- Java API： “QAMessage 接口”一节第 575 页
- SQL API： “消息标头”一节第 646 页

消息属性

每条消息都包含用于支持应用程序定义的属性值的内置工具。这些消息属性允许您执行应用程序定义的消息过滤。

消息属性是名称值对，可选择将其插入消息以提供结构。例如，在 .NET API 中，由常量 MessageProperties.ORIGINATOR 标识的预定义消息属性 ias_Originator 将提供发送消息的消息存储库 ID。消息属性可在传输规则中使用，以确定消息是否适合传输。

消息属性的类型共有两种：

- **预定义的消息属性** 这些消息属性始终用 ias_ 或 IAS_ 作为前缀。
- **自定义消息属性** 这些消息属性可由您进行定义。不能以 ias_ 或 IAS_ 作为其前缀。

在这两种情况下，均可使用 get 和 set 方法来访问消息存储库属性，并可将预定义属性或自定义属性的名称作为第一个参数来进行传递。

请参见“管理消息属性”一节第 689 页。

预定义的消息属性

为方便起见，预定义了某些消息属性。预定义的消息属性可以读取但不得设置。预定义的消息属性有：

- **ias_Adapters** 用于网络状态通知消息，是一个可用于连接到 MobiLink 服务器的网络适配器的列表。此列表为字符串，由竖线分隔。
- **ias_DeliveryCount** Int.到目前为止尝试传送消息的次数。
- **ias_MessageType** Int.指示消息的类型。消息类型可以是：

值	消息类型	说明
0	REGULAR	如果未设置消息的 ias_MessageType 属性，则消息为常规消息。
13	PUSH_NOTIFICATION	当收到来自服务器的推式通知时，PUSH_NOTIFICATION 类型的消息将被发送到系统队列。请参见“有关推式通知的通知”一节第 65 页。
14	NETWORK_STATUS_NOTIFICATION	网络状态发生变化时，此类型的消息将被发送到系统队列。请参见“网络状态通知”一节第 64 页。

- **ias_RASNames** 字符串。用于网络状态通知消息，是一个可用于连接到 MobiLink 服务器的 RAS 条目名的列表。此列表用竖线分隔。

- **ias_NetworkStatus** Int.用于网络状态通知消息，表示网络连接的状态。如果已连接，则值为 1；否则，值为 0。
- **ias_Originator** 字符串。消息发出方的消息存储库 ID。
- **ias_Status** Int.消息的当前状态。SQL API 不支持此属性。其值可以是：

状态代码	说明
1	待执行 - 消息已发送但未被接收。
10	正在接收 - 消息处于正在接收过程中，或者已接收但未确认。
20	最终 - 消息已达到最终状态。
30	已到期 - 消息在达到到期时间之前未被接收。
40	已取消 - 消息已取消。
50	无法收到 - 消息格式错误或尝试传送时失败次数过多。
60	已接收 - 消息已接收并得到确认。

存在用于表示状态值的常量。请参见：

- .NET API： “StatusCodes 枚举”一节第 331 页
- C++ API： “StatusCodes 类”一节第 492 页
- Java API： “StatusCodes 接口”一节第 605 页

- **ias_StatusTime** 消息转为其当前状态的时间。它的单位符合平台要求。它为本地时间。在 C++ API 中，对于 Windows 和 PocketPC 平台，时间戳为 SYSTEMTIME，它被转换为 FILETIME 并复制到 qa_long 值中。SQL API 不支持此属性。

API	此属性返回……
.NET	DateTime
C++	string
Java	java.util.Date 对象

消息属性常量

用于 .NET、C++ 和 Java 的 QAnywhere API 提供指定消息属性的常量。请参见：

- .NET API： “MessageProperties 成员”一节第 215 页
- C++ API： “MessageProperties 类”一节第 388 页
- Java API： “MessageProperties 接口”一节第 499 页

自定义消息属性

QAnywhere 允许使用 C++、Java 或 .NET API 定义消息属性。自定义消息属性允许创建与对象关联的名称值对。例如：

```
msg.SetStringProperty("Product", "widget");
msg.SetFloatProperty("Price",1.00);
msg.SetIntProperty("Quantity",10);
```

消息属性名称不区分大小写。可以使用字母、数字和下划线序列，但第一个字符必须是字母。以下名称将被保留，不能用作消息属性名：

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE
- 所有以 **ias_** 开头的名称

管理消息属性

以下 QAMessage 方法可用于管理消息属性。

可以获取和设置自定义属性，但只应获得预定义属性。

管理消息属性的 .NET 方法

- Object GetProperty(String name)
- void SetProperty(String name, Object value)
- boolean GetBooleanProperty(String name)
- void SetBooleanProperty(String name, boolean value)
- byte GetByteProperty(String name)
- void SetByteProperty(String name, byte value)
- short GetShortProperty(String name)
- void SetShortProperty(String name, short value)
- int GetIntProperty(String name)
- void SetIntProperty(String name, int value)
- long GetLongProperty(String name)
- void SetLongProperty(String name, long value)
- float GetFloatProperty(String name)
- void SetFloatProperty(String name, float value)
- double GetDoubleProperty(String name)
- void SetDoubleProperty(String name, double value)
- String GetStringProperty(String name)
- void SetStringProperty(String name, String value)
- IEnumerator GetPropertyNames()
- void ClearProperties()
- PropertyType GetPropertyType(string propName)
- bool PropertyExists(string propName)

请参见“QAMessage 接口”一节第 304 页。

管理消息属性的 C++ 方法

- qa_bool getBooleanProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanProperty(qa_const_string name, qa_bool value)
- qa_bool getByteProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteProperty(qa_const_string name, qa_byte value)
- qa_bool getShortProperty(qa_const_string name, qa_short * value)
- qa_bool setShortProperty(qa_const_string name, qa_short value)
- qa_bool getIntProperty(qa_const_string name, qa_int * value)
- qa_bool setIntProperty(qa_const_string name, qa_int value)
- qa_bool getLongProperty(qa_const_string name, qa_long * value)
- qa_bool setLongProperty(qa_const_string name, qa_long value)
- qa_bool getFloatProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleProperty(qa_const_string name, qa_double value)
- qa_int getStringProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringProperty(qa_const_string name, qa_const_string value)
- void QAMessage::clearProperties()
- qa_short QAMessage::getPropertyType(qa_const_string name)
- qa_bool QAMessage::propertyExists(qa_const_string name)

请参见“QAMessage 类”一节第 458 页。

管理消息属性的 Java 方法

- void clearProperties()
- boolean getBooleanProperty(String name)
- void setBooleanProperty(String name, boolean value)
- byte getByteProperty(String name)
- void setByteProperty(String name, byte value)
- double getDoubleProperty(String name)
- void setDoubleProperty(String name, double value)
- java.util.Date getExpiration() void setFloatProperty(String name, float value)
- float getFloatProperty(String name)
- int getIntProperty(String name)
- void setIntProperty(String name, int value)
- long getLongProperty(String name)
- void setLongProperty(String name, long value)
- Object getProperty(String name)
- void setProperty(String name, Object value)
- java.util.Enumeration getPropertyNames()
- short getPropertyType(String name)
- short getShortProperty(String name)
- void setShortProperty(String name, short value)
- String getStringProperty(String name)
- void setStringProperty(String name, String value)
- boolean propertyExists(String name)

请参见“QAMessage 接口”一节第 575 页。

管理消息属性的 SQL 存储过程

- ml_qa_getbooleanproperty
- ml_qa_getbyteproperty
- ml_qa_getdoubleproperty
- ml_qa_getfloatproperty
- ml_qa_getintproperty
- ml_qa_getlongproperty
- ml_qa_getpropertynames
- ml_qa_getshortproperty
- ml_qa_getstringproperty
- ml_qa_setbooleanproperty
- ml_qa_setbyteproperty
- ml_qa_setdoubleproperty
- ml_qa_setfloatproperty
- ml_qa_setfloatproperty
- ml_qa_setintproperty
- ml_qa_setlongproperty
- ml_qa_setshortproperty
- ml_qa_setstringproperty

请参见“消息属性”一节第 655 页。

示例

```
// C++ example.
QAManagerFactory factory;
QAManager * mgr = factory->createQAManager( NULL );
mgr->open(AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT);
QAMessage * msg = mgr->createTextMessage();
msg->setStringProperty( "tm_Subject", "Some message subject." );
mgr->putMessage( "myqueue", mgr );

// C# example.
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(null);
mgr.Open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "tm_Subject", "Some message subject." );
mgr.PutMessage( "myqueue", msg );

// Java example
QAManager mgr = QAManagerFactory.getInstance().createQAManager(null);
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.createTextMessage();
msg.setStringProperty("tm_Subject", "Some message subject.");
mgr.putMessage("myqueue", mgr);

-- SQL example
begin
    DECLARE @msgid VARCHAR(128);
    SET @msgid = ml_qa_createmessage();
    CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
    CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
    CALL ml_qa_putmessage( @msgid, 'clientid\queuename' );
    COMMIT;
end
```


服务器管理请求参考

目录

服务器管理请求父标记

Condition 标记

使用以下 condition 子标记过滤消息以包括在 MessageDetailsRequest 中。您可以在 <condition> 标记中指定任意多个这些标记。如果使用多个同样的标记，则会对给定的若干个值进行逻辑 OR 运算；如果使用两个不同的标记，则会对这些值进行逻辑 AND 运算。

<condition> 子标记	说明
<address>	选择目标地址为指定地址的消息。
<archived>	返回档案消息存储库中消息的详细信息。
<customRule>	根据规则选择消息。
<kind>	过滤二进制或文本消息。例如，<kind>*text*</kind> 过滤文本消息，而 <kind>*binary*</kind> 过滤二进制消息。
<messageId>	选择具有特定消息 ID 的消息。
<originator>	选择源自指定客户端的消息。
<priority>	选择当前具有指定优先权的消息。
<property>	选择具有指定消息属性的消息。要检查属性名称和值，使用语法 <property>*property-name*=*property-value*</property>。要检查某属性是否存在，使用格式 <property>*property-name*</property>。
<status>	选择当前具有指定状态的消息。

CustomRule 标记

要构造更复杂的条件语句，请使用 <customRule> 标记作为 <condition> 标记（和其它标记）的子标记。此标记使用类似用于服务器传输规则的服务器规则作为其数据。可以采用构造传输规则的条件部分的方式构造这些查询。请参见“条件语法”一节第 763 页。

示例

以下 condition 在选择消息时依据的搜索条件是：优先级设置为 4；发出方名称类似 '%sender%'；状态大于等于 20。

```
<condition>
 <priority>4</priority>
 <customRule>ias_Originator LIKE '%sender%' AND ias_Status &gt;= 20</
```



```
customRule>
</condition>
```

Schedule 标记

可以选择将服务器管理请求设置为根据调度运行。使用以下 <schedule> 子标记定义请求据以运行的调度。

<schedule> 子标记	说明
<starttime>	定义一天中服务器开始生成报告的时间。例如： `<starttime>09:00:00</starttime>`

<between>	包含两个子标记，starttime 和 endtime，它们定义服务器生成报告的时间间隔。不能与 starttime 用在同一调度中。例如：
	
	`<between>`
	`  <starttime>Mon Jan 16 09:00:00 EST 2006</starttime>`
	`  <endtime>Mon Jan 17 09:00:00 EST 2006</endtime>`
	`</between>`

<everyhour>	定义后续报告之间的时间间隔（以小时为单位）。不能与 everyminute 或 everysecond 用在同一调度中。例如，以下请求从 9 AM 开始每两小时生成一个报告：
	
	`<schedule>`
	`  <starttime>09:00:00</starttime>`
	`  <everyhour>2</everyhour>`
	`</schedule>`

<everyminute>	定义后续报告之间的时间间隔（以分钟为单位）。不能与 everyhour 或 everysecond 用在同一调度中。
	
	`<schedule>`
	`  <everyminute>10</everyminute>`
	`</schedule>`

<everysecond>	定义后续报告之间的时间间隔（以秒为单位）。不能与 everyhour 或 everyminute 用在同一调度中。
	
	`<schedule>`
	`  <everysecond>45</everysecond>`
	`</schedule>`

<ondayofweek>	每个标记包含一周中调度处于活动状态的一天。例如，以下调度在星期一和星期二运行：
	
	`<schedule>`
	`  <ondayofweek>Monday</ondayofweek>`
	`  <ondayofweek>Tuesday</ondayofweek>`
	`</schedule>`

<schedule> 子标记	说明

<ondayofmonth>	每个标记包含一月中调度处于活动状态的一天。例如，以下调度在月的 15 号运行：
	
	`<schedule>`
	`  <ondayofmonth>15</ondayofmonth>`
	`</schedule>`

<startdate>	调度转为活动状态的日期。例如： `<startdate>Mon Jan 16 2006</startdate>`

要修改一个调度，请使用同一个 requestId 注册新服务器管理请求。要删除一个调度，使用同一个 requestId 注册服务器管理请求，但要包括调度标记 `<schedule>none</schedule>`。

注意

- 除了 <ondayofweek> 和 <ondayofmonth> 标记外，每个标记在一个调度中只能使用一次。
- <between> 标记和单个 <starttime> 标记不能在同一个调度中同时使用。
- 在同一个调度中仅可以使用 <everysecond>、<everyminute> 和 <everyhour> 中的一个。

示例

以下示例创建一个持久调度，该调度报告服务器上的所有消息，包括每条消息的 ID 和状态。它还覆盖以前指派给请求 ID dailyMessageStatus 的任何持久请求。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
   <request>
     <replyAddr>myclient\messageStatusQueue</replyAddr>
     <requestId>dailyMessageStatus</requestId>
     <schedule>
       <everyhour>24</everyhour>
     </schedule>
     <persistent/>
     <messageId/>
     <status/>
   </request>
  </MessageDetailsRequest>
</actions>
```

以下是一个可能得到的报告样例。它被发送到地址 myclient\messageStatusQueue。它表示服务器上有两条消息，一条状态为 60（已接收），另一条状态为 1（待执行）。

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
   <requestId>dailyMessageStatus</requestId>
   <UTCDatetime>Mon Jan 16 15:03:04 EST 2007</UTCDatetime>
   <statusDescription>Scheduled report</statusDescription>
   <messageCount>2</messageCount>
   <message>
     <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
     <status>60</status>
   </message>
```



```
    <message>
      <messageId>ID:fe857fa8-a7d7-4266-985b-a1818a85d1a2</messageId>
      <status>1</status>
    </message>
</MessageDetailsReport>
```

MessageDetailsReport 标记

每个消息详细信息报告都是一条 XML 消息，其中包含 <MessageDetailsReport> 标记，并由报告标题后接可选的 <message> 标记组成。每个报告的标题由以下标记组成：

<MessageDetailsReport> 子标记	说明
<message>	报告的正文由一列 <message> 标记组成，这些 <message> 标记的子标记显示符合选择条件的每条消息的特定详细信息。如果未选择任何消息或者在原始请求中未指定任何详细信息元素，则报告中不包括任何 <message> 标记。否则，每条消息都有自己的 <message> 标记。
<messageCount>	满足请求的选择条件的消息数。
<requestId>	生成报告的请求的 ID。
<statusDescription>	生成此报告的原因的简短说明。
<UTCDateline>	生成此报告的时间和日期。

消息标记

<message> 子标记	说明
<address>	消息的地址。例如，myclient\myqueue。
<contentSize>	消息内容的大小。如果消息是文本消息，则为字符数。如果是二进制消息，则为字节数。
<expires>	消息过期的日期和时间（如果不发送）。
<kind>	表示该消息是二进制消息 (1) 还是文本消息 (2)。
<messageId>	新消息的消息 ID。请参见“消息标头”一节第 684 页。
<originator>	消息发出方的消息存储库 ID。
<priority>	消息优先级：0 到 9 之间的一个整数，其中 0 表示最低优先级，9 表示最高优先级。

<message> 子标记	说明
<property>	消息的属性。请参见“消息属性”一节第 687 页。
<status>	消息的当前状态。这些状态代码是在“预定义的消息属性”一节第 687 页中定义的。
<statusTime>	消息转为其当前状态的时间。此时间为本地时间。
<transmissionStatus>	消息的同步状态。此值可以是以下各项之一： ● 0 - 消息尚未传输到其预期的接收者消息存储库。 ● 1 - 消息已经传输到其预期的接收者消息存储库。 ● 2 - 接收者消息存储库和源消息存储库相同，无需传输。 ● 3 - 消息已经传输到其预期的接收者，但此传输有待确认。有可能消息传输中断，而 QAnywhere 可能再次传输此消息。

示例

以下是消息详细信息报告的一个示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
 <requestId>testReport</requestId>
 <UTCDatetime>Mon Jan 16 15:03:04 EST 2006</UTCDatetime>
 <statusDescription>Scheduled report</statusDescription>
 <messageCount>1</messageCount>
 <message>
  <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
  <status>60</status>
  <property>
   <name>myPropName</name>
   <value>myPropVal</value>
  </property>
 </message>
</MessageDetailsReport>
```

以下 condition 在选择消息时依据的搜索条件是：(msgId=ID:144...OR msgId=ID225...) AND（status=待执行）AND（kind=文本消息）AND（包含值为 'myVal' 的属性 'myProp'）

```
<condition>
 <messageId>ID:144d7e44dc2d7e1d</messageId>
 <messageId>ID:22578sd5dsd99s8e</messageId>
 <status>1</status>
 <kind>text</kind>
 <property>myProp=myVal</property>
</condition>
```

一次性请求是已忽略 <schedule> 标记的请求。这些请求用于生成单个报告，并在报告发送后删除。此请求生成一个报告，该报告显示当前在服务器上优先级为 9 的所有消息的消息 ID、状态和目标地址。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <MessageDetailsRequest>
  <request>
```



```
    <requestId>testRequest</client>
    <condition>
     <priority>9</priority>
    </condition>
    <messageId/>
    <status/>
    <address/>
   </request>
  </MessageDetailsRequest>
 </actions>
```

以下消息详细信息请求示例生成一个包括消息 ID 和消息状态的报告。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
    <!-- ... -->
    <messageId />
    <status />
  </MessageDetailsRequest>
</actions>
```

服务器管理请求 DTD

以下为服务器管理请求 XML 文档类型的完整定义。此 DTD 是作为本章介绍的服务器管理标记的概览提供的。

```
<!-- Set of requests -->

<!ELEMENT actions (actionsResponseId?,(CloseConnector|OpenConnector|
RestartRules|SetProperty
 |ClientStatusRequest|MessageDetailsRequest|CancelMessageRequest
 |GetClientList)+)>

<!ELEMENT actionsResponseId(requestId)>

<!-- Request for list of all clients -->

<!ELEMENT GetClientList EMPTY>

<!-- Request to close a connector -->

<!ELEMENT CloseConnector (client)>

<!-- Request to open a connector -->

<!ELEMENT OpenConnector (client)>

<!-- Request to restart transmission rules for a client -->

<!ELEMENT RestartRules (client)>

<!-- Request for setting a property -->

<!ELEMENT SetProperty (client,prop)>

<!-- Request for client properties -->

<!ELEMENT GetProperties (client,replyAddr?)>

<!-- Request for the status on a connector -->

<!ELEMENT ClientStatusRequest (request)>

<!-- Request for clients -->

<!ELEMENT MessageDetailsRequest (request)>
<!ELEMENT CancelMessageRequest (request)>

<!ELEMENT request (requestId?,replyAddr?,schedule*,onEvent*,condition?,
archived?
 persistent?,report?,messageId?,status?,priority?,address?,originator?,kind?,
 statusTime?,contentSize?,customRule?,property*)>

<!ELEMENT client (#PCDATA)>

<!ELEMENT prop (name?,value?)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT value (#PCDATA)>

<!ELEMENT replyAddr (#PCDATA)>
```

```
<!ELEMENT requestId (#PCDATA)>

<!ELEMENT persistent EMPTY>

<!ELEMENT report EMPTY>

<!ELEMENT schedule ((starttime|
between)?,everyhour?,everyminute?,everysecond?,
 ondayofweek*,ondayofmonth*)>

<!ELEMENT between (starttime,endtime)>

<!ELEMENT starttime (#PCDATA)>
<!ELEMENT endtime (#PCDATA)>
<!ELEMENT everyhour (#PCDATA)>
<!ELEMENT everyminute (#PCDATA)>
<!ELEMENT everysecond (#PCDATA)>
<!ELEMENT ondayofweek (#PCDATA)>
<!ELEMENT ondayofmonth (#PCDATA)>

<!ELEMENT onEvent (#PCDATA)>

<!ELEMENT condition ((messageId|status|priority|address|originator|kind|
archived|
 customRule|property)+)>

<!ELEMENT archived (#PCDATA)>
<!ELEMENT messageId (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT transmissionStatus (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT originator (#PCDATA)>
<!ELEMENT kind (#PCDATA)>
<!ELEMENT statusTime (#PCDATA)>
<!ELEMENT expires (#PCDATA)>
<!ELEMENT contentSize (#PCDATA)>
<!ELEMENT customRule (#PCDATA)>
<!ELEMENT property (#PCDATA)>

<!-- Reports and response sent back by the server -->

<!-- Report returned as a response to a CancelMessageRequest -->

<!ELEMENT CancelMessageReport (requestId,UTCDatetime,statusDescription,
 messageCount,message*)>

<!-- Report returned as a response to a ClientStatusRequest -->

<!ELEMENT ClientStatusReport (requestId,componentReport)>

<!-- Report returned as a response to a MessageDetailsRequest -->

<!ELEMENT MessageDetailsReport (requestId,UTCDatetime,statusDescription,
 messageCount,message*)>

<!-- Response to a GetPropertiesRequest -->

<!ELEMENT GetPropertiesResponse (client,prop*)>

<!-- Response to a GetClientList -->

<!ELEMENT GetClientListResponse (client*)>
```

```
<!ELEMENT UTCDatetime (#PCDATA)>

<!ELEMENT statusDescription (#PCDATA)>

<!ELEMENT messageCount (#PCDATA)>

<!ELEMENT message ((messageId|status|transmissionStatus|priority|address|
originator|kind|
 statusTime|expires|contentSize|property)*)>

<!-- Report on a specific server component (such as a connector) -->

<!ELEMENT componentReport (client,UTCDatetime,statusCode,statusSubcode?,
    statusDescription?,vendorStatusCode?,vendorStatusDescription?)>

<!ELEMENT statusCode (#PCDATA)>
<!ELEMENT statusSubcode (#PCDATA)>
<!ELEMENT vendorStatusCode (#PCDATA)>
<!ELEMENT vendorStatusDescription (#PCDATA)>
```

QAnywhere 代理实用程序参考

目录

qaagent 实用程序

在单个客户端设备上，使用 QAnywhere 代理 (qaagent) 为所有 QAnywhere 应用程序发送和接收消息。只有在客户端消息存储库是 SQL Anywhere 数据库时，才使用此实用程序。

QAnywhere 不支持用于同步消息存储库的外部实用程序，如 dbmlsync。

语法

qaagent [*option* ...]

选项	说明
@data	读取来自指定的环境变量或配置文件的选项。请参见“@data 选项”一节第 706 页。
-c *connection-string*	为客户端消息存储库指定连接字符串。请参见“-c 选项”一节第 706 页。
-fd *seconds*	指定试图连接主服务器的重试之间的延迟时间。请参见“-fd 选项”一节第 707 页。
-fr *number-of-retries*	指定连接失败后重试连接到主服务器的次数。请参见“-fr 选项”一节第 708 页。
-id *id*	指定 QAnywhere 代理要连接到的客户端消息存储库的 ID。请参见“-id 选项”一节第 709 页。
-idl *download-size*	指定要在消息传输期间使用的下载的最大大小。请参见“-idl 选项”一节第 709 页。
-iu *upload-size*	指定要在消息传输期间使用的上载的最大大小。请参见“-iu 选项”一节第 710 页。
-lp *number*	指定监听器监听来自 MobiLink 服务器的 UDP 通知所使用的端口号。缺省值为 5001。请参见“-lp 选项”一节第 711 页。
-mn *password*	为 MobiLink 用户指定新口令。请参见“-mn 选项”一节第 711 页。
-mp *password*	指定 MobiLink 用户的口令。请参见“-mp 选项”一节第 712 页。
-mu *username*	指定 MobiLink 用户。请参见“-mu 选项”一节第 712 页。
-o *logfile*	指定记录输出消息的文件。请参见“-o 选项”一节第 713 页。

选项	说明
-on *size*	指定 QAnywhere 代理消息日志文件的最大大小，达到该大小后，将用扩展名 .old 重命名该文件并启用一个新文件。请参见“-on 选项”一节第 713 页。
-os *size*	指定 QAnywhere 代理消息日志文件的最大大小，达到该大小后，将创建并使用一个具有新名称的新日志文件。请参见“-os 选项”一节第 714 页。
-ot *logfile*	指定记录输出消息的文件。请参见“-ot 选项”一节第 715 页。
-pc{+|-}	为消息传输启用持久连接。请参见“-pc 选项”一节第 715 页。
-policy *policy-type*	指定 QAnywhere 代理使用的传输策略。请参见“-policy 选项”一节第 716 页。
-push *mode*	启用或禁用推式通知。缺省值为启用。请参见“-push 选项”一节第 717 页。
-q	以安静模式启动 QAnywhere 代理，并且将窗口最小化到系统任务栏。请参见“-q 选项”一节第 718 页。
-qi	以安静模式启动 QAnywhere 代理，并且将窗口完全隐藏。请参见“-qi 选项”一节第 719 页。
-si	初始化数据库以用作客户端消息存储库。请参见“-si 选项”一节第 719 页。
-su	在不运行 dbunload/reload 的情况下，将客户端消息存储库升级到当前版本。请参见“-su 选项”一节第 720 页。
-sur	将客户端消息存储库升级到当前版本，并且执行该消息存储库的 dbunload/reload。请参见“-sur 选项”一节第 721 页
-sv	将 SQL Anywhere 网络数据库服务器用作数据库服务器。请参见“-sv 选项”一节第 722 页。
-v [*levels*]	指定详细程度级别。请参见“-v 选项”一节第 722 页。
-x { **http**|**tcpip**|**tls**|**https** } [(*keyword=value*;...)]	指定与 MobiLink 服务器通信的协议选项。请参见“-x 选项”一节第 723 页。
-xd	指定 QAnywhere 代理应使用 MobiLink 服务器的动态寻址。请参见“-xd 选项”一节第 723 页。

另请参见

- “启动 QAnywhere 代理”一节第 48 页

@data 选项

读取来自指定的环境变量或配置文件的选项。

语法

qaagent @{ *filename* | *environment-variable* } ...

注释

使用此选项，可以将命令行选项放在环境变量或配置文件中。如果两者都存在并使用您指定的名称，则使用环境变量。

请参见“使用配置文件”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果要保护口令或配置文件中的其它信息，可以使用文件隐藏实用程序对配置文件的内容进行模糊处理。请参见“文件隐藏实用程序 (dbfhide)”一节 《SQL Anywhere 服务器 - 数据库管理》。

此选项对于 Windows Mobile 很有用，因为快捷方式中的命令行限于 256 个字符。

Sybase Central 的等效功能

Sybase Central 的 QAnywhere 插件包含一个称作 [**创建代理命令文件**] 的任务。选择该任务后，将提示输入文件名，然后将出现 [**属性**] 窗口以便输入命令信息。所生成的文件包含扩展名 *.qaa*。文件扩展名 *.qaa* 是 Sybase Central 的约定；该文件与要为 @data 选项创建的文件相同。可将通过 Sybase Central 创建的命令文件用作 @data 配置文件。

-c 选项

指定连接到客户端消息存储库的字符串。

语法

qaagent -c *connection-string* ...

缺省值

连接参数	缺省值
uid	ml_qa_user
pwd	qanywhere

注释

连接字符串必须指定连接参数，形式为 *keyword=value*，以分号分隔，参数与参数间不留空格。

通常，不在客户端设备上使用 DSN。qaagent 不使用 ODBC。

以下是可能需要使用的部分连接参数：

- **dbf=*filename*** 使用指定的文件名连接到消息存储库。请参见“DatabaseFile 连接参数 [DBF]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **dbn=*database-name*** 可以通过指定数据库名称而不是数据库文件来连接到已经运行的客户端消息存储库。请参见“DatabaseName 连接参数 [DBN]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **eng=*server-name*** 指定已运行的数据库服务器的名称。缺省值为数据库的名称。请参见“ServerName 连接参数 [ENG]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **uid=*user*** 指定数据库用户 ID 以连接到客户端消息存储库。如果更改缺省的 UID 或 PWD 连接参数，则需要此参数。请参见“Userid 连接参数 [UID]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **pwd=*password*** 为数据库用户 ID 指定口令。如果更改了缺省的 UID 或 PWD 连接参数，则需要此参数。请参见“Password 连接参数 [PWD]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **dbkey=*key*** 指定访问数据库所需的加密密钥。请参见“DatabaseKey 连接参数 [DBKEY]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **start=*startline*** 指定数据库服务器启动行。如果未指定启动行，Windows Mobile 的缺省设置是 `start=dbsrv11 -m -gn 5`，其它 Windows 平台的缺省设置是 `start=dbsrv11 -m`。-m 选项会导致事务日志的内容在检查点处被删除，建议使用该选项。请参见：
  - “StartLine 连接参数 [START]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “-m 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “-gn 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

另请参见

- “连接参数”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere 数据库连接” 《SQL Anywhere 服务器 - 数据库管理》

示例

```
qaagent -id Device1 -c "DBF=qanyclient.db" -x tcpip(host=hostname) -policy
automatic
```

-fd 选项

与 -fr 选项同时指定时，此选项可指定连接到 MobiLink 服务器的尝试之间的延迟时间。

语法

qaagent -fd *seconds ...*

缺省值

- 如果指定 -fr 但不指定 -fd，则延迟时间为 0（在重试连接的尝试之间无延迟）。
- 如果不指定 -fr，则缺省值为无重试连接的尝试。

注释

必须将此选项与 qaagent -fr 选项一起使用。-fr 选项指定重试与主服务器连接的次数，-fd 选项指定重试连接的尝试之间的延迟时间。

通过 -x 选项指定故障转移 MobiLink 服务器时通常使用此选项。缺省情况下，当您设置故障转移 MobiLink 服务器时，QAnywhere 代理在遇到故障时会立即尝试使用备用服务器与主服务器连接。使用 -fr 选项可在转到备用服务器之前触发 QAnywhere 代理再次尝试与主服务器连接，使用 -fd 选项可指定重试与主服务器进行连接之间的间隔时间。

建议将此选项设置为 10 秒或更少。

不能将此选项与 qaagent -xd 选项一起使用。

另请参见

- “-fr 选项”一节第 708 页
- “-x 选项”一节第 723 页
- “设置故障转移机制”一节第 38 页

-fr 选项

指定 QAnywhere 代理与主 MobiLink 服务器重试连接的次数。

语法

qaagent -fr *number-of-retries* ...

缺省值

0 - QAnywhere 代理不尝试重试连接主 MobiLink 服务器。

注释

缺省情况下，如果 QAnywhere 代理无法连接到 MobiLink 服务器，则不会出现错误，也不会发送消息。此选项指定 QAnywhere 代理应重试与 MobiLink 服务器的连接，并指定在尝试使用备用服务器之前应重试连接的次数，或者在未指定备用服务器的情况下发出错误消息。

通过 -x 选项指定故障转移 MobiLink 服务器时通常使用此选项。缺省情况下，当您设置故障转移 MobiLink 服务器时，QAnywhere 代理在遇到故障时会立即尝试使用备用服务器与主服务器连接。此选项将使 QAnywhere 代理在转到备用服务器之前再次尝试主服务器。

此外，可使用 -fd 选项指定重试主服务器之间的时间间隔。

不能将此选项与 qaagent -xd 选项一起使用。

另请参见

- “-fd 选项”一节第 707 页
- “-x 选项”一节第 723 页
- “设置故障转移机制”一节第 38 页

-id 选项

指定 QAnywhere 代理要连接到的客户端消息存储库的 ID。

语法

qaagent -id *id* ...

缺省值

ID 的缺省值是正在运行代理的设备的名称。在某些情况下，设备名可能不唯一，这时必须使用 -id 选项。

注释

每个客户端消息存储库都由唯一的字符序列表示，称为消息存储库 ID。如果在第一次连接到消息存储库时没有提供 ID，则缺省值为设备名。在后续连接中，必须始终使用 -id 选项指定同一个消息存储库 ID。

消息存储库 ID 与 MobiLink 远程 ID 相关。消息存储库 ID 是必需项，因为在所有 MobiLink 应用程序中，每个远程数据库都必须具有唯一的 ID。请参见 “创建和注册 MobiLink 用户”一节 《MobiLink - 客户端管理》。

如果要在某台设备上启动 qaagent 的第二个实例，则必须使用 -id 选项指定唯一的消息存储库 ID。

在 ID 中不能使用以下字符：

- 双引号
- 控制字符
- 双反斜线

其它适用的约束如下：

- ID 所能包含的字符数最大为 120 个。
- 只有当作为转义字符使用时，才能使用单反斜线。
- 如果客户端消息存储库数据库的 quoted_identifier 数据库选项设置为 Off，则 ID 只能包含字母数字字符、下划线、@ 符号、井号和美元符号。

另请参见

- “MobiLink 用户简介”一节 《MobiLink - 客户端管理》
- “设置客户端消息存储库”一节第 23 页

-idl 选项

指定增量下载大小。

语法

qaagent -idl *download-size* [**K** | **M**] ...

缺省值

-1 - 无最大下载大小。

注释

此选项以字节为单位指定消息传输的下载部分的大小。使用后缀 K 或 M 分别指定以千字节或兆字节为单位。

QAnywhere 代理启动时，它将此选项指定的值指派给 ias_MaxDownloadSize 消息存储库属性。此消息存储库属性定义下载大小的上限。触发某个传输时，服务器标记这些传送到客户端的消息，直到所有消息的总大小达到通过此选项设置的上限为止。服务器继续发送批量消息，直到将所有排队的消息传送完。传输完每批消息后，都要重新执行传输规则，以便在传输期间出现高优先级的消息排队时，它跳到排头。

超过下载阈值的、排队准备发送到客户端的消息被拆分为多个较小的消息部分。每个消息部分都可以单独下载，这样便可以通过数次同步逐步地下载消息。所有的消息部分都到达目标后，完整的消息即到达。

增量下载大小是一个近似值。实际下载大小还取决于消息大小以外的许多其它因素。

另请参见

- “预定义的客户端消息存储库属性”一节第 746 页中的 ias_MaxDownloadSize

-iu 选项

指定增量上载大小。

语法

qaagent -iu *upload-size* [**K** | **M**] ...

缺省值

256K

注释

此选项以字节为单位指定消息传输的上载部分的大小。使用后缀 K 或 M 分别指定以千字节或兆字节为单位。

QAnywhere 代理启动时，它将此选项指定的值指派给 ias_MaxUploadSize 消息存储库属性。该消息存储库属性定义上载大小的上限。触发某个传输时，代理标记这些传送到服务器的消息，直到所有消息的总大小达到通过此选项设置的上限为止。达到该上限后，会将这些消息发送到服务器。只要这些消息到达服务器并且服务器向客户端成功发送确认，即使该传输的下载阶段失败，仍认为传送消息成功。代理继续向服务器批量发送消息，直到将所有排队的消息传送完。传输完每批消息后，都要重新执行传输规则，以便在传输期间出现高优先级的消息排队时，它跳到排头。

超过上载阈值的消息被拆分为多个较小的消息部分。每个消息部分都可以单独上载，这样便可以通过数次同步逐步地上载消息。所有的消息部分都到达目标后，完整的消息即到达。

增量上载大小是一个近似值。实际上载大小还取决于消息大小以外的许多其它因素。

另请参见

- “预定义的客户端消息存储库属性”一节第 746 页中的 ias_MaxUploadSize

-lp 选项

指定监听器端口。

语法

qaagent -lp *number ...*

缺省值

5001

注释

监听器监听来自 MobiLink 服务器的 UDP 通知所使用的端口号。通知用于通知 QAnywhere 代理有消息正在等待。QAnywhere 代理还使用 UDP 端口向监听器发送控制命令。

另请参见

- “使用推式通知进行消息传递的方案”一节第 6 页
- “-push 选项”一节第 717 页

-mn 选项

为 MobiLink 用户指定新口令。

语法

qaagent -mp *password ...*

缺省值

无

注释

用于更改口令。

另请参见

- “MobiLink 用户” 《MobiLink - 客户端管理》
- “-mp 选项”一节第 712 页
- “-mu 选项”一节第 712 页

-mp 选项

为 MobiLink 用户指定 MobiLink 口令。

语法

qaagent -mp *password* ...

缺省值

无

注释

如果 MobiLink 服务器需要用户验证，则使用 -mp 提供 MobiLink 口令。

另请参见

- “MobiLink 用户” 《MobiLink - 客户端管理》
- “-mu 选项”一节第 712 页

-mu 选项

指定 MobiLink 用户名。

语法

qaagent -mu *username* ...

缺省值

客户端消息存储库 ID

注释

MobiLink 服务器进行验证时要使用 MobiLink 用户名。

如果指定的用户名不存在，则为您创建该用户名。

所有 MobiLink 用户名都必须在服务器消息存储库中注册。请参见“注册 QAnywhere 客户端用户名”一节第 31 页。

另请参见

- “MobiLink 用户” 《MobiLink - 客户端管理》
- “-id 选项”一节第 709 页
- “-mp 选项”一节第 712 页
- “远程 ID”一节 《MobiLink - 客户端管理》

-o 选项

将输出发送到指定的日志文件中。

语法

qaagent -o *logfile* ...

缺省值

无

注释

QAnywhere 代理将输出记录到指定的文件名。如果文件已经存在，则新日志信息会附加到文件。SQL Anywhere 同步客户端 (dbmlsync) 将输出记录到具有相同名称但包含后缀 *_sync* 的文件中。监听器实用程序 (dblsn) 将输出记录到具有相同名称但包含后缀 *_lsn* 的文件中。

例如，如果将日志文件指定为 *c:\tmp\mylog.out*，则 qaagent 会将输出记录到 *c:\tmp\mylog.out*，dbmlsync 会记录到 *c:\tmp\mylog_sync.out*，而 dblsn 会记录到 *c:\tmp\mylog_lsn.out*。

另请参见

- “-ot 选项”一节第 715 页
- “-on 选项”一节第 713 页
- “-os 选项”一节第 714 页
- “-v 选项”一节第 722 页

-on 选项

指定 QAnywhere 代理消息日志文件的最大大小，达到该大小后，将用扩展名 *.old* 重命名该文件并启用一个新文件。

语法

qaagent -on *size* [**k** | **m**] ...

缺省值

无

注释

size 是消息日志文件大小的最大值，以字节为单位。使用后缀 k 或 m 分别指定以千字节或兆字节为单位。最小值限制为 10K。

日志文件达到指定大小后，QAnywhere 代理将用扩展名 *.old* 重命名输出文件，并以原始名称起用一个新文件。

注意
如果 *.old* 文件已经存在，则将覆盖它。为避免丢失旧的日志文件，请使用 -os 选项。

此选项不能与 -os 选项一起使用。

另请参见

- “-o 选项”一节第 713 页
- “-ot 选项”一节第 715 页
- “-os 选项”一节第 714 页
- “-v 选项”一节第 722 页

-os 选项

指定 QAnywhere 代理消息日志文件的最大大小，达到该大小后，将创建并使用一个具有新名称的新日志文件。

语法

qaagent -os *size* [**k** | **m**] ...

缺省值

无

注释

size 是记录输出消息的文件大小的最大值。缺省单位是字节。使用后缀 k 或 m 分别指定以千字节或兆字节为单位。最小值限制为 10K。

在 QAnywhere 代理将输出消息记录到文件之前，它会检查当前文件的大小。如果日志消息使得文件大小超过指定的大小，QAnywhere 代理会将消息日志文件重命名为 *yymmddxx.mls*。其中，*xx* 是从 00 到 99 的连续字符，而 *yymmdd* 表示当前的年、月、日。

可以使用该选项删除旧的消息日志文件，从而释放磁盘空间。最新输出总是附加到由 -o 或 -ot 指定的文件中。

注意
此选项不能与 -on 选项一起使用。

另请参见

- “-o 选项”一节第 713 页
- “-ot 选项”一节第 715 页
- “-on 选项”一节第 713 页
- “-v 选项”一节第 722 页

-ot 选项

截断日志文件并将输出消息附加到该文件。

语法

qaagent -ot *logfile* ...

缺省值

无

注释

QAnywhere 代理将输出记录到指定的文件名。如果该文件存在，则首先将其大小截断为 0。SQL Anywhere 同步客户端 (dbmlsync) 将输出记录到具有相同名称但文件名包括后缀 *_sync* 的文件。监听器实用程序 (dblsn) 将输出记录到具有相同名称但包含后缀 *_lsn* 的文件中。

例如，如果将日志文件指定为 *c:\tmp\mylog.out*，则 qaagent 会将输出记录到 *c:\tmp\mylog.out*，dbmlsync 会记录到 *c:\tmp\mylog_sync.out*，而 dblsn 会记录到 *c:\tmp\mylog_lsn.out*。

另请参见

- “-o 选项”一节第 713 页
- “-on 选项”一节第 713 页
- “-os 选项”一节第 714 页
- “-v 选项”一节第 722 页

-pc 选项

在两个同步之间保持与 MobiLink 服务器的持久连接。

语法

qaagent -pc { **+** | **-** } ...

缺省值

-pc-

注释

当网络覆盖率高而且通过 QAnywhere 的消息流量大时，启用持久连接 (-pc+) 很有用。在这种情况下，可以减少每次发生消息传输时设置和取消一个 TCP/IP 连接的网络开销。

当客户端设备具有一个公共 IP 地址并且可由 UDP 或 SMS 访问时，在以下情况下禁用持久连接 (-pc-) 很有用：

- 客户端设备正在使用拨号网络并且连接时间的费用需要考虑。
- 通过 QAnywhere 的消息流量很小。持久的 TCP/IP 连接会消耗网络服务器的资源，因此会影响到可伸缩性。
- 客户端设备的网络覆盖率不可靠。在可以连接时，可使用自动策略传输消息。在这种环境中尝试维持持久连接并不实用，并且会浪费 CPU 资源。

另请参见

- “-push 选项”一节第 717 页
- “-pc 选项”一节 《MobiLink - 客户端管理》

-policy 选项

指定确定何时发生消息传输的策略。

语法

qaagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

缺省值

- 缺省的策略类型是 [自动]。
- [调度] 策略的缺省时间间隔是 900 秒（15 分钟）。

注释

QAnywhere 使用一种策略确定何时发生消息传输。*policy-type* 可以是下列值之一：

- **要求时** 仅当 QAnywhere 客户端应用程序进行适当的方法调用时，才传输消息。

  QAManager PutMessage() 方法使消息在本地排队。直到调用 QAManager TriggerSendReceive() 方法后，才将这些消息传输到服务器。同样，直到客户端调用 TriggerSendReceive() 方法后才将在服务器上等待的消息发送给客户端。

  使用 [要求时] 策略时，应用程序负责在从服务器收到推式通知后触发消息传输。推式通知会导致将系统消息发送到 QAnywhere 客户端。在应用程序中，可以选择通过调用 TriggerSendReceive() 响应此系统消息。

  有关示例内容，请参见“系统队列”一节第 64 页。

- **调度** 指定调度后，只要满足以下任何一个条件，代理就将每隔 *n* 秒钟执行一次消息传输：

  ○ 前一个时间间隔结束后在客户端消息存储库中放入了新的消息。

- 前一个时间间隔结束后某条消息的状态发生了更改。当某条消息得到应用程序的确认时通常会发生这种情况。有关确认的详细信息，请参见：

  - .NET： “AcknowledgementMode 枚举”一节第 212 页
  - C++： “AcknowledgementMode 类”一节第 386 页
  - Java： “AcknowledgementMode 接口”一节第 498 页

- 前一个时间间隔结束后接收到推式通知。
- 前一个时间间隔结束后接收到网络状态更改通知。
- 推式通知被禁用。

可调用 TriggerSendReceive 方法来替换时间间隔。该方法会强制在时间间隔结束前传输消息。请参见：

- .NET： “TriggerSendReceive 方法”一节第 297 页
- C++： “triggerSendReceive 函数”一节第 453 页
- Java： “triggerSendReceive 方法”一节第 570 页
- SQL： “ml_qa_triggersendreceive”一节第 682 页

- **自动** 发生下面描述的事件之一时，将传输消息。

  QAnywhere 代理尝试保持消息队列为最新。下面的任一事件会导致将客户端的消息队列传送到服务器并将服务器的消息队列传送到客户端：

  - 调用 PutMessage()。
  - 调用 TriggerSendReceive()。
  - 出现推式通知。

    有关通知的信息，请参见“使用推式通知进行消息传递的方案”一节第 6 页。

  - 客户端上发生消息状态更改。例如，当应用程序从本地队列中检索消息时消息状态会从待执行更改为已接收。

- ***rules-file*** 指定客户端传输规则文件。传输规则文件可以指示一组更复杂的规则以确定何时传输消息。

  请参见“客户端传输规则”一节第 769 页。

另请参见

- “确定在客户端进行消息传输的时间”一节第 51 页
- “使用推式通知进行消息传递的方案”一节第 6 页

-push 选项

指定是否启用推式通知。

语法

qaagent -push *mode* ...

mode : **none** | **connected** | **disconnected**

缺省值

connected

选项

模式	说明
none	为该代理禁用推式通知。不启动监听器 (dblsn)。
connected	使用持久连接通过 TCP/IP 为该代理启用推式通知。通过 qaagent 启动监听器 (dblsn)，并且尝试与 MobiLink 服务器维护持久连接。当客户端设备没有公共 IP 地址或者当 MobiLink 服务器受到不允许 UDP 消息通过的防火墙保护时，此模式很有用。这是缺省设置。
disconnected	不使用持久连接通过 UDP 为该代理启用推式通知。通过 qaagent 启动监听器 (dblsn)，但不与 MobiLink 服务器维护持久连接。而是由 UDP 监听器从 MobiLink 接收推式通知。当客户端设备具有一个公共 IP 地址并且可由 UDP 或 SMS 访问时，此模式在以下情况下很有用： ● 客户端设备正在使用拨号网络并且连接时间的费用需要考虑。 ● 通过 QAnywhere 的消息流量很小。持久的 TCP/IP 连接会消耗网络服务器的资源，因此会影响到可伸缩性。 ● 客户端设备的网络覆盖率不可靠。在可以连接时，可使用自动策略传输消息。在这种环境中尝试维持持久连接并不实用，并且会浪费 CPU 资源。

注释

如果不想使用通知，将此选项设置为 [无]。这样将不需要在客户端部署 *dblsn.exe* 可执行程序。

有关不使用通知的 QAnywhere 说明，请参见“简单消息传递方案”一节第 4 页。

如果使用 UDP，由于 ActiveSync 的 UDP 实现的限制，不能在断开连接模式下通过 ActiveSync 使用推式通知。

另请参见

- “使用推式通知”一节第 34 页
- “-pc 选项”一节第 715 页
- “启动 QAnywhere 代理”一节第 48 页
- “有关推式通知的通知”一节第 65 页
- “-lp 选项”一节第 711 页

-q 选项

以安静模式启动 QAnywhere 代理，并且将窗口最小化到系统任务栏。

语法

qaagent -q ...

缺省值

无

注释

当使用 -q 以安静模式启动 QAnywhere 代理时，主窗口将最小化到系统任务栏中。此外，用于消息存储库的数据库服务器可通过 -qi 选项来启动。

另请参见

- “-qi 选项”一节第 719 页

-qi 选项

以安静模式启动 QAnywhere 代理，并且将窗口完全隐藏。

语法

qaagent -qi ...

缺省值

无

注释

以安静模式启动 QAnywhere 代理时，在 Windows 桌面操作系统中，主窗口将最小化到系统任务栏中，而在 Windows Mobile 中，主窗口将隐藏。此外，用于消息存储库的数据库服务器可通过 -qi 选项来启动。

对于某些 Windows Mobile 应用程序，安静模式很有用，因为它能防止某个应用程序在 Windows Mobile 达到 32 个并发进程的限制时关闭。安静模式允许 QAnywhere 代理像服务一样运行。

在 -qi 安静模式下时，只能通过键入 **qastop** 来停止 QAnywhere 代理。

另请参见

- “qastop 实用程序”一节第 743 页
- “-q 选项”一节第 718 页

-si 选项

初始化数据库以用作客户端消息存储库。

语法

qaagent -c "*connection-string*" -si ...

缺省值

无。只使用一次此选项以初始化客户端消息存储库。

注释

使用此选项之前，必须创建 SQL Anywhere 数据库。使用 -si 时，QAnywhere 代理使用数据库对象初始化数据库（例如 QAnywhere 系统表），然后立即退出。

运行 -si 时，必须使用 -c 选项指定连接字符串，来指示要初始化的数据库。在 -c 选项中指定的连接字符串还应指定一个具有 DBA 特权的用户 ID。如果没有指定用户 ID 和口令，则使用缺省的用户 DBA 和口令 SQL。

-si 选项为客户端消息存储库创建一个名为 ml_qa_user 的数据库用户，口令为 qanywhere。名为 ml_qa_user 的用户具有仅适用于 QAnywhere 应用程序的权限。如果不更改此数据库用户名和口令，启动 qaagent 时就不需要在 -c 选项中指定 pwd 或 uid。如果更改了其中一个，必须在 qaagent 命令行的 -c 选项中提供 uid 和/或 pwd。

注意
您应该更改缺省口令。可使用 GRANT 语句进行更改。请参见“更改口令”一节 《SQL Anywhere 服务器 - 数据库管理》。

-si 选项不向客户端消息存储库提供 ID。运行 -si 或下次运行 qaagent 时可以使用 -id 选项指派 ID；如果不这样做，缺省情况下 qaagent 会将设备名指派为 ID。

如果创建了消息存储库但没有使用 ID 进行设置，相对于消息存储库的本地 QAnywhere 应用程序可以发送和接收消息，但不能与远程 QAnywhere 应用程序交换消息。指派 ID 后，远程消息传递将也可以发生。

另请参见

- “设置客户端消息存储库”一节第 23 页
- “创建安全的客户端消息存储库”一节第 132 页

示例

下面的命令连接到名为 *qaclient.db* 的数据库并将其初始化为 QAnywhere 客户端消息存储库。初始化完成后，QAnywhere 代理立即退出。

```
qaagent -si -c "DBF=qaclient.db"
```

-su 选项

将客户端消息存储库升级到当前版本。

语法

qaagent -su -c "*connection-string*" ...

注释

如果在卸载/重装消息存储库之后和 qaagent 升级之前，要执行自定义操作，此选项很有用。如果从版本 10.0.0 以前的消息存储库升级，并且要该代理自动为您执行卸载/重装步骤，则使用 -sur 选项。

如果从版本 10.0.0 以前的消息存储库升级，则必须首先手工卸载然后重装消息存储库。

升级完成后，此操作会退出。

此操作无法撤消。

另请参见

- “-sur 选项”一节第 721 页

示例

要从版本 9 的数据库升级，首先要卸载并重装数据库：

```
dbunload  -q  -c "UID=dba;PWD=sql;DBF=qanywhere.db" -ar
```

然后，使用 -su 选项运行 qaagent：

```
qaagent -q -su -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-sur 选项

将客户端消息存储库升级到当前版本。

语法

qaagent -sur -c "*connection-string*" ...

注释

在连接字符串中指定要升级的数据库。-sur 选项自动卸载消息存储库并重装，然后将其升级。

从版本 9 的消息存储库升级到版本 10 的消息存储库，必须执行卸载/重装。-su 选项可与手工卸载/重装一起执行。例如，如果需要在重装后升级前执行自定义操作，则使用 -su 选项。

升级完成后，此操作会退出。

此操作无法撤消。

另请参见

- “-su 选项”一节第 720 页

示例

以下示例卸载并重装版本 9.0.2 的 SQL Anywhere 数据库（名为 qanywhere.db），使其可用于 QAnywhere 版本 11。

```
qaagent -q -sur -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```


-sv 选项

通知代理将 SQL Anywhere 网络数据库服务器用作数据库服务器，而不是使用个人数据库服务器。

语法

qaagent -sv -c "*connection-string*" ...

注释

缺省情况下，qaagent 会使用 dbeng11 应用程序连接到个人数据库服务器。而指定了 -sv 时，qaagent 将使用 dbsrv11 应用程序连接到网络数据库服务器。

另请参见

- “SQL Anywhere 数据库服务器”一节 《SQL Anywhere 服务器 - 数据库管理》

-v 选项

允许您指定 QAnywhere 代理消息日志文件中记录的信息和 QAnywhere 代理消息窗口中显示的信息。

语法

qaagent -v *levels* ...

缺省值

最低详细程度

注释

-v 选项影响消息日志文件和消息窗口。只有在 qaagent 命令行中指定 -o 或 -ot 时才能有一个消息日志文件。

高详细程度可能会影响性能，通常应该仅在开发阶段使用高详细程度。

如果仅指定 -v，将记录最少的信息。

levels 的值如下所示。一次可以使用一个或多个选项，例如 -vlm。

- **+** 打开所有记录选项。
- **l** 显示所有 MobiLink 监听器记录。这使 MobiLink 监听器 (dblsn) 以详细程度级别 -v3 启动。请参见“用于 Windows 设备的监听器实用程序”一节 《MobiLink - 服务器启动的同步》中的 -v 选项。
- **m** 显示所有 dbmlsync 记录。这使 SQL Anywhere 同步客户端 (dbmlsync) 以详细程度级别 -v+ 启动。请参见 dbmlsync “-v 选项”一节 《MobiLink - 客户端管理》。
- **n** 显示所有网络状态更改通知。QAnywhere 代理从监听器实用程序接收这些通知。
- **p** 显示所有消息推式通知。QAnywhere 代理通过 MobiLink 服务器（包括 MobiLink Notifier）从监听器实用程序接收这些通知。

- **q** 显示用于表示传输规则的 SQL。
- **s** 显示由 QAnywhere 代理初始化的所有消息同步。

另请参见

- “-o 选项”一节第 713 页
- “-ot 选项”一节第 715 页
- “-on 选项”一节第 713 页
- “-os 选项”一节第 714 页

-x 选项

指定网络协议和协议选项以便与 MobiLink 服务器通信。

语法

qaagent -x *protocol* [**(** *protocol-options*;... **)** ...]

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

注释

有关 *protocol-options* 的完整列表，请参见“MobiLink 客户端网络协议选项” 《MobiLink - 客户端管理》。

当 MobiLink 服务器与 QAnywhere 代理不在同一设备上时，需要使用 -x 选项。

可以多次指定 -x。这使您可以为多个 MobiLink 服务器设置故障转移。设置故障转移时，QAnywhere 代理按照在命令行中输入的顺序尝试连接 MobiLink 服务器。

QAnywhere 代理还具有从 MobiLink 服务器（其上有可用于向客户端传输的消息）接收通知的监听器。此监听器仅使用所指定的第一个 MobiLink 服务器，并且不故障转移到其它服务器。

另请参见

- “MobiLink 客户端网络协议选项” 《MobiLink - 客户端管理》
- “加密通信流”一节第 134 页
- “传送层安全” 《SQL Anywhere 服务器 - 数据库管理》
- “设置故障转移机制”一节第 38 页
- “-fd 选项”一节第 707 页
- “-fr 选项”一节第 708 页

-xd 选项

指定 QAnywhere 代理应使用 MobiLink 服务器的动态寻址。

语法

qaagent -xd

注释

当指定 -xd 时，QAnywhere 代理根据消息存储库属性确定 MobiLink 服务器的协议和地址。这意味着它可以动态确定单个 MobiLink 服务器的地址，其中服务器地址取决于正在运行 QAnywhere 代理的设备目前使用的网络。

QAnywhere 应用程序必须初始化描述 MobiLink 服务器通信协议和地址的消息存储库属性，并建立与当前活动网络接口的关系。由于移动设备会在不同网络间切换，QAnywhere 代理会检测处于活动状态的网络，并自动调整 MobiLink 服务器的通信协议和地址—而不必重新启动。

另请参见

- “客户端消息存储库属性”一节第 26 页

示例

以下示例会设置属性，以便根据设备所处的网络类型使用相应的 MobiLink 地址。例如，如果设备位于 LAN 上，则使用相应的 LAN 地址。

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress",
"host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress",
"host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Sierra Wireless AirCard 555 Adapter.type",
"WAN" );
```


qauagent 实用程序

在单个客户端设备上，使用 QAnywhere UltraLite 代理 (qauagent) 为所有 QAnywhere 应用程序发送和接收消息。只有在客户端消息存储库是 UltraLite 数据库时，才使用此实用程序。

注意
dbmlsync 实用程序不支持 UltraLite 消息存储库，该实用程序被设计用于同步 SQL Anywhere 消息存储库。

语法

qauagent [*option* ...]

选项	说明
@data	读取来自指定的环境变量或配置文件的选项。请参见“@data 选项”一节第 726 页。
-c *connection-string*	为客户端消息存储库指定连接字符串。请参见“-c 选项”一节第 727 页。
-fd *seconds*	指定试图连接主服务器的重试之间的延迟时间。请参见“-fd 选项”一节第 728 页。
-fr *number-of-retries*	指定连接失败后重试连接到主服务器的次数。请参见“-fr 选项”一节第 728 页。
-id *id*	指定 QAnywhere UltraLite 代理要连接到的客户端消息存储库的 ID。请参见“-id 选项”一节第 729 页。
-idl *download-size*	指定要在消息传输期间使用的下载的最大大小。请参见“-idl 选项”一节第 730 页。
-iu *upload-size*	指定要在消息传输期间使用的上载的最大大小。请参见“-iu 选项”一节第 731 页。
-lp *number*	指定监听器监听来自 MobiLink 服务器的 UDP 通知所使用的端口号。缺省值为 5001。请参见“-lp 选项”一节第 731 页。
-mn *password*	为 MobiLink 用户指定新口令。请参见“-mn 选项”一节第 732 页。
-mp *password*	指定 MobiLink 用户的口令。请参见“-mp 选项”一节第 732 页。
-mu *username*	指定 MobiLink 用户。请参见“-mp 选项”一节第 732 页。
-o *logfile*	指定记录输出消息的文件。请参见“-o 选项”一节第 733 页。

选项	说明
-on *size*	指定 QAnywhere UltraLite 代理消息日志文件的最大大小，达到该大小后，将用扩展名 .old 重命名该文件并启用一个新文件。请参见“-on 选项”一节第 734 页。
-os *size*	指定 QAnywhere UltraLite 代理消息日志文件的最大大小，达到该大小后，将创建并使用一个具有新名称的新日志文件。请参见“-os 选项”一节第 734 页。
-ot *logfile*	指定记录输出消息的文件。请参见“-ot 选项”一节第 735 页。
-policy *policy-type*	指定 QAnywhere UltraLite 代理使用的传输策略。请参见“-policy 选项”一节第 736 页。
-push *mode*	启用或禁用推式通知。缺省值为启用。请参见“-push 选项”一节第 737 页。
-q	以安静模式启动 QAnywhere UltraLite 代理，并且将窗口最小化到系统任务栏。请参见“-q 选项”一节第 738 页。
-qi	以安静模式启动 QAnywhere UltraLite 代理，并且窗口完全隐藏。请参见“-qi 选项”一节第 738 页。
-si	初始化数据库以用作客户端消息存储库。请参见“-si 选项”一节第 739 页。
-v [*levels*]	指定详细程度级别。请参见“-v 选项”一节第 740 页。
-x { **http**|**tcpip**|**tls**|**https** } [(*keyword*=*value*;...)]	指定与 MobiLink 服务器通信的协议选项。请参见“-x 选项”一节第 741 页。
-xd	指定 QAnywhere 代理应使用 MobiLink 服务器的动态寻址。请参见“-xd 选项”一节第 741 页。

另请参见

- “启动 QAnywhere 代理”一节第 48 页

@data 选项

读取来自指定的环境变量或配置文件的选项。

语法

qauagent @{ *filename* | *environment-variable* } ...

注释

使用此选项，可以将命令行选项放在环境变量或配置文件中。如果两者都存在并使用您指定的名称，则使用环境变量。

请参见“使用配置文件”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果要保护口令或配置文件中的其它信息，可以使用文件隐藏实用程序对配置文件的内容进行模糊处理。请参见“文件隐藏实用程序 (dbfhide)”一节 《SQL Anywhere 服务器 - 数据库管理》。

此选项对于 Windows Mobile 很有用，因为快捷方式中的命令行限于 256 个字符。

Sybase Central 的等效功能

Sybase Central 的 QAnywhere 插件包含一个称作 [**创建代理命令文件**] 的任务。选择该任务后，将提示输入文件名，然后将出现 [**属性**] 窗口以便输入命令信息。所生成的文件包含扩展名 *.qaa*。文件扩展名 *.qaa* 是 Sybase Central 的约定；该文件与要为 @data 选项创建的文件相同。可将通过 Sybase Central 创建的命令文件用作 @data 配置文件。

-c 选项

指定连接到客户端消息存储库的字符串。

语法

qauagent -c *connection-string ...*

缺省值

连接参数	缺省值
uid	ml_qa_user
pwd	qanywhere

注释

连接字符串必须指定连接参数，形式为 *keyword=value*，以分号分隔，参数与参数间不留空格。

通常，不在客户端设备上使用 DSN。qauagent 不使用 ODBC。

以下是可能需要使用的部分连接参数：

- **dbf=*filename*** 使用指定的文件名连接到消息存储库。请参见“UltraLite DBF 连接参数”一节 《UltraLite - 数据库管理和参考》。
- **dbn=*database-name*** 可以通过指定数据库名称而不是数据库文件来连接到已经运行的客户端消息存储库。请参见“UltraLite DBN 连接参数”一节 《UltraLite - 数据库管理和参考》。
- **uid=*user*** 指定数据库用户 ID 以连接到客户端消息存储库。如果更改缺省的 UID 或 PWD 连接参数，则需要此参数。请参见“UltraLite UID 连接参数”一节 《UltraLite - 数据库管理和参考》。

- **pwd=*password*** 为数据库用户 ID 指定口令。如果更改了缺省的 UID 或 PWD 连接参数，则需要此参数。请参见“UltraLite PWD 连接参数”一节 《UltraLite - 数据库管理和参考》。
- **dbkey=*key*** 指定访问数据库所需的加密密钥。请参见“UltraLite DBKEY 连接参数”一节 《UltraLite - 数据库管理和参考》。

另请参见

- “连接参数”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere 数据库连接” 《SQL Anywhere 服务器 - 数据库管理》

示例

```
qauagent -id Device1 -c "DBF=qanyclient.db" -x tcpip(host=hostname) -policy
automatic
```

-fd 选项

与 -fr 选项同时指定时，此选项可指定连接到 MobiLink 服务器的尝试之间的延迟时间。

语法

qauagent -fd *seconds* ...

缺省值

- 如果指定 -fr 但不指定 -fd，则延迟时间为 0（在重试连接的尝试之间无延迟）。
- 如果不指定 -fr，则缺省值为无重试连接的尝试。

注释

必须将此选项与 qauagent -fr 选项一起使用。-fr 选项指定重试与主服务器连接的次数，-fd 选项指定重试连接的尝试之间的延迟时间。

通过 -x 选项指定故障转移 MobiLink 服务器时通常使用此选项。缺省情况下，当您设置故障转移 MobiLink 服务器时，QAnywhere UltraLite 代理在遇到故障时会立即尝试使用备用服务器与主服务器连接。使用 -fr 选项可在转到备用服务器之前触发 QAnywhere UltraLite 代理再次尝试与主服务器连接，使用 -fd 选项可指定重试与主服务器进行连接之间的间隔时间。

建议将此选项设置为 10 秒或更少。

不能将此选项与 qauagent -xd 选项一起使用。

另请参见

- “-fr 选项”一节第 728 页
- “-x 选项”一节第 741 页
- “设置故障转移机制”一节第 38 页

-fr 选项

指定 QAnywhere UltraLite 代理与主 MobiLink 服务器重试连接的次数。

语法

qauagent -fr *number-of-retries* ...

缺省值

0 - QAnywhere UltraLite 代理不尝试重试连接主 MobiLink 服务器。

注释

缺省情况下，如果 QAnywhere UltraLite 代理无法连接到 MobiLink 服务器，则不会出现错误，也不会发送消息。此选项指定 QAnywhere UltraLite 代理应重试与 MobiLink 服务器的连接，并指定在尝试使用备用服务器之前应重试连接的次数，或者在未指定备用服务器的情况下发出错误消息。

通过 -x 选项指定故障转移 MobiLink 服务器时通常使用此选项。缺省情况下，当您设置故障转移 MobiLink 服务器时，QAnywhere UltraLite 代理在遇到故障时会立即尝试使用备用服务器与主服务器连接。此选项将使 QAnywhere UltraLite 代理在转到备用服务器之前再次尝试主服务器。

此外，可使用 -fd 选项指定重试主服务器之间的时间间隔。

不能将此选项与 qauagent -xd 选项一起使用。

另请参见

- “-fd 选项”一节第 728 页
- “-x 选项”一节第 741 页
- “设置故障转移机制”一节第 38 页

-id 选项

指定 QAnywhere UltraLite 代理要连接到的客户端消息存储库的 ID。

语法

qauagent -id *id* ...

缺省值

ID 的缺省值是正在运行代理的设备的名称。在某些情况下，设备名可能不唯一，这时必须使用 -id 选项。

注释

每个客户端消息存储库都由唯一的字符序列表示，称为消息存储库 ID。如果在第一次连接到消息存储库时没有提供 ID，则缺省值为设备名。在后续连接中，必须始终使用 -id 选项指定同一个消息存储库 ID。

消息存储库 ID 与 MobiLink 远程 ID 相关。消息存储库 ID 是必需项，因为在所有 MobiLink 应用程序中，每个远程数据库都必须具有唯一的 ID。请参见“创建和注册 MobiLink 用户”一节 《MobiLink - 客户端管理》 。

如果要在某台设备上启动 qauagent 的第二个实例，则必须使用 -id 选项指定唯一的消息存储库 ID。

在 ID 中不能使用以下字符：

- 双引号
- 控制字符
- 双反斜线

其它适用的约束如下：

- ID 所能包含的字符数最大为 120 个。
- 只有当作为转义字符使用时，才能使用单反斜线。
- 如果客户端消息存储库数据库的 quoted_identifier 数据库选项设置为 Off，则 ID 只能包含字母数字字符、下划线、@ 符号、井号和美元符号。

另请参见

- “MobiLink 用户简介”一节 《MobiLink - 客户端管理》
- “设置客户端消息存储库”一节第 23 页

-idl 选项

指定增量下载大小。

语法

qauagent -idl *download-size* [**K** | **M**] ...

缺省值

-1 - 无最大下载大小。

注释

此选项以字节为单位指定消息传输的下载部分的大小。使用后缀 K 或 M 分别指定以千字节或兆字节为单位。

QAnywhere UltraLite 代理启动时，它将此选项指定的值指派给 ias_MaxDownloadSize 消息存储库属性。此消息存储库属性定义下载大小的上限。触发某个传输时，服务器标记这些传送到客户端的消息，直到所有消息的总大小达到通过此选项设置的上限为止。服务器继续发送批量消息，直到将所有排队的消息传送完。传输完每批消息后，都要重新执行传输规则，以便在传输期间出现高优先级的消息排队时，它跳到排头。

超过下载阈值的、排队准备发送到客户端的消息被拆分为多个较小的消息部分。每个消息部分都可以单独下载，这样便可以通过数次同步逐步地下载消息。所有的消息部分都到达目标后，完整的消息即到达。

增量下载大小是一个近似值。实际下载大小还取决于消息大小以外的许多其它因素。

另请参见

- “预定义的客户端消息存储库属性”一节第 746 页中的 ias_MaxDownloadSize

-iu 选项

指定增量上载大小。

语法

qauagent -iu *upload-size* [**K** | **M**] ...

缺省值

256K

注释

此选项以字节为单位指定消息传输的上载部分的大小。使用后缀 K 或 M 分别指定以千字节或兆字节为单位。

QAnywhere UltraLite 代理启动时，它将此选项指定的值指派给 ias_MaxUploadSize 消息存储库属性。该消息存储库属性定义上载大小的上限。触发某个传输时，代理标记这些传送到服务器的消息，直到所有消息的总大小达到通过此选项设置的上限为止。达到该上限后，会将这些消息发送到服务器。只要这些消息到达服务器并且服务器向客户端成功发送确认，即使该传输的下载阶段失败，仍认为传送消息成功。代理继续向服务器批量发送消息，直到将所有排队的消息传送完。传输完每批消息后，都要重新执行传输规则，以便在传输期间出现高优先级的消息排队时，它跳到排头。

超过上载阈值的消息被拆分为多个较小的消息部分。每个消息部分都可以单独上载，这样便可以通过数次同步逐步地上载消息。所有的消息部分都到达目标后，完整的消息即到达。

增量上载大小是一个近似值。实际上载大小还取决于消息大小以外的许多其它因素。

另请参见

- “预定义的客户端消息存储库属性”一节第 746 页中的 ias_MaxUploadSize

-lp 选项

指定监听器端口。

语法

qauagent -lp *number* ...

缺省值

5001

注释

监听器监听来自 MobiLink 服务器的 UDP 通知所使用的端口号。通知用于告知 QAnywhere UltraLite 代理有消息正在等待。QAnywhere UltraLite 代理还使用 UDP 端口向监听器发送控制命令。

另请参见

- “使用推式通知进行消息传递的方案”一节第 6 页
- “-push 选项”一节第 737 页

-mn 选项

为 MobiLink 用户指定新口令。

语法

qauagent -mp *password* ...

缺省值

无

注释

用于更改口令。

另请参见

- “MobiLink 用户” 《MobiLink - 客户端管理》
- “-mp 选项”一节第 732 页
- “-mu 选项”一节第 733 页

-mp 选项

为 MobiLink 用户指定 MobiLink 口令。

语法

qauagent -mp *password* ...

缺省值

无

注释

如果 MobiLink 服务器需要用户验证，则使用 -mp 提供 MobiLink 口令。

另请参见

- “MobiLink 用户” 《MobiLink - 客户端管理》
- “-mu 选项”一节第 733 页

-mu 选项

指定 MobiLink 用户名。

语法

qauagent -mu *username* ...

缺省值

客户端消息存储库 ID

注释

MobiLink 服务器进行验证时要使用 MobiLink 用户名。

如果指定的用户名不存在，则为您创建该用户名。

所有 MobiLink 用户名都必须在服务器消息存储库中注册。请参见“注册 QAnywhere 客户端用户名”一节第 31 页。

另请参见

- “MobiLink 用户” 《MobiLink - 客户端管理》
- “-id 选项”一节第 729 页
- “-mp 选项”一节第 732 页
- “远程 ID”一节 《MobiLink - 客户端管理》

-o 选项

将输出发送到指定的日志文件中。

语法

qauagent -o *logfile* ...

缺省值

无

注释

QAnywhere UltraLite 代理将输出记录到指定的文件名。如果文件已经存在，则新日志信息会附加到文件。监听器实用程序 (dblsn) 将输出记录到具有相同名称但包含后缀 *_lsn* 的文件中。

例如，如果将日志文件指定为 *c:\tmp\mylog.out*，则 qauagent 会将输出记录到 *c:\tmp\mylog.out*，而 dblsn 会记录到 *c:\tmp\mylog_lsn.out*。

另请参见

- “-ot 选项”一节第 735 页
- “-on 选项”一节第 734 页
- “-os 选项”一节第 734 页
- “-v 选项”一节第 740 页

-on 选项

指定 QAnywhere UltraLite 代理消息日志文件的最大大小，达到该大小后，将用扩展名 *.old* 重命名该文件并起用一个新文件。

语法

qauagent -on *size* [**k** | **m**] ...

缺省值

无

注释

size 是消息日志文件大小的最大值，以字节为单位。使用后缀 k 或 m 分别指定以千字节或兆字节为单位。最小值限制为 10K。

日志文件达到指定大小后，QAnywhere UltraLite 代理将用扩展名 *.old* 重命名输出文件，并以原始名称起用一个新文件。

注意

如果 *.old* 文件已经存在，则将覆盖它。为避免丢失旧的日志文件，请使用 -os 选项。

此选项不能与 -os 选项一起使用。

另请参见

- “-o 选项”一节第 733 页
- “-ot 选项”一节第 735 页
- “-os 选项”一节第 734 页
- “-v 选项”一节第 740 页

-os 选项

指定 QAnywhere UltraLite 代理消息日志文件的最大大小，达到该大小后，将创建并使用一个具有新名称的新日志文件。

语法

qauagent -os *size* [**k** | **m**] ...

缺省值

无

注释

size 是记录输出消息的文件大小的最大值。缺省单位是字节。使用后缀 k 或 m 分别指定以千字节或兆字节为单位。最小值限制为 10K。

在 QAnywhere UltraLite 代理将输出消息记录到文件之前，它会检查当前文件的大小。如果日志消息使得文件大小超过指定的大小，QAnywhere UltraLite 代理会将消息日志文件重命名为 *yymmddxx.mls*。其中，*xx* 是从 00 到 99 的连续字符，而 *yymmdd* 表示当前的年、月、日。

可以使用该选项删除旧的消息日志文件，从而释放磁盘空间。最新输出总是附加到由 -o 或 -ot 指定的文件中。

注意
此选项不能与 -on 选项一起使用。

另请参见

- “-o 选项”一节第 733 页
- “-ot 选项”一节第 735 页
- “-on 选项”一节第 734 页
- “-v 选项”一节第 740 页

-ot 选项

截断日志文件并将输出消息附加到该文件。

语法

qauagent -ot *logfile* ...

缺省值

无

注释

QAnywhere UltraLite 代理将输出记录到指定的文件名。如果该文件存在，则首先将其大小截断为 0。监听器实用程序 (dblsn) 将输出记录到具有相同名称但文件名包括后缀 *_lsn* 的文件。

例如，如果将日志文件指定为 *c:\tmp\mylog.out*，则 qauagent 会将输出记录到 *c:\tmp\mylog.out*，而 dblsn 会记录到 *c:\tmp\mylog_lsn.out*。

另请参见

- “-o 选项”一节第 733 页
- “-on 选项”一节第 734 页
- “-os 选项”一节第 734 页
- “-v 选项”一节第 740 页

-policy 选项

指定确定何时发生消息传输的策略。

语法

qauagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

缺省值

- 缺省的策略类型是 [自动]。
- [调度] 策略的缺省时间间隔是 900 秒（15 分钟）。

注释

QAnywhere 使用一种策略确定何时发生消息传输。*policy-type* 可以是下列值之一：

- **要求时** 仅当 QAnywhere 客户端应用程序进行适当的方法调用时，才传输消息。

  QAManager PutMessage() 方法使消息在本地排队。直到调用 QAManager TriggerSendReceive() 方法后，才将这些消息传输到服务器。同样，直到客户端调用 TriggerSendReceive() 方法后才将在服务器上等待的消息发送给客户端。

  使用 [要求时] 策略时，应用程序负责在从服务器收到推式通知后触发消息传输。推式通知会导致将系统消息发送到 QAnywhere 客户端。在应用程序中，可以选择通过调用 TriggerSendReceive() 响应此系统消息。

  有关示例内容，请参见“系统队列”一节第 64 页。

- **调度** 按指定的时间间隔传输消息。缺省值为 900 秒（15 分钟）。

  指定调度后，只要满足以下任何一个条件，代理就将每隔 *n* 秒钟执行一次消息传输：

  - 前一个时间间隔结束后在客户端消息存储库中放入了新的消息。
  - 前一个时间间隔结束后某条消息的状态发生了更改。当某条消息得到应用程序的确认时通常会发生这种情况。
  - 前一个时间间隔结束后接收到推式通知。
  - 前一个时间间隔结束后接收到网络状态更改通知。
  - 推式通知被禁用。

  可调用 TriggerSendReceive 方法来替换时间间隔。该方法会强制在时间间隔结束前传输消息。

- **自动** 发生下面描述的事件之一时，将传输消息。

  QAnywhere UltraLite 代理尝试保持消息队列为最新。下面的任一事件会导致将客户端的消息队列传送到服务器并将服务器的消息队列传送到客户端：

  - 调用 PutMessage()。
  - 调用 TriggerSendReceive()。
  - 出现推式通知。

有关通知的信息，请参见“使用推式通知进行消息传递的方案”一节第 6 页。

- 客户端上发生消息状态更改。例如，当应用程序从本地队列中检索消息时消息状态会从待执行更改为已接收。

- ***rules-file*** 指定客户端传输规则文件。传输规则文件可以指示一组更复杂的规则以确定何时传输消息。

  请参见“客户端传输规则”一节第 769 页。

另请参见

- “确定在客户端进行消息传输的时间”一节第 51 页
- “使用推式通知进行消息传递的方案”一节第 6 页

-push 选项

指定是否启用推式通知。

语法

qauagent -push *mode* ...

mode : **none** | **connected** | **disconnected**

缺省值

connected

选项

模式	说明
none	为该代理禁用推式通知。不启动监听器 (dblsn)。
connected	使用持久连接通过 TCP/IP 为该代理启用推式通知。通过 qauagent 启动监听器 (dblsn)，并且尝试与 MobiLink 服务器保持持久连接。当客户端设备没有公共 IP 地址或者当 MobiLink 服务器受到不允许 UDP 消息通过的防火墙保护时，此模式很有用。这是缺省设置。
disconnected	不使用持久连接通过 UDP 为该代理启用推式通知。通过 qauagent 启动监听器 (dblsn)，但不与 MobiLink 服务器保持持久连接。而是由 UDP 监听器从 MobiLink 接收推式通知。当客户端设备具有一个公共 IP 地址并且可由 UDP 或 SMS 访问时，此模式在以下情况下很有用： ● 客户端设备正在使用拨号网络并且连接时间的费用需要考虑。 ● 通过 QAnywhere 的消息流量很小。持久的 TCP/IP 连接会消耗网络服务器的资源，因此会影响到可伸缩性。 ● 客户端设备的网络覆盖率不可靠。在可以连接时，可使用自动策略传输消息。在这种环境中尝试维持持久连接并不实用，并且会浪费 CPU 资源。

注释

如果不想使用通知，将此选项设置为 [无]。这样将不需要在客户端部署 *dblsn.exe* 可执行程序。

有关不使用通知的 QAnywhere 说明，请参见“简单消息传递方案”一节第 4 页。

如果使用 UDP，由于 ActiveSync 的 UDP 实现的限制，不能在断开连接模式下通过 ActiveSync 使用推式通知。

另请参见

- “使用推式通知”一节第 34 页
- “启动 QAnywhere 代理”一节第 48 页
- “有关推式通知的通知”一节第 65 页
- “-lp 选项”一节第 731 页

-q 选项

以安静模式启动 QAnywhere UltraLite 代理，并且将窗口最小化到系统任务栏。

语法

qauagent -q ...

缺省值

无

注释

当使用 -q 以安静模式启动 QAnywhere UltraLite 代理时，主窗口将最小化到系统任务栏中。此外，用于消息存储库的数据库服务器可通过 -qi 选项来启动。

另请参见

- “-qi 选项”一节第 738 页

-qi 选项

以安静模式启动 QAnywhere UltraLite 代理，并且窗口完全隐藏。

语法

qauagent -qi ...

缺省值

无

注释

以安静模式启动 QAnywhere UltraLite 代理时，在 Windows 桌面操作系统中，主窗口将最小化到系统任务栏中，而在 Windows Mobile 中，主窗口将隐藏。此外，用于消息存储库的数据库服务器可通过 -qi 选项来启动。

对于某些 Windows Mobile 应用程序，安静模式很有用，因为它能防止某个应用程序在 Windows Mobile 达到 32 个并发进程的限制时关闭。安静模式允许 QAnywhere UltraLite 代理像服务一样运行。

在 -qi 安静模式下时，只能通过键入 **qastop** 来停止 QAnywhere UltraLite 代理。

另请参见

- “qastop 实用程序”一节第 743 页
- “-q 选项”一节第 738 页

-si 选项

初始化数据库以用作客户端消息存储库。

语法

qauagent -c "*connection-string*" -si ...

缺省值

无。只使用一次此选项以初始化客户端消息存储库。

注释

使用此选项之前，必须创建 UltraLite 数据库。使用 -si 时，QAnywhere UltraLite 代理使用数据库对象初始化数据库（例如 QAnywhere 系统表），然后立即退出。

运行 -si 时，必须使用 -c 选项指定连接字符串，来指示要初始化的数据库。在 -c 选项中指定的连接字符串还应指定一个具有 DBA 特权的用户 ID。如果没有指定用户 ID 和口令，则使用缺省的用户 DBA 和口令 SQL。

-si 选项为客户端消息存储库创建一个名为 ml_qa_user 的数据库用户，口令为 qanywhere。名为 ml_qa_user 的用户具有仅适用于 QAnywhere 应用程序的权限。如果不更改此数据库用户名和口令，启动 qauagent 时就不需要在 -c 选项中指定 pwd 或 uid。如果更改了其中一个，必须在 qauagent 命令行的 -c 选项中提供 uid 和/或 pwd。

> **注意**
> 您应该更改缺省口令。可使用 GRANT 语句进行更改。请参见“更改口令”一节 《SQL Anywhere 服务器 - 数据库管理》。

-si 选项不向客户端消息存储库提供 ID。运行 -si 或下次运行 qauagent 时可以使用 -id 选项指派 ID；如果不这样做，缺省情况下 qauagent 会将设备名指派为 ID。

如果创建了消息存储库但没有使用 ID 进行设置，相对于消息存储库的本地 QAnywhere 应用程序可以发送和接收消息，但不能与远程 QAnywhere 应用程序交换消息。指派 ID 后，远程消息传递将也可以发生。

另请参见

- “设置客户端消息存储库”一节第 23 页
- “创建安全的客户端消息存储库”一节第 132 页

示例

下面的命令连接到名为 *qaclient.db* 的数据库并将其初始化为 QAnywhere 客户端消息存储库。初始化完成后，QAnywhere UltraLite 代理立即退出。

```
qauagent -si -c "DBF=qaclient.db"
```

-v 选项

允许您指定消息日志文件中记录的信息和 QAnywhere UltraLite 代理控制台上显示的信息。

语法

qauagent -v *levels* ...

缺省值

最低详细程度

注释

-v 选项影响日志文件和控制台。只有在 qauagent 命令行中指定 -o 或 -ot 时才能有一个消息日志。

高详细程度可能会影响性能，通常应该仅在开发阶段使用高详细程度。

如果仅指定 -v，将记录最少的信息。

levels 的值如下所示。一次可以使用一个或多个选项，例如 -vlm。

- **+** 打开所有记录选项。
- **l** 显示所有 MobiLink 监听器记录。这使 MobiLink 监听器 (dblsn) 以详细程度级别 -v3 启动。请参见“用于 Windows 设备的监听器实用程序”一节 《MobiLink - 服务器启动的同步》中的 -v 选项。
- **m** 显示所有同步记录。
- **n** 显示所有网络状态更改通知。QAnywhere UltraLite 代理从监听器实用程序接收这些通知。
- **p** 显示所有消息推式通知。QAnywhere UltraLite 代理通过 MobiLink 服务器（包括 MobiLink Notifier）从监听器实用程序接收这些通知。
- **q** 显示用于表示传输规则的 SQL。
- **s** 显示由 QAnywhere UltraLite 代理初始化的所有消息同步。

另请参见

- “-o 选项”一节第 733 页
- “-ot 选项”一节第 735 页
- “-on 选项”一节第 734 页
- “-os 选项”一节第 734 页

-x 选项

指定网络协议和协议选项以便与 MobiLink 服务器通信。

语法

qauagent -x *protocol* [**(** *protocol-options*;... **)** ...]

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

注释

有关 *protocol-options* 的完整列表，请参见“MobiLink 客户端网络协议选项” 《MobiLink - 客户端管理》。

当 MobiLink 服务器与 QAnywhere UltraLite 代理不在同一设备上时，需要使用 -x 选项。

可以多次指定 -x。这使您可以为多个 MobiLink 服务器设置故障转移。设置故障转移时，QAnywhere UltraLite 代理按照在命令行中输入的顺序尝试连接 MobiLink 服务器。

QAnywhere UltraLite 代理还具有从 MobiLink 服务器（其上有可用于向客户端传输的消息）接收通知的监听器。此监听器仅使用所指定的第一个 MobiLink 服务器，并且不故障转移到其它服务器。

另请参见

- “MobiLink 客户端网络协议选项” 《MobiLink - 客户端管理》
- “加密通信流”一节第 134 页
- “传送层安全” 《SQL Anywhere 服务器 - 数据库管理》
- “设置故障转移机制”一节第 38 页
- “-fd 选项”一节第 728 页
- “-fr 选项”一节第 728 页

-xd 选项

指定 QAnywhere 代理应使用 MobiLink 服务器的动态寻址。

语法

qauagent -xd

注释

当指定 -xd 时，QAnywhere UltraLite 代理根据消息存储库属性确定 MobiLink 服务器的协议和地址。这意味着它可以动态确定单个 MobiLink 服务器的地址，其中服务器地址取决于正在运行 QAnywhere UltraLite 代理的设备目前使用的网络。

QAnywhere 应用程序必须初始化描述 MobiLink 服务器通信协议和地址的消息存储库属性，并建立与当前活动网络接口的关系。由于移动设备会在不同网络间切换，QAnywhere UltraLite 代理会检测处于活动状态的网络，并自动调整 MobiLink 服务器的通信协议和地址—而不必重新启动。

另请参见

- “客户端消息存储库属性”一节第 26 页

示例

以下示例会设置属性，以便根据设备所处的网络类型使用相应的 MobiLink 地址。例如，如果设备位于 LAN 上，则使用相应的 LAN 地址。

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress",
"host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress",
"host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Sierra Wireless AirCard 555 Adapter.type",
"WAN" );
```


qastop 实用程序

QAnywhere 停止实用程序用于停止正在以安静模式运行的 QAnywhere 代理或 QAnywhere 代理。

语法

qastop [*option* ...]

选项	说明
-id *id*	指定 QAnywhere 代理或 UltraLite 代理要停止的客户端消息存储库的 ID。
-wc *name*	指定 QAnywhere 代理或 UltraLite 代理要停止的窗口类名。

另请参见

- “-q 选项”一节第 718 页
- “-q 选项”一节第 738 页
- “停止 QAnywhere 代理”一节第 48 页

QAnywhere 属性

目录

客户端消息存储库属性

以下几节提供有关客户端消息存储库属性的信息。

预定义的客户端消息存储库属性

为方便起见，预定义了一些客户端消息存储库属性。预定义的消息存储库属性为：

- **ias_Adapters** 可用于连接到 MobiLink 服务器的网络适配器列表。此列表为字符串，由竖线分隔。
- **ias_MaxDeliveryAttempts** 定义此属性之后，将消息状态设为 UNRECEIVABLE 之前，在不进行确认的情况下能够接收消息的最大次数。缺省情况下，不定义此属性且此属性等效于值 -1，这意味着客户端库会继续不断地尝试传送未确认的消息。
- **ias_MaxDownloadSize** 下载增量大小。缺省情况下，QAnywhere 使用的最大下载大小为 -1，这意味着不存在最大值。如果源自服务器连接器和目标别名的消息超出指定的下载增量大小，消息将被拆分为若干较小的消息部分并在单独的下载中发送。此属性由 qaagent -idl 选项设置。请参见“-idl 选项”一节第 709 页。
- **ias_MaxUploadSize** 上载增量大小。缺省情况下，QAnywhere 以 256K 的增量上载消息。如果消息超出指定的上载增量大小，消息将被拆分为若干较小的消息部分并在单独的上载中发送。此属性由 qaagent -iu 选项设置。请参见“-iu 选项”一节第 710 页。
- **ias_Network** 有关当前正在使用的网络的信息。此属性可以读取但不得设置。ias_Network 为特殊属性。它具有多个内置属性，提供有关设备当前所使用的网络的信息。以下属性由 QAnywhere 自动设置：
  - **ias_Network.Adapter** 网卡（如果有）的当前名称。（网络连接建立后，指派给 Adapter 属性的网卡名显示在 [代理] 窗口中。）
  - **ias_Network.RAS** 当前 RAS 条目名（如果有）。
  - **ias_Network.IP** 指派给设备的当前 IP 地址（如果有）。
  - **ias_Network.MAC** 正在使用的网卡的当前 MAC 地址（如果有）。
- **ias_RASNames** 字符串。可用于连接到 MobiLink 服务器的 RAS 条目名列表。此列表用竖线分隔。
- **ias_StoreID** 消息存储库 ID。
- **ias_StoreInitialized** 如果为进行 QAnywhere 消息传递成功地初始化了消息存储库，则为 True；否则，为 False。

  请参见“-si 选项”一节第 719 页。
- **ias_StoreVersion** QAnywhere 定义的此消息存储库版本号。

有关管理预定义消息属性的信息，请参见：

- C++ API： “MessageStoreProperties 类”一节第 395 页
- .NET API： “MessageStoreProperties 类”一节第 223 页
- Java API： “MessageStoreProperties 接口”一节第 505 页
- SQL API： “消息存储库属性”一节第 674 页

自定义客户端消息存储库属性

QAnywhere 允许使用 QAnywhere C++、Java、SQL 或 .NET API 定义自己的客户端消息存储库属性。这些属性由连接到同一消息库的应用程序共享。它们还与服务器消息存储库同步，以便可用于此客户端的服务器端传输规则。

客户端消息存储库属性名不区分大小写。可以使用字母、数字和下划线序列，但第一个字符必须是字母。以下名称将被保留，不能用作消息存储库属性名：

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE（仅限 SQL Anywhere 消息存储库）
- 所有以 **ias_** 开头的名称

使用自定义客户端消息存储库属性特性

客户端消息存储库属性可以具有您定义的特性。可通过在属性名后面追加一个点，并在其后跟有特性名，来定义特性。此功能的主要用途是：可以在传输规则中使用有关您的网络的信息。

将 UltraLite 用作客户端消息存储库时，对属性特性的支持有限。UltraLite 消息存储库只支持预定义的 ias_Network 属性。

示例（仅限 SQL Anywhere）

以下简单示例说明了自定义客户端消息存储库属性特性的设置方法。在此例中，Object 属性有两个特性：Shape 和 Color。Shape 特性的值为 Round，Color 特性的值为 Blue。

```
// C++ example.
mgr->setStringStoreProperty( "Object.Shape", "Round" );
mgr->setStringStoreProperty( "Object.Color", "Blue" );

// C# example.
mgr.SetStoreStringProperty( "Object.Shape", "Round" );
mgr.SetStringStoreProperty( "Object.Color", "Blue" );
```

```
// Java example
mgr.setStringStoreProperty( "Object.Shape", "Round" );
mgr.setStringStoreProperty( "Object.Color", "Blue" );

-- SQL example
BEGIN
    CALL ml_qa_setstoreproperty( 'Object.Shape', 'Round' );
    CALL ml_qa_setstoreproperty( 'Object.Color', 'Blue' );
    COMMIT;
END
```

所有客户端消息存储库属性都有 Type 属性（无初始值）。Type 特性的值必须是另一个属性的名称。设置某个属性的 Type 特性时，该属性将继承指派给它的属性的特性。在以下示例中，Object 属性继承了 Circle 属性的特性。因此，Object.Shape 的值为 Round，Object.Color 的值为 Blue。

```
// C++ example
QAManager qa_manager;
qa_manager->setStoreStringProperty( "Circle.Shape", "Round" );
qa_manager->setStoreStringProperty( "Circle.Color", "Blue" );
qa_manager->setStoreStringProperty( "Object.Type", "Circle" );

// C# example
QAManager qa_manager;
qa_manager.SetStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.SetStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.SetStringStoreProperty( "Object.Type", "Circle" );

// Java example
QAManager qa_manager;
qa_manager.setStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.setStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.setStringStoreProperty( "Object.Type", "Circle" );

-- SQL example
BEGIN
    CALL ml_qa_setstoreproperty( 'Circle.Shape', 'Round' );
    CALL ml_qa_setstoreproperty( 'Circle.Color', 'Blue' );
    CALL ml_qa_setstoreproperty( 'Object.Type', 'Circle');
    COMMIT;
END
```

示例

以下 C# 示例介绍了如何使用消息存储库属性，将您的网络信息提供给传输规则。

假定您的 Windows 膝上型计算机具有以下网络连接选项：LAN、Wireless LAN 和 Wireless WAN。名为 My LAN Card 的网卡提供通过 LAN 对网络的访问。名为 My Wireless LAN Card 的网卡提供通过 Wireless LAN 对网络的访问。名为 My Wireless WAN Card 的网卡提供通过 Wireless WAN 对网络的访问。

假定您要开发一个消息传递应用程序，该应用程序在使用 LAN 或 Wireless LAN 连接时，将所有消息都发送到服务器，在使用 Wireless WAN 连接时，仅发送高优先级消息。将高优先级消息定义为优先级大于等于 7 的消息。

首先，查找网络适配器的名称。一旦插入网卡并安装了驱动程序，网络适配器的名称就固定了。要查找特殊网卡的名称，可通过该适配器连接到网络，然后使用 -vn 选项运行 qaagent。QAnywhere 代理采用以下形式显示网络适配器名：

```
"Listener thread received message '[netstat] network-adapter-name !...'
```


其次，为每个网络类型分别定义三个客户端消息存储库属性：LAN、WLAN 和 WWAN。为每个属性指派一个 Cost 特性。Cost 特性是 1 到 3 之间的值，表示网络的使用成本。值 1 表示最低成本。

```
QAManager qa_manager;
qa_manager.SetStoreProperty( "LAN.Cost", "1" );
qa_manager.SetStoreProperty( "WLAN.Cost", "2" );
qa_manager.SetStoreProperty( "WWAN.Cost", "3" );
```

然后，定义三个客户端消息存储库属性（每个使用的网卡定义一个）。属性名必须与网卡名匹配。通过向 Type 特性指派网络类型，为每个属性指派适当的网络分类。因此，每个属性都将继承为其指派的网络类型的特性。

```
QAManager qa_manager;
qa_manager.SetStoreProperty( "My LAN Card.Type", "LAN" );
qa_manager.SetStoreProperty( "My Wireless LAN Card.Type", "WLAN" );
qa_manager.SetStoreProperty( "My Wireless WAN Card.Type", "WWAN" );
```

在网络连接建立后，QAnywhere 将自动根据正在使用的网络将 ias_Network 属性的 Adapter 特性定义为 My LAN Card、My Wireless LAN Card 或 My Wireless WAN Card 中的一个。同样，它自动将 ias_Network 属性的 Type 特性定义为 My LAN Card、My Wireless LAN Card 或 My Wireless WAN Card 中的一个，以使 ias_Network 属性继承正在使用的网络的特性。

最后，创建以下传输规则。

```
automatic=ias_Network.Cost < 3 or ias_Priority >= 7
```

有关传输规则的详细信息，请参见“QAnywhere 传输和删除规则”第 761 页。

枚举客户端消息存储库属性

QAnywhere .NET、C++ 和 Java API 可以提供预定义和自定义客户端消息存储库属性的枚举。

.NET 示例

请参见“GetStorePropertyNames 方法”一节第 281 页。

```
// qaManager is a QAManager instance.
IEnumerator propertyNames = qaManager.GetStorePropertyNames();
```

C++ 示例

请参见“beginEnumStorePropertyNames 函数”一节第 427 页。

```
// qaManager is a QAManager instance.
qa_store_property_enum_handle handle = qaManager-
>beginEnumStorePropertyNames();
qa_char propertyName[256];

if( handle != qa_null ) {
    while(qaManager->nextStorePropertyName(handle, propertyName, 255) != -1)
{
        // Do something with the message store property name.
    }
    // Message store properties cannot be set after
    // the beginEnumStorePropertyNames call
    // and before the endEnumStorePropertyNames call.
    qaManager->endEnumStorePropertyNames(handle);
}
```

Java 示例

请参见“getStorePropertyNames 方法”一节第 558 页。

```
// qaManager is a QAManager instance.
Enumeration propertyNames = qaManager.getStorePropertyNames();
```

在应用程序中管理客户端消息存储库属性

以下 QAManagerBase 方法可用于获取和设置客户端消息存储库属性。

管理客户端消息存储库属性的 C++ 方法

- qa_bool getBooleanStoreProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanStoreProperty(qa_const_string name, qa_bool value)
- qa_bool getByteStoreProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteStoreProperty(qa_const_string name, qa_byte value)
- qa_bool getShortStoreProperty(qa_const_string name, qa_short * value)
- qa_bool setShortStoreProperty(qa_const_string name, qa_short value)
- qa_bool getIntStoreProperty(qa_const_string name, qa_int * value)
- qa_bool setIntStoreProperty(qa_const_string name, qa_int value)
- qa_bool getLongStoreProperty(qa_const_string name, qa_long * value)
- qa_bool setLongStoreProperty(qa_const_string name, qa_long value)
- qa_bool getFloatStoreProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatStoreProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleStoreProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleStoreProperty(qa_const_string name, qa_double value)
- qa_int getStringStoreProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringStoreProperty(qa_const_string name, qa_const_string value)
- qa_store_property_enum_handle QAManagerBase::beginEnumStorePropertyNames()
- virtual qa_int QAManagerBase::nextStorePropertyName(qa_store_property_enum_handle h, qa_string buffer, qa_int bufferLen)
- virtual void QAManagerBase::endEnumStorePropertyNames(qa_store_property_enum_handle h)

请参见“QAManagerBase 类”一节第 425 页。

管理客户端消息存储库属性的 C# 方法

- Object GetStoreProperty(String name)
- void SetStoreProperty(String name, Object value)
- boolean GetBooleanStoreProperty(String name)
- void SetBooleanStoreProperty(String name, boolean value)
- byte GetByteStoreProperty(String name)
- void SetByteStoreProperty(String name, byte value)
- short GetShortStoreProperty(String name)
- void SetShortStoreProperty(String name, short value)
- int GetIntStoreProperty(String name)
- void SetIntStoreProperty(String name, int value)
- long GetLongStoreProperty(String name)
- void SetLongStoreProperty(String name, long value)
- float GetFloatStoreProperty(String name)
- void SetFloatStoreProperty(String name, float value)
- double GetDoubleStoreProperty(String name)
- void SetDoubleStoreProperty(String name, double value)
- String GetStringStoreProperty(String name)
- void SetStringStoreProperty(String name, String value)
- IEnumerator GetStorePropertyNames()

请参见“QAManagerBase 接口”一节第 257 页。

管理客户端消息存储库属性的 Java 方法

- boolean getBooleanStoreProperty(String name)
- void setBooleanStoreProperty(String name, boolean value)
- byte getByteStoreProperty(String name)
- void setByteStoreProperty(String name, byte value)
- double getDoubleStoreProperty(String name)
- void setDoubleStoreProperty(String name, double value)
- float getFloatStoreProperty(String name)
- void setFloatStoreProperty(String name, float value)
- int getIntStoreProperty(String name)
- void setIntStoreProperty(String name, int value)
- long getLongStoreProperty(String name)
- void setLongStoreProperty(String name, long value)
- short getShortStoreProperty(String name)
- void setShortStoreProperty(String name, short value)
- void setStringStoreProperty(String name, String value)
- String getStringStoreProperty(String name)
- java.util.Enumeration getStorePropertyNames()

请参见“QAManagerBase 接口”一节第 541 页。

管理客户端消息存储库属性的 SQL 存储过程

- ml_qa_getstoreproperty
- ml_qa_setstoreproperty

请参见“消息存储库属性”一节第 674 页。

服务器属性

可以在 Sybase Central 中或通过服务器管理请求来设置服务器属性。在所有情况下，服务器属性都存储在数据库中。请参见：

- “使用服务器管理请求设置服务器属性”一节第 187 页
- “通过 Sybase Central 设置服务器属性”一节第 754 页

服务器属性

- **ianywhere.qa.server.autoRulesEvaluationPeriod** 两次规则评测（包括消息传输和持久性规则）之间间隔的毫秒数。通常，在消息被传输到服务器存储库时会对规则进行动态评测，因此规则评测期限仅用于那些对时间性要求很高的规则。缺省值为 **60000**（1 分钟）。

- **ianywhere.qa.server.compressionLevel** 应用于 QAnywhere 连接器所接收的每条消息的缺省压缩量。压缩级别是介于 0 到 9 之间的整数，其中 0 表示无压缩，9 表示最大压缩级别。缺省值为 **0**。

  如果在连接器属性文件中也设置了连接器压缩级别，则该连接器的这一设置会被替换。请参见“配置 JMS 连接器属性”一节第 157 页。

- **ianywhere.qa.server.connectorPropertiesFiles**

  不建议使用的功能
  用 Sybase Central 替换。

  包含一个或多个文件的列表，指定连接外部消息系统（例如 JMS）的 QAnywhere 连接器的配置。缺省情况为无连接器。

  请参见“连接器”第 153 页。

- **ianywhere.qa.server.disableNotifications** 将其设置为 true 以禁用来自服务器的、有关待执行消息的通知。这样一来，当消息正在启动发往客户端的通知所需的服务器上等待被发往客户端时，应在服务器上进行的处理过程将会被禁用。在任何无法从服务器发送通知的设置中（例如，由于防火墙限制而不能发送通知时），将其设置为 true。缺省值为 false。

- **ianywhere.qa.server.logLevel** 消息传递的记录级别。属性值可以是 1、2、3 或 4。1 表示只记录错误消息。2 表示除错误之外还记录警告。3 表示除错误和警告之外还记录信息性消息。4 表示除以上各类消息之外还记录更为详细的信息性消息，包括由 MobiLink 服务器传输的每条 QAnywhere 消息的详细信息。缺省值为 **2**。

  这些记录消息将被输出到 MobiLink 服务器消息窗口。如果指定了 mlsrv11 -o 或 -ot 选项，消息将被输出到 MobiLink 服务器消息日志文件。

- **ianywhere.qa.server.id** 指定向其发送服务器管理请求的地址的代理部分。如果未设置该属性，则此值为 ianywhere.server。

- **ianywhere.qa.server.password.e** 指定用于验证服务器管理请求的口令。如果未设置该属性，则口令为 QAnywhere。

  请参见“服务器管理请求简介”一节第 172 页。

- **ianywhere.qa.server.scheduleDateFormat** 指定用于服务器端传输规则的日期格式。缺省日期格式为 yyyy-MM-dd。

字母	日期组成部分	示例
y	年	1996
M	年中的月份	July
d	月份中的日期	10

- **ianywhere.qa.server.scheduleTimeFormat** 指定用于服务器端传输规则的时间格式。缺省时间格式为 HH:mm:ss。

字母	日期组成部分	示例
a	AM/PM 标记	PM
H	一天中的小时，其值在 0 到 23 之间	0
k	一天中的小时，其值在 1 到 24 之间	24
K	AM/PM 中的小时，其值在 0 到 11 之间	0
h	AM/PM 中的小时，其值在 1 到 12 之间	12
m	小时中的分钟	30
s	分钟中的秒	55

- **ianywhere.qa.server.transmissionRulesFile**

不建议使用的功能
用 Sybase Central 替换。

用于指定消息的传输和持久性控制规则的文件。缺省情况下，不对消息进行过滤，且当消息的最终状态已传输到消息发出方时，该消息会被删除。

通过 Sybase Central 设置服务器属性

♦ **通过 Sybase Central 设置服务器属性**

1. 启动 Sybase Central：

   选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 选择 [**连接**] » [**使用 QAnywhere 11 连接**]。

3. 指定一个 [**ODBC 数据源名称**] 或 [**ODBC 数据源文件**]、[**用户 ID**] 和 [**口令**]（如果需要）。单击 [**OK**]。
4. 在左窗格中的 [**服务器消息存储库**] 下，选择数据源的名称。
5. 选择 [**文件**] » [**属性**]。

JMS 连接器属性

以下属性用于配置 JMS 连接器：

- **ianywhere.connector.nativeConnection** 实现连接器的 Java 类。它仅供 QAnywhere 内部使用，并且不应删除或修改。
- **ianywhere.connector.id（不建议使用）** 唯一标识连接器的标识符。缺省值为连接器属性 ianywhere.connector.address 的值。
- **ianywhere.connector.address** QAnywhere 客户端用来标识连接器位置的连接器地址。此地址还用作前缀，作用于显示在此连接器的 MobiLink 服务器消息窗口中所有记录的错误、警告和信息性消息。

  请参见“向 JMS 连接器发送 QAnywhere 消息”一节第 158 页。

  在 Sybase Central 中，您可在 [**连接器向导**] 的 [**连接器名称**] 页的 [**连接器名称**] 字段中设置此属性。
- **ianywhere.connector.incoming.priority** 以整数形式表示的优先级，指派给所有进来的消息。如果值未指定或为负数，则使用该类型连接器的缺省值。在 JMS 中，缺省使用 JMS 消息的优先级。在 Web 服务中，缺省值是 4。
- **ianywhere.connector.incoming.retry.max** 将 JMS 消息传输到 QAnywhere 消息存储库中时，在放弃之前连接器可进行重试的最大次数。在尝试失败达到最大次数后，会将此 JMS 消息转发到 ianywhere.connector.jms.deadMessageDestination 属性值中。缺省值为 -1，意味着连接器不放弃重试。
- **ianywhere.connector.incoming.ttl** 以整数形式表示的生存期（以毫秒为单位），指派给所有进来的消息。如果值未指定或为负数，则使用该类型连接器的缺省值。如果值为 0，则消息不会过期。在 JMS 中，使用 JMS 消息的到期时间来计算缺省值。在 Web 服务中，缺省值是 0。
- **ianywhere.connector.outgoing.deadMessageAddress** 无法处理消息时将消息发送到的地址。例如，当消息包含有格式错误或未知的 JMS 地址时，该消息会被标记为无法接收，并且消息副本会被发送到失效消息地址。

  如果未指定失效消息地址，此消息会被标记为无法接收，但不会发送消息副本。

  在 Sybase Central 中，通过在 [**连接器属性**] 窗口的 [**属性**] 选项卡中单击 [**新建**]，可设置此属性。
- **ianywhere.connector.logLevel** MobiLink 服务器消息窗口和消息日志文件中显示的连接器信息的数量。日志级别的值如下：
  - **1** 记录错误消息。
  - **2** 记录错误和警告消息。
  - **3** 记录错误、警告和信息性消息。
  - **4** 记录错误、警告、信息性消息和调试消息。

  在 Sybase Central 中，可在 [**连接器属性**] 窗口的 [**常规**] 选项卡的 [**记录级别**] 部分设置此属性。

  还可为所有连接器设置此属性。要在 Sybase Central 中进行此操作，请连接到服务器消息存储库，并选择 [**更改此消息存储库的属性**] 的任务。打开 [**服务器属性**] 选项卡。

- **ianywhere.connector.compressionLevel** 从 JMS 接收到的消息的缺省消息压缩因子：它是 0 到 9 之间的一个整数，0 表示无压缩，9 表示最大压缩。

  在 Sybase Central 中，可在 [**连接器属性**] 窗口的 [**常规**] 选项卡的 [**压缩级别**] 部分中设置此属性。

  还可为所有连接器设置此属性。要在 Sybase Central 中进行此操作，请连接到服务器消息存储库，选择 [**更改此消息存储库的属性**] 的任务，并打开 [**服务器属性**] 选项卡。

- **ianywhere.connector.jms.deadMessageDestination** JMS 消息在无法转换为 QAnywhere 消息时，将被发送到的地址。如果 JMS 消息是不受支持的类的实例，或者 JMS 消息未指定 QAnywhere 地址，或者发生意外的 JMS 提供程序异常，或者发生意外的 QAnywhere 异常，则有可能发生这种情况。

  在 Sybase Central 中，可在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**其它**] 部分的 [**失效的消息目标**] 字段中设置此属性。

- **ianywhere.connector.outgoing.retry.max** 从 QAnywhere 到外部消息传递系统进行消息传递的缺省重试次数。缺省值为 5。指定 0 可使连接器始终进行重试。

  在 Sybase Central 中，通过在 [**连接器属性**] 窗口的 [**属性**] 选项卡中单击 [**新建**]，可设置此属性。

- **ianywhere.connector.runtimeError.retry.max** 连接器对导致 RuntimeException 的消息进行重试的次数。如果指定了失效消息队列，则该消息会被放置到此队列中。否则，该消息将标记为无法接收并被跳过。指定值 0 可使服务器始终不放弃。

- **ianywhere.connector.startupType** 启动类型可以是自动、手工或禁止。

- **xjms.jndi.authName** 用于连接到外部 JMS JNDI 名称服务的验证名称。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**用户名**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**用户名**] 字段中设置此属性。

- **xjms.jndi.factory** 用于访问外部 JMS JNDI 名称服务的工厂名称。在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**JNDI 工厂**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**JNDI 工厂**] 字段中设置此属性。

- **xjms.jndi.password.e** 用于连接到外部 JMS JNDI 名称服务的验证口令。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**口令**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**口令**] 字段中设置此属性。

- **xjms.jndi.url** 用于访问 JMS JNDI 名称服务的 URL。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**名称服务 URL**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**URL**] 字段中设置此属性。

- **xjms.password.e** 用于连接到外部 JMS 提供程序的验证口令。

- **xjms.queueConnectionAuthName** 用于连接到外部 JMS 队列连接的用户 ID。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 队列设置**] 页的 [**用户名**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**队列**] 部分中的 [**用户名**] 字段中设置此属性。

- **xjms.queueConnectionPassword.e** 用于连接到外部 JMS 队列连接的口令。

在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 队列设置**] 页的 [**口令**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**队列**] 部分中的 [**口令**] 字段中设置此属性。

- **xjms.queueFactory** 外部 JMS 提供程序队列的工厂名称。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 队列设置**] 页的 [**队列工厂**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**队列**] 部分中的 [**队列工厂**] 字段中设置此属性。

- **xjms.receiveDestination** 连接器用来监听从 JMS 到 QAnywhere 客户端的消息的队列名。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**连接器名称**] 页的 [**接收器目标**] 字段中设置此属性。

- **xjms.topicFactory** 外部 JMS 提供程序的主题工厂名称。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 主题设置**] 页的 [**主题工厂**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**主题**] 部分中的 [**主题工厂**] 字段中设置此属性。

- **xjms.topicConnectionAuthName** 用于连接到外部 JMS 主题连接的用户 ID。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 主题设置**] 页的 [**用户名**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**主题**] 部分中的 [**用户名**] 字段中设置此属性。

- **xjms.topicConnectionPassword.e** 用于连接到外部 JMS 主题连接的口令。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 主题设置**] 页的 [**口令**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**主题**] 部分中的 [**口令**] 字段中设置此属性。

- **ianywhere.connector.nativeConnection** 实现连接器的 Java 类。它仅供 QAnywhere 内部使用，并且不应删除或修改。

- **ianywhere.connector.id（不建议使用）** 唯一标识连接器的标识符。缺省值为连接器属性 ianywhere.connector.address 的值。

- **ianywhere.connector.address** QAnywhere 客户端用来标识连接器位置的连接器地址。此地址还用作前缀，作用于显示在此连接器的 MobiLink 服务器消息窗口中所有记录的错误、警告和信息性消息。

  请参见“向 JMS 连接器发送 QAnywhere 消息”一节第 158 页。

  在 Sybase Central 中，您可在 [**连接器向导**] 的 [**连接器名称**] 页的 [**连接器名称**] 字段中设置此属性。

- **ianywhere.connector.incoming.priority** 以整数形式表示的优先级，指派给所有进来的消息。如果值未指定或为负数，则使用该类型连接器的缺省值。在 JMS 中，缺省使用 JMS 消息的优先级。在 Web 服务中，缺省值是 4。

- **ianywhere.connector.incoming.retry.max** 将 JMS 消息传输到 QAnywhere 消息存储库中时，在放弃之前连接器可进行重试的最大次数。在尝试失败达到最大次数后，会将此 JMS 消息转发到 ianywhere.connector.jms.deadMessageDestination 属性值中。缺省值为 -1，意味着连接器不放弃重试。

- **ianywhere.connector.incoming.ttl** 以整数形式表示的生存期（以毫秒为单位），指派给所有进来的消息。如果值未指定或为负数，则使用该类型连接器的缺省值。如果值为 0，则消息不会过期。在 JMS 中，使用 JMS 消息的到期时间来计算缺省值。在 Web 服务中，缺省值是 0。

- **ianywhere.connector.outgoing.deadMessageAddress** 无法处理消息时将消息发送到的地址。例如，当消息包含有格式错误或未知的 JMS 地址时，该消息会被标记为无法接收，并且消息副本会被发送到失效消息地址。

  如果未指定失效消息地址，此消息会被标记为无法接收，但不会发送消息副本。

  在 Sybase Central 中，通过在 [**连接器属性**] 窗口的 [**属性**] 选项卡中单击 [**新建**]，可设置此属性。

- **ianywhere.connector.logLevel** MobiLink 服务器消息窗口和消息日志文件中显示的连接器信息的数量。日志级别的值如下：

  - **1** 记录错误消息。
  - **2** 记录错误和警告消息。
  - **3** 记录错误、警告和信息性消息。
  - **4** 记录错误、警告、信息性消息和调试消息。

  在 Sybase Central 中，可在 [**连接器属性**] 窗口的 [**常规**] 选项卡的 [**记录级别**] 部分设置此属性。

  还可为所有连接器设置此属性。要在 Sybase Central 中进行此操作，请连接到服务器消息存储库，并选择 [**更改此消息存储库的属性**] 的任务。打开 [**服务器属性**] 选项卡。

- **ianywhere.connector.compressionLevel** 从 JMS 接收到的消息的缺省消息压缩因子：它是 0 到 9 之间的一个整数，0 表示无压缩，9 表示最大压缩。

  在 Sybase Central 中，可在 [**连接器属性**] 窗口的 [**常规**] 选项卡的 [**压缩级别**] 部分中设置此属性。

  还可为所有连接器设置此属性。要在 Sybase Central 中进行此操作，请连接到服务器消息存储库，选择 [**更改此消息存储库的属性**] 的任务，并打开 [**服务器属性**] 选项卡。

- **ianywhere.connector.jms.deadMessageDestination** JMS 消息在无法转换为 QAnywhere 消息时，将被发送到的地址。如果 JMS 消息是不受支持的类的实例，或者 JMS 消息未指定 QAnywhere 地址，或者发生意外的 JMS 提供程序异常，或者发生意外的 QAnywhere 异常，则有可能发生这种情况。

  在 Sybase Central 中，可在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**其它**] 部分的 [**失效的消息目标**] 字段中设置此属性。

- **ianywhere.connector.outgoing.retry.max** 从 QAnywhere 到外部消息传递系统进行消息传递的缺省重试次数。缺省值为 5。指定 0 可使连接器始终进行重试。

  在 Sybase Central 中，通过在 [**连接器属性**] 窗口的 [**属性**] 选项卡中单击 [**新建**]，可设置此属性。

- **ianywhere.connector.runtimeError.retry.max** 连接器对导致 RuntimeException 的消息进行重试的次数。如果指定了失效消息队列，则该消息会被放置到此队列中。否则，该消息将标记为无法接收并被跳过。指定值 0 可使服务器始终不放弃。

- **ianywhere.connector.startupType** 启动类型可以是自动、手工或禁止。

- **xjms.jndi.authName** 用于连接到外部 JMS JNDI 名称服务的验证名称。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**用户名**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**用户名**] 字段中设置此属性。

- **xjms.jndi.factory** 用于访问外部 JMS JNDI 名称服务的工厂名称。在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**JNDI 工厂**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**JNDI 工厂**] 字段中设置此属性。
- **xjms.jndi.password.e** 用于连接到外部 JMS JNDI 名称服务的验证口令。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**口令**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**口令**] 字段中设置此属性。
- **xjms.jndi.url** 用于访问 JMS JNDI 名称服务的 URL。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JNDI 设置**] 页的 [**名称服务 URL**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**JNDI**] 部分中的 [**URL**] 字段中设置此属性。
- **xjms.password.e** 用于连接到外部 JMS 提供程序的验证口令。
- **xjms.queueConnectionAuthName** 用于连接到外部 JMS 队列连接的用户 ID。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 队列设置**] 页的 [**用户名**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**队列**] 部分中的 [**用户名**] 字段中设置此属性。
- **xjms.queueConnectionPassword.e** 用于连接到外部 JMS 队列连接的口令。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 队列设置**] 页的 [**口令**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**队列**] 部分中的 [**口令**] 字段中设置此属性。
- **xjms.queueFactory** 外部 JMS 提供程序队列的工厂名称。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 队列设置**] 页的 [**队列工厂**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**队列**] 部分中的 [**队列工厂**] 字段中设置此属性。
- **xjms.receiveDestination** 连接器用来监听从 JMS 到 QAnywhere 客户端的消息的队列名。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**连接器名称**] 页的 [**接收器目标**] 字段中设置此属性。
- **xjms.topicFactory** 外部 JMS 提供程序的主题工厂名称。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 主题设置**] 页的 [**主题工厂**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**主题**] 部分中的 [**主题工厂**] 字段中设置此属性。
- **xjms.topicConnectionAuthName** 用于连接到外部 JMS 主题连接的用户 ID。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 主题设置**] 页的 [**用户名**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**主题**] 部分中的 [**用户名**] 字段中设置此属性。
- **xjms.topicConnectionPassword.e** 用于连接到外部 JMS 主题连接的口令。

  在 Sybase Central 中，可在 [**连接器向导**] 的 [**JMS 主题设置**] 页的 [**口令**] 字段中设置此属性；还可以在 [**连接器属性**] 窗口的 [**JMS**] 选项卡的 [**主题**] 部分中的 [**口令**] 字段中设置此属性。

QAnywhere 传输和删除规则

目录

规则语法

规则语法

每个规则的形式如下：

schedules=*condition*

调度语法

调度语法

schedules : { **AUTOMATIC** | *schedule-spec* ,... }

schedule-spec :
{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
[**ON** { **(** *day-of-week*, ... **)** | **(** *day-of-month*, ... **)** }]
[**START DATE** *start-date*]

参数

- **AUTOMATIC** 对于传输规则，当消息更改状态或网络状态发生变化时，将对规则进行评测。对于删除规则，当消息传输启动时，满足删除规则条件的消息将被删除。
- **schedule-spec** 与 AUTOMATIC 不同，调度说明规定了评测条件的时间。在这些调度的时间，将对相应条件进行评测。
- **START TIME** 调度事件的每天中的最初调度时间。如果指定了 START DATE，则 START TIME 引用该日期。如果未指定 START DATE，则 START TIME 在当天（除非该时间已经过去）和随后的每一天（如果调度中含有 EVERY 或 ON）。
- **BETWEEN ...AND ...** 显示一天中的一段时间，在该时间段之外，没有调度的时间。如果指定了 START DATE，则调度的时间直到该日期才存在。
- **EVERY** 连续调度事件之间的间隔。调度的事件仅在当天的 START TIME 之后或在 BETWEEN ...AND 指定的范围内发生。
- **ON** 调度事件发生日的列表。如果指定了 EVERY，则缺省为每天。事件发生日可指定为周内某日或月内某日。

  周内某日是周一、周二等等。您还可以使用完整形式的日期，例如 Monday。如果您所使用的语言不是英语，不是连接字符串中客户端请求的语言，也不是服务器窗口中出现的语言，则您必须使用完整形式的日期名称。

  月内某日是从 0 到 31 的整数。0 值表示任何月的最后一天。
- **START DATE** 调度事件开始发生的日期。缺省为当前日期。

用法

可以为给定条件创建多个调度。这样可以实现复杂的调度。

如果调度说明的定义中含有 EVERY 或 ON，则调度说明将反复出现；如果未使用这些保留字，则调度最多指定一次。试图创建开始时间为过去时间的非反复出现的调度将会出错。

每到调度时间，都计算关联的条件，然后计算下次调度时间和日期。

通过检查一个或多个调度以及查找未来的下次调度时间来计算下次调度时间。如果调度指定每分钟一次，但计算条件需要 65 秒，则调度将每两分钟运行一次。如果想让执行重叠进行，则必须创建多个规则。

1. 如果使用了 EVERY 子句，则查看下次调度时间是否在当天，以及是否在 BETWEEN ...AND 范围的开始时间。如果是，即为下次调度时间。
2. 如果下一个调度时间不在当天，请查找下一个执行事件的日期。
3. 查找此日期的 START TIME，或 BETWEEN ...AND 范围的开始时间。

QAnywhere 调度语法派生自 SQL Anywhere CREATE EVENT 调度语法。

关键字不区分大小写。

另请参见

- “CREATE EVENT 语句”一节 《SQL Anywhere 服务器 - SQL 参考》

示例

以下示例服务器传输规则文件适用于由客户端消息存储库 ID sample_store_id 标识的客户端。它创建一个双重调度：高优先级消息每小时发送一次。调度为 `every 1 hours` 而条件是 `ias_priority=9`。此外，在上午 8 点到 9 点之间的时间内，高优先级消息每分钟发送一次。

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
   START TIME '06:00:00' EVERY 1 hours = ias_Priority = 9
   BETWEEN '08:00:00' AND '09:00:00' EVERY 1 minutes = ias_Priority = 9
```

条件语法

QAnywhere 条件使用类似 SQL 的语法。将针对消息存储库中的消息对条件进行评测。条件的评测结果可以是 true、false 或 unknown。如果条件为空，则所有消息都视为满足条件。条件可用于传输规则、删除规则和 QAnywhere 编程 API。

关键字和字符串比较不区分大小写。

语法

condition :
expression **IS** [**NOT**] **NULL**
| *expression compare expression*
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *pattern* [**ESCAPE** *character*]
| *expression* [**NOT**] **IN (** *string*, ... **)**
| **NOT** *condition*
| *condition* **AND** *condition*

| *condition* **OR** *condition*
| **(** *condition* **)**

compare: **= |> | < | >= | <= | <>**

expression:
constant
| *rule-variable*
| *-expression*
| *expression operator expression*
| **(** *expression* **)**
| *rule-function* **(** *expression*, ... **)**

constant: *integer* | *floating point number* | *string* | *boolean*

integer: An integer in the range -2**63 to 2**63-1.

floating point number: A number in scientific notation in the range 2.2250738585072e-308 to 1.79769313486231e+308.

string: A sequence of characters enclosed in single quotes. A single quote in a string is represented by two consecutive single quotes.

boolean: A statement that is **TRUE** or **FALSE**, **T** or **F**, **Y** or **N**, **1** or **0**.

operator: **+** | **-** | ***** | **/**

rule-variable:

请参见“规则变量”一节第 767 页。

rule-function:

请参见“规则函数”一节第 766 页。

参数

- **BETWEEN** BETWEEN 条件的评测结果可以是 true、false 或 unknown。在没有 NOT 关键字的情况下，如果 *expression* 大于等于开始表达式且小于等于结束表达式，则条件的评测结果为 true。

  NOT 关键字会使条件的含义相反，但对于 UNKNOWN，其含义保持不变。

  BETWEEN 条件相当于两个不等式的组合：

  [NOT] (*expression* **>=** *start-expression* **AND** *arithmetic-expression* **<=** *end-expr* **)**

  例如：

  - `age BETWEEN 15 AND 19` 等效于 `age >=15 AND age <= 19`
  - `age NOT BETWEEN 15 AND 19` 等效于 `age < 15 OR age > 19`。

- **IN** IN 条件是按照下面的规则计算的：
  - 如果 *expression* 不为空值且至少等于列表中的一个值，则返回 true。

- 如果 *expression* 为空值而值列表非空，或者如果至少其中一个值为空值且表达式不等于任何其它值，则返回 unknown。
- 如果没有值为空值，且 *expression* 不等于列表中的任何值，则返回 false。

使用 NOT 关键字将使 true 与 false 相互转换。

例如：

- `Country IN ( 'UK', 'US', 'France' )` 对于 `'UK'` 为真而对于 `'Peru'` 为假。它等效于以下语句：

```
( Country = 'UK' )        \
OR ( Country = 'US' )     \
OR ( Country = 'France' )
```

- `Country NOT IN ( 'UK', 'US', 'France' )` 对于 `'UK'` 为假而对于 `'Peru'` 为真。它等效于以下语句：

```
NOT ( ( Country = 'UK' )          \
      OR ( Country = 'US' )       \
      OR ( Country = 'France' ) )
```

- **LIKE** LIKE 条件的值可以是 true、false 或 unknown。

  无关键字 NOT 时，如果 *expression* 与 *like expression* 匹配，则条件的评测结果为 true。如果 *expression* 或 *like expression* 中有一个值为空值，则此条件为 unknown。

  NOT 关键字会使条件的含义相反，但对于 UNKNOWN，其含义保持不变。

  like expression 可包含任意数目的通配符。通配符包括：

通配符	匹配项
_（下划线）	任意一个字符
%（百分号）	包含零个或多个字符的任意字符串

  例如：

  - `phone LIKE 12%3` 对于 `'123'` 或 `'12993'` 为真而对于 `'1234'` 为假
  - `word LIKE 's_d'` 对于 `'sad'` 为真而对于 `'said'` 为假
  - `phone NOT LIKE '12%3'` 对于 `'123'` 或 `'12993'` 为假而对于 `'1234'` 为真

- **ESCAPE CHARACTER** ESCAPE CHARACTER 是单字符的字符串文字，其字符用于在 *pattern* 中将通配符字符（**_**、**%**）的特殊含义进行转义。例如：
  - `underscored LIKE '\_%' ESCAPE '\'` 对于 `'_myvar'` 为真而对于 `'myvar'` 为假。
- **IS NULL** 如果 rule-variable 为 unknown，则 IS NULL 条件的评测结果为 true；否则为 false。NOT 关键字使条件的含义相反。此条件的评测结果不能为 unknown。

规则函数

可在传输规则中使用以下函数：

语法	说明
DATEADD(*datepart*, *count*, *datetime* **)**	返回通过将多个日期部分添加到 datetime 中而产生的日期时间。*datepart* 可以是 year、quarter、month、week、day、hour、minute 或 second 之一。例如，以下示例加了两个月，得到值 2006-07-02 00:00:00.0： `DATEADD( month, 2, '2006/05/02' )`
DATEPART(*datepart*, *date* **)**	返回 datetime 值中一部分的值。datepart 可以是 year、quarter、month、week、day、dayofyear、weekday、hour、minute 或 second 之一。例如，以下示例获取以数字表示的五月份，得到值 5： `DATEPART( month, '2006/05/02' )`
DATETIME(*string* **)**	将字符串值转换为 datetime。字符串的格式必须为 'yyyy-mm-dd hh:nn:ss'。
LENGTH(*string* **)**	返回字符串中的字符数。
SUBSTRING(*string*, *start*, *length* **)**	返回字符串的子串。*start* 是要返回的子串的起始位置（以字符为单位）。*length* 是要返回的子串的长度（以字符为单位）。

示例

以下删除规则将删除 10 天前进入最终状态的所有消息：

```
START TIME '06:00:00' every 1 hours = ias_Status >= ias_FinalState \
  AND ias_StatusTime < DATEADD( day, -10, ias_CurrentTimestamp) \
  AND ias_TransmissionStatus = ias_Transmitted
```

规则变量

QAnywhere 规则变量可用于规则的条件部分。可使用以下内容作为规则变量：

- “消息属性”一节第 687 页
- “客户端消息存储库属性”一节第 26 页
- “规则引擎定义的变量”一节第 767 页

将属性用作规则变量

消息属性和消息存储库属性可用作传输规则变量。在这两种情况下，都可使用预定义属性或可创建自定义属性。如果拥有同名消息属性和消息存储库属性，则使用消息属性。要替换这种优先级，可按如下方式显式引用属性：

- 在消息存储库属性名前加上 ias_Store。
- 在消息属性名前加上 ias_Message。

例如，以下自动传输规则会选择自定义消息属性 urgent 设置为 yes 的所有消息：

```
automatic = ias_Message.urgent = 'yes'
```

以下自动传输规则在自定义消息存储库属性 transmitNow 设置为 yes 时选择消息：

```
automatic = ias_Store.transmitNow = 'yes'
```

规则引擎定义的变量

以下变量由规则引擎定义：

- **ias_Address** 消息的地址。例如，myclient\myqueue。
- **ias_ContentSize** 消息内容的大小。如果消息是文本消息，则为字符数。如果是二进制消息，则为字节数。
- **ias_ContentType** 消息类型：

IAS_TEXT_CONTENT	消息内容由 unicode 字符组成。
IAS_BINARY_CONTENT	将消息内容视为未解释的字节序列。

- **ias_CurrentDate** 当前日期。

  如果字符串以下面一种方法提供，就能与 ias_currentDate 进行比较：

  - 作为明确解释的格式字符串。
  - 作为根据为客户端消息存储库数据库设置的 date_format 数据库选项确定的字符串。

    请参见“设置数据库选项”一节 《SQL Anywhere 服务器 - 数据库管理》和“date_format 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ias_CurrentTime** 当前时间。

  如果字符串中时、分和秒采用 hh:mm:ss:sss 的格式用冒号分开，则该字符串就能与 ias_CurrentTime 进行比较。除非指定 **am** 或 **pm**，否则都采用 24 小时制。请参见“time_format 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ias_CurrentTimestamp** 当前时间戳（当前日期和时间）。请参见“time_format 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ias_Expires** 消息过期的日期和时间（如果不发送）。

- **ias_Network** 有关当前使用的网络的信息。ias_Network 是一个特殊的传输变量。它具有多个内置属性，提供有关设备当前所使用的网络的信息。

- **ias_Priority** 消息优先级：0 到 9 之间的一个整数，其中 0 表示最低优先级，9 表示最高优先级。

- **ias_Status** 消息的当前状态。其值可以是：

IAS_CANCELLED_STATE	消息已取消。
IAS_EXPIRED_STATE	消息在被预期接收者接收之前过期。
IAS_FINAL_STATE	消息已接收或过期。因此，`>=IAS_FINAL_STATE` 表示消息已接收或已过期，而 `<IAS_FINAL_STATE` 表示消息既未接收也未过期。
IAS_PENDING_STATE	消息还未由预期接收者接收。
IAS_RECEIVED_STATE	消息已由预期接收者接收。
IAS_UNRECEIVABLE_STATE	消息已被标记为无法接收，因为它可能是格式错误或尝试传送时失败次数过多。

- **ias_TransmissionStatus** 消息的同步状态。它可以是以下各项之一：

IAS_UNTRANSMITTED	消息尚未传输到其预期的接收者消息存储库。
IAS_TRANSMITTED	消息已经传输到其预期的接收者消息存储库。
IAS_DO_NOT_TRANSMIT	接收者消息存储库和源消息存储库相同，无需传输。
IAS_TRANSMITTING	消息已经传输到其预期的接收者，但此传输有待确认。有可能消息传输中断，而 QAnywhere 可能再次传输此消息。

示例

有关如何创建客户端存储库属性以及在传输规则中使用它们的示例，请参见“使用自定义客户端消息存储库属性特性”一节第 747 页。

消息传输规则

可以在服务器和客户端上指定传输规则。请参见：

- “客户端传输规则”一节第 769 页
- “服务器传输规则”一节第 770 页

客户端传输规则

客户端传输规则控制消息从客户端传输到服务器的行为。客户端传输规则由 QAnywhere 代理处理。

缺省情况下，QAnywhere 代理使用 [自动] 策略。可以通过将传输规则文件指定为 QAnywhere 代理的传输策略来更改和自定义此行为。

以下的部分 qaagent 命令行显示如何指定 QAnywhere 代理的规则文件：

```
qaagent -policy myrules.txt ...
```

有关如何编写传输规则的完整说明，请参见“规则语法”一节第 762 页。

有关策略的详细信息，请参见：

- “确定在客户端进行消息传输的时间”一节第 51 页
- “-policy 选项”一节第 716 页

有关客户端删除规则的信息，请参见“客户端删除规则”一节第 772 页。

传输规则文件拥有以下类型的条目：

- **规则** 每行只能输入一条规则。

  每条规则只能在一行中输入，但可以使用 \ 作为续行符。

- **注释** 以 # 或 ; 字符开头的行为注释。该行的所有字符都将被忽略。

请参见“规则语法”一节第 762 页和“条件语法”一节第 763 页。

还可以使用传输规则文件来确定何时从消息存储库中删除消息。

请参见“消息删除规则”一节第 772 页。

还可使用 Sybase Central QAnywhere 插件来创建 QAnywhere 代理规则文件。

示例

例如，以下客户端传输规则文件指定了在业务时间内只传输小的高优先级消息，在业务时间之外，则可以传输任何消息。此规则是自动的，这表示只要条件满足，就立即传输消息。此示例证明条件可以使用派生自消息的信息以及其它信息（例如，当前时间）。

```
automatic=(ias_ContentSize < 100000 AND ias_Priority > 7 ) \
 OR datepart(Weekday,ias_CurrentDate) in ( 1, 7 ) \
 OR ias_CurrentTime < datetime('8:00:00') \
 OR ias_CurrentTime > datetime('18:00:00')
```

服务器传输规则

设置缺省规则

可为特定的消息存储库或目标别名指定服务器传输规则，或为所有的客户端设置缺省规则。每个不具有显式传输规则的用户都使用缺省规则。

要设置缺省规则，应使用特殊的客户端名称 ianywhere.server.defaultClient。

调度服务器传输规则

指定调度服务器传输规则时，请牢记以下几点：

- 某一给定客户端进行同步时以及自动规则评测期限内，将评测该客户端的自动规则。
- 按照指定的调度评测给定客户端的调度规则。
- 对规则进行评测将使推式通知被发送到当前具有满足条件规则的消息的客户端。
- 每次客户端同步时，满足该客户端对应的自动规则的条件的消息将传输到该客户端。
- 当且仅当上次客户端同步后调度规则已使推式通知发送到客户端的情况下，所有满足下次同步时的调度规则条件的消息才会传输到该客户端。

使用传输规则文件指定服务器传输规则（不建议使用）

可以创建服务器传输规则文件并使用 QAnywhere 消息传送属性文件中的 ianywhere.qa.server.transmissionRulesFile 属性进行指定。

有关消息传递属性文件的详细信息，请参见“-m 选项”一节 《MobiLink - 服务器管理》。

要为特定客户端指定传输规则，需要在规则部分前加上带方括号的客户端消息存储库 ID。

可创建适用于所有用户的缺省服务器传输规则。

要指定缺省传输规则，使用以下行来开始一个部分：

```
[ianywhere.server.defaultClient]
```

要使新传输规则生效，必须重新启动 MobiLink 服务器。这仅适用于在传输规则文件中指定的传输规则。使用 Sybase Central 或服务器管理请求指定的服务器传输规则会立即生效。

有关服务器删除规则的信息，请参见“服务器删除规则”一节第 772 页。

示例

服务器传输规则文件的以下部分创建缺省规则，该缺省规则指定仅发送高优先级消息：

```
[ianywhere.server.defaultClient]
auto = ias_Priority > 6
```

在以下服务器传输规则文件的示例中，规则仅适用于由客户端消息存储库 ID sample_store_id 标识的客户端。

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
```



```
; store with id sample_store_id.
;
;    ias_Priority >= 7
;
; Messages with priority 7 or greater should always be
; transmitted.
;
;    ias_ContentSize < 100
;
; Small messages (messages less than 100 characters or
; bytes in size) should always be transmitted.
;
;    ias_CurrentTime < '8:00am' OR ias_CurrentTime > '6:00pm'
;
; Messages outside business hours should always be
; transmitted

auto = ias_Priority >= 7 OR ias_ContentSize < 100 \
    OR ias_CurrentTime < datetime('8:00:00') \
    OR ias_CurrentTime > datetime('18:00:00')
```

在以下示例中，规则仅适用于由客户端消息存储库 ID qanywhere 标识的客户端。

```
[qanywhere]
; This rule governs when messages are transmitted to the client
; store with id qanywhere.
;
;    tm_Subject not like '%non-business%'
;
; Messages with the property tm_Subject set to a value that
; includes the phrase 'non-business' should not be transmitted
;
;    ias_CurrentTime < '8:00:00' OR ias_CurrentTime > '18:00:00'
;
; Messages outside business hours should always be
; transmitted

auto = tm_Subject NOT LIKE '%non-business%' \
    OR ias_CurrentTime < datetime('8:00am') OR ias_CurrentTime >
datetime('6:00pm')
```

消息删除规则

删除规则确定客户端消息存储库和服务器消息存储库中的消息的持久性。

缺省行为

已超过到期时间但尚未接收到消息或将消息传输到任何位置时，则 QAnywhere 消息到期。消息到期之后，缺省的删除规则将删除该消息。如果已接收消息至少一次，但未对其确认，则可能会再次接收消息，即使已经超过到期时间。

客户端删除规则

缺省情况下，当消息的状态确定为已接收、已到期、已取消或无法送达且其最终状态已传输至服务器消息存储库时，该消息将从客户端消息存储库中删除。您可能希望更快地删除消息，或更长时间地保留消息。可通过在客户端传输规则文件中创建一个删除区来实现。必须在删除区的前面加上 **[system:delete]**。

有关确认的详细信息，请参见：

- .NET： “AcknowledgementMode 枚举”一节第 212 页
- C++： “AcknowledgementMode 类”一节第 386 页
- Java： “AcknowledgementMode 接口”一节第 498 页

有关客户端传输规则的详细信息，请参见“客户端传输规则”一节第 769 页。

下面是客户端传输规则文件中的删除规则部分的一个示例：

```
[system:delete]

; This rule governs when messages are deleted from the client
; store.
;
;    start time '1:00:00' on ( 'Sunday' )
;
; Messages are deleted every Sunday at 1:00 A.M.
;
;    ias_Status >= ias_FinalState
;
; Typically, messages are deleted when they reach a final
; state: received, unreceivable, expired, or canceled.

START TIME '1:00:00' ON ( 'Sunday' ) = ias_Status >= ias_FinalState
```

有关 ias_Status 的说明，请参见“规则变量”一节第 767 页。

服务器删除规则

缺省情况下，当消息的状态确定为已接收、已到期、已取消或无法送达且其最终状态已传回至消息发出方时，该消息将从服务器消息存储库中删除。出于某些目的（例如，审计），您可能希望将消息保留更长时间。

服务器端的删除规则适用于服务器消息存储库中的所有消息。

有关服务器传输规则的详细信息，请参见“服务器传输规则”一节第 770 页。

有关 ias_Status 的说明，请参见“规则变量”一节第 767 页。

术语表

术语表

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。在版本 10.0.0 中，Adaptive Server Anywhere 更名为 SQL Anywhere 服务器，SQL Anywhere Studio 更名为 SQL Anywhere。

另请参见： “SQL Anywhere”一节第 792 页

包

Java 中相关类的集合。

被引用对象

一种对象（如表），该对象在另一个对象（如视图）的定义中被直接引用。

另请参见： “主键”一节第 802 页

编码

也称作字符编码，编码是一种方法，通过该方法可以将字符集中的每个字符映射到一个或多个字节的信息，这些信息通常以十六进制数字表示。编码的一个例子是 UTF-8。

另请参见：

- “字符集”一节第 802 页
- “代码页”一节第 777 页
- “归类”一节第 781 页

标识符

用于引用数据库对象（如表或列）的字符串。标识符可以包含 A 到 Z、a 到 z、0 到 9、下划线 (_)、at 符号 (@)、数字符号 (#) 或美元符号 ($) 中的任何字符。

并发

同时执行两个或更多个独立并且可能存在竞争关系的进程。SQL Anywhere 会自动使用锁定来隔离事务，并确保每个并发应用程序看到的数据集均一致。

另请参见：

- “事务”一节第 789 页
- “隔离级别”一节第 780 页

参考数据库

MobiLink 中一种用于 UltraLite 客户端开发的 SQL Anywhere 数据库。在开发过程中，可以将一个 SQL Anywhere 数据库同时作为参考数据库和统一数据库使用。通过其它产品建立的数据库无法用作参考数据库。

参照完整性

遵守数据一致性控制规则（具体而言，不同表中主键值与外键值之间的关系）。若要实现参照完整性，每个外键中的值必须与被引用表中行的主键值相符。

另请参见：

- “主键”一节第 802 页
- “外键”一节第 794 页

策略

QAnywhere 中指定应在何时进行消息传输的方式。

插件模块

Sybase Central 中一种用于访问和管理产品的方法。当您安装相应的产品时，插件通常会自动安装并注册 Sybase Central。通常，插件在 Sybase Central 主窗口中作为顶级容器出现，并且使用产品本身的名称，如 SQL Anywhere。

另请参见： “Sybase Central”一节第 793 页

查询

一条或一组 SQL 语句，用于访问和/或操作数据库中的数据。

另请参见： “SQL”一节第 792 页

冲突解决

在 MobiLink 中，冲突解决是指一种逻辑，它指定当两个用户修改不同远程数据库上同一行时的处理方法。

重定向器

一种 Web 服务器插件，用于为客户端与 MobiLink 服务器之间的请求和响应选择发送路径。此插件还实现了负荷平衡和故障转移机制。

抽取

SQL Remote 复制中从统一数据库卸载相应结构和数据的行为。此信息用于初始化远程数据库。

另请参见： “复制”一节第 779 页

触发器

一种特殊形式的存储过程，用户运行修改数据的查询时会自动执行该存储过程。

另请参见：

- “行级触发器”一节第 781 页
- “语句级触发器”一节第 799 页
- “完整性”一节第 795 页

传输规则

QAnywhere 中用于确定何时进行消息传输、传输哪些消息以及应在何时删除消息的逻辑。

窗口

作为分析功能执行对象的行组。一个窗口可以包含一行、多行或所有行的数据，这些数据已根据窗口定义中提供的分组规格进行了分区。窗口会进行移动，以包括为输入中的当前行执行计算所需的行数或行范围。窗口结构的主要优点是，不需要执行附加查询就可以有机会对结果进行分组和分析。

创建者 ID

UltraLite Palm OS 应用程序中一种在创建应用程序时指派的 ID。

存储过程

存储过程是数据库中存储的一组 SQL 指令，用于在数据库服务器上执行一组操作或查询。

代理表

一种本地表，它所包含的元数据可以像访问本地表一样访问远程数据库服务器上的表。

另请参见： “元数据”一节第 800 页

代理 ID

另请参见： “客户端消息存储库 ID”一节第 785 页

代码页

代码页是一种将字符集的字符映射到数字表示的编码，数字表示通常是 0 到 255 之间的一个整数。例如，Windows 代码页 1252 就是一个代码页。就本文档而言，代码页和编码这两个术语可以互换。

另请参见：

- “字符集”一节第 802 页
- “编码”一节第 775 页
- “归类”一节第 781 页

DBA 权限

使用户能够在数据库中执行管理活动的权限级别。DBA 用户在缺省情况下具有 DBA 权限。

另请参见： “数据库管理员 (DBA)”一节第 791 页

dbspace

用于创建更多数据存储空间的附加数据库文件。一个数据库可以包含在最多 13 个独立的文件（一个初始文件和 12 个 dbspace）中。每个表及其索引必须包含在单个数据库文件中。SQL 命令 CREATE DBSPACE 可将新文件添加到数据库中。

另请参见： “数据库文件”一节第 792 页

动态 SQL

执行前由程序以编程方式生成的 SQL。UltraLite 动态 SQL 是一种专用于小型设备的 SQL 变体。

对象树

Sybase Central 中数据库对象的层次。对象树的顶层显示您的 Sybase Central 版本所支持的全部产品。每种产品展开后会显示其自己的对象子树。

另请参见： “Sybase Central”一节第 793 页

EBF

快速错误修正软件。快速错误修正软件是含有一个或多个错误修正软件的软件子集。错误修正软件列在更新程序的发行说明中。错误修正软件更新可能只适用于具有相同版本号的已安装软件。已对该软件执行了一些测试，但该软件尚未进行完全测试。除非您自己已验证了软件的适用性，否则不要随应用程序分发这些文件。

发布

MobiLink 或 SQL Remote 中一种用于标识将要同步的数据的数据库对象。在 MobiLink 中，发布仅存在于客户端。一个发布包括多个项目。SQL Remote 用户可以通过预订发布来接收发布。MobiLink 用户可以通过创建发布的同步预订来同步发布。

另请参见：

- “复制”一节第 779 页
- “项目”一节第 797 页
- “发布更新”一节第 778 页

发布更新

SQL Remote 复制中对一个数据库中的一个或多个发布所做更改的列表。发布更新将作为复制消息的一部分定期发送到远程数据库。

另请参见：

- “复制”一节第 779 页
- “发布”一节第 778 页

发布者

SQL Remote 复制中数据库内可以与其它复制数据库交换复制消息的单个用户。

另请参见： “复制”一节第 779 页。

FILE

SQL Remote 复制中一种使用共享文件来交换复制消息的消息系统。它对测试以及在没有显式消息传送系统的情况下进行的安装很有用。

另请参见“复制”一节第 779 页。

分析树

查询的代数表示。

服务

在 Windows 操作系统上，服务是在运行应用程序的用户 ID 未登录时的应用程序运行方式。

服务器管理请求

一种 QAnywhere 消息，其格式设置为 XML 并发送到 QAnywhere 系统队列，作为一种管理服务器消息存储库或监控 QAnywhere 应用程序的方法。

服务器启动的同步

一种从 MobiLink 服务器启动 MobiLink 同步的方式。

服务器消息存储库

QAnywhere 中在消息传输到客户端消息存储库或 JMS 系统之前服务器上用于临时存储消息的关系数据库。消息通过服务器消息存储库在各客户端之间进行交换。

复制

在物理上不相同的数据库之间共享数据。Sybase 有三种复制技术：MobiLink、SQL Remote 和复制服务器。

复制代理

请参见： “LTM”一节第 786 页

复制服务器

Sybase 的一种基于连接的复制技术，用于与 SQL Anywhere 和 Adaptive Server Enterprise 一起使用。它专用于在一些数据库之间进行接近实时的复制。

另请参见： “LTM”一节第 786 页

复制频率

SQL Remote 复制中一项针对每个远程用户的设置，它决定发布者的消息代理向该远程用户发送复制消息的频率应为多少。

另请参见： “复制”一节第 779 页。

复制消息

SQL Remote 或复制服务器中一种在发布数据库与预订数据库之间发送的通信。消息包含复制系统所需的数据、直通语句及信息。

另请参见：

- “复制”一节第 779 页
- “发布更新”一节第 778 页

隔离级别

一个事务中的操作对其它并发事务中的操作的可见程度。隔离级别有四级，编号依次为 0 至 3。第 3 级提供最高级别的隔离。级别 0 为缺省设置。SQL Anywhere 还支持以下三个快照隔离级别：快照、语句快照和只读语句快照。

另请参见： “快照隔离”一节第 785 页

个人服务器

与客户端应用程序在同一台计算机上运行的数据库服务器。个人数据库服务器通常由单个用户在一台计算机上使用，但它可以支持来自该用户的几个并发连接。

工作表

一种内部存储区域，用于在查询优化过程中存储中间结果。

故障切换

在活动服务器、系统或网络出现故障或意外终止时切换到冗余或备用的服务器、系统或网络。故障转移会自动进行。

关系数据库管理系统 (RDBMS)

一种以相关表的形式存储数据的数据库管理系统。

另请参见： “数据库管理系统 (DBMS)”一节第 791 页

规范化

对数据库模式的改进，目的在于按照基于关系数据库理论的规则消除冗余并改善组织。

归类

定义数据库中文本属性的字符集与排序顺序的组合。对于 SQL Anywhere 数据库，缺省归类取决于运行服务器时所使用的操作系统和语言；例如，英语 Windows 系统上的缺省归类为 1252LATIN1。归类（也称作归类序列）用于对字符串进行比较和排序。

另请参见：

- “字符集”一节第 802 页
- “代码页”一节第 777 页
- “编码”一节第 775 页

行级触发器

每更改一行即执行一次的触发器。

另请参见：

- “触发器”一节第 777 页
- “语句级触发器”一节第 799 页

回退日志

对在每个未提交的事务执行过程中所做更改的记录。当收到 ROLLBACK 请求或者系统出现故障时，未提交的事务会从数据库中回退，将数据库返回其原先的状态。每个事务都有一个单独的回退日志，事务完成时日志会被删除。

另请参见： “事务”一节第 789 页

iAnywhere JDBC 驱动程序

iAnywhere JDBC 驱动程序提供了一个 JDBC 驱动程序，与纯 Java jConnect JDBC 驱动程序相比，该驱动程序拥有一些性能优势和功能优点，但它不是纯 Java 解决方案。建议在大多数情况下使用 iAnywhere JDBC 驱动程序。

另请参见：

- “JDBC”一节第 782 页
- “jConnect”一节第 782 页

InfoMaker

一种报告和数据维护工具，它用于创建复杂的表格、报告、图形、交叉表和表，并创建将这些报告用作构件块的应用程序。

Interactive SQL

一种 SQL Anywhere 应用程序，用于查询和更改数据库中的数据以及修改数据库的结构。Interactive SQL 不但提供了一个用于输入 SQL 语句的窗格，还提供了一些用于返回有关查询处理过程的信息和结果集的窗格。

JAR 文件

Java 档案文件。一种压缩的文件格式，由一个或多个用于 Java 应用程序的包的集合组成。它将安装和运行 Java 程序所需的全部资源都放在一个压缩文件中。

Java 类

Java 中的主要代码结构单元。它是组合在一起的过程和变量的集合，将过程和变量组合在一起的原因是它们都与某个特定的可识别类别有关。

jConnect

JavaSoft JDBC 标准的 Java 实现。它为 Java 开发人员提供多层和异类环境中的本地数据库访问。但在大多数情况下，iAnywhere JDBC 驱动程序是首选的 JDBC 驱动程序。

另请参见：

- “JDBC”一节第 782 页
- “iAnywhere JDBC 驱动程序”一节第 781 页

JDBC

Java 数据库连接。一种 SQL 语言编程接口，它允许 Java 应用程序访问关系数据。首选的 JDBC 驱动程序是 iAnywhere JDBC 驱动程序。

另请参见：

- “jConnect”一节第 782 页
- “iAnywhere JDBC 驱动程序”一节第 781 页

基表

永久性的数据表。有时为区别于临时表和视图，会将这种表称作**基表**。

另请参见：

- “临时表”一节第 785 页
- “视图”一节第 789 页

基于会话的同步

一种同步类型，这种同步会使数据表示在统一数据库和远程数据库都一致。MobiLink 基于会话。

基于脚本的上载

MobiLink 中一种将上载过程自定义为使用日志文件的替代方法的方式。

基于 SQL 的同步

MobiLink 中一种使用 MobiLink 事件将表数据与支持 MobiLink 的统一数据库进行同步的方式。对于基于 SQL 的同步，可以直接使用 SQL，也可以使用面向 Java 和 .NET 平台的 MobiLink 服务器 API 返回 SQL。

基于文件的下载

在 MobiLink 中同步数据的一种方式，其中下载以文件的方式进行分发，从而支持脱机分发同步更改。

集成登录

一种登录功能，它允许将同一个用户 ID 和口令用于操作系统登录、网络登录和数据库连接。

监听器

一个程序 (dblsn)，用于 MobiLink 服务器启动的同步。监听器安装在远程设备上，它们被配置为在接收到来自通告程序的信息时启动针对设备的操作。

另请参见： “服务器启动的同步”一节第 779 页

检查点

将对数据库的所有更改都保存到数据库文件中的时间点。在其它时间，所提交的更改仅保存到事务日志中。

检查约束

对列或列集强制实施指定条件的一种限制。

另请参见：

- “约束”一节第 801 页
- “外键约束”一节第 795 页
- “主键约束”一节第 802 页
- “唯一约束”一节第 796 页

脚本

MobiLink 中为处理 MobiLink 事件而编写的代码。脚本通过编程方式控制数据交换，以满足业务需要。

另请参见： “事件模型”一节第 789 页

脚本版本

MobiLink 中为创建同步而一起应用的一组同步脚本。

校验

测试数据库、表或索引是否受到特定类型的文件损坏。

校验和

随数据库页本身一起记录的计算出的数据库页位数。校验和能够确保数据库页写入磁盘时位数相符，因此数据库管理系统可以通过它来验证数据库页的完整性。如果计数相符，即认为数据库页已成功写入。

镜像日志

另请参见： “事务日志镜像”一节第 790 页

角色

概念性数据库建模中从一个角度描述某种关系的动词或短语。您可以用两个角色来描述每种关系。例如，"包含" 和 "隶属于" 便是角色。

角色名

外键的名称。由于它命名外表和主表之间的关系，因此称作角色名。缺省情况下，角色名就是表名，除非其它外键已经使用该名称（在这种情况下，缺省的角色名是表名后接一个三位的唯一数字）。也可以自己创建角色名。

另请参见： “外键”一节第 794 页

局部临时表

一种临时表，仅在复合语句执行期间或连接结束之前存在。当您只需要将数据集装载一次时，局部临时表非常有用。缺省情况下，行会在提交时被删除。

另请参见：

- “临时表”一节第 785 页
- “全局临时表”一节第 788 页

客户端/服务器

一种软件体系结构，在这种体系结构中，一个应用程序（客户端）从另一个应用程序（服务器）获取信息并向该应用程序发送信息。这两个应用程序常位于通过网络连接的不同计算机上。

客户端消息存储库

QAnywhere 中一种用于在远程设备上存储消息的 SQL Anywhere 数据库。

客户端消息存储库 ID

QAnywhere 中一种对客户端消息存储库进行唯一标识的 MobiLink 远程 ID。

快照隔离

一种为发出读请求的事务返回数据的已提交版本的隔离级别。SQL Anywhere 提供了以下三种快照隔离级别：快照、语句快照和只读语句快照。使用快照隔离时，读操作不会阻塞写操作。

另请参见： “隔离级别”一节第 780 页

连接

关系系统中的一种基本操作，它通过比较指定列中的值将两个或更多个表中的行链接在一起。

连接 ID

用于标识客户端应用程序与数据库之间给定连接的唯一编号。可以使用以下 SQL 语句来确定当前连接 ID：

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

连接类型

SQL Anywhere 提供了四种类型的连接：交叉连接、键连接、自然连接和使用 ON 子句的连接。

另请参见： “连接”一节第 785 页

连接配置

连接到数据库所需的一组参数，如用户名、口令和服务器名称，它们在存储后即可方便地使用。

连接启动的同步

一种 MobiLink 服务器启动的同步，在这种同步下，连接发生变化时会启动同步。

另请参见： “服务器启动的同步”一节第 779 页

连接条件

一种影响连接结果的限制。您可以通过紧跟在连接语句的后面插入 ON 子句或 WHERE 子句来指定连接条件。对于自然连接和关键连接，SQL Anywhere 会生成连接条件。

另请参见：

- “连接”一节第 785 页
- “生成的连接条件”一节第 790 页

临时表

为临时存储数据而创建的表。有两种类型：全局临时表和局部临时表。

另请参见：

- “局部临时表”一节第 784 页
- “全局临时表”一节第 788 页

LTM

日志传送管理器（Log Transfer Manager，简称 LTM）也称作复制代理。LTM 是一个与 Replication Server 一起使用的程序，它读取数据库事务日志并将提交的更改发送到 Sybase 复制服务器。

请参见： “复制服务器”一节第 780 页

轮询

在 MobiLink 服务器启动的同步中，轻量级轮询器（例如 MobiLink 监听器）从通告程序请求推式通知的方式。

另请参见： “服务器启动的同步”一节第 779 页

逻辑索引

指向物理索引的引用（指针）。磁盘上不存储逻辑索引的索引结构。

命令文件

包含 SQL 语句的文本文件。命令文件可以手工建立，也可以通过数据库实用程序自动建立。例如，dbunload 实用程序会创建一个命令文件，其中包含重新创建给定数据库所需的 SQL 语句。

MobiLink

一种基于会话的同步技术，其设计用途是将 UltraLite 和 SQL Anywhere 远程数据库与统一数据库同步。

另请参见：

- “统一数据库”一节第 793 页
- “同步”一节第 793 页
- “UltraLite”一节第 794 页

MobiLink 服务器

运行 MobiLink 同步的计算机程序，即 mlsrv11。

MobiLink 监控器

一种用于监控 MobiLink 同步的图形化工具。

MobiLink 客户端

有两种 MobiLink 客户端。对于 SQL Anywhere 远程数据库，MobiLink 客户端是 dbmlsync 命令行实用程序。对于 UltraLite 远程数据库，MobiLink 客户端内置于 UltraLite 运行时库中。

MobiLink 系统表

MobiLink 同步所需的系统表。它们由 MobiLink 安装程序脚本安装到 MobiLink 统一数据库中。

MobiLink 用户

MobiLink 用户用于与 MobiLink 服务器进行连接。在远程数据库上创建 MobiLink 用户，然后在统一数据库中注册该用户。MobiLink 用户名完全独立于数据库用户名。

模式

数据库的结构，其中包括表、列和索引以及它们之间的关系。

内连接

一种连接，在这种连接中，仅当两个表都满足连接条件时行才会出现在结果集中。内连接是缺省设置。

另请参见：

- “连接”一节第 785 页
- “外连接”一节第 795 页

ODBC

开放式数据库连接。一种用于与数据库管理系统连接的标准 Windows 接口。ODBC 是 SQL Anywhere 所支持的几种接口之一。

ODBC 管理器

一种随 Windows 操作系统提供的 Microsoft 程序，用于设置 ODBC 数据源。

ODBC 数据源

用户要通过 ODBC 访问的数据的规范以及获取该数据时所需的信息。

PDB

Palm 数据库文件。

PowerDesigner

一种数据库建模应用程序。PowerDesigner 为设计数据库或数据仓库提供了结构化的方法。SQL Anywhere 包括 PowerDesigner 的 Physical Data Model 组件。

PowerJ

一种 Sybase 产品，用于开发 Java 应用程序。

QAnywhere

应用程序到应用程序的消息传递（包括移动设备到移动设备和移动设备与企业之间的消息传递），它使在移动或无线设备上运行的自定义程序能够与处在中央位置的服务器应用程序进行通信。

QAnywhere 代理

QAnywhere 中一种运行在客户端设备上的进程，用于监控客户端消息存储库和确定应在何时传输消息。

嵌入式 SQL

一种 C 语言程序编程接口。SQL Anywhere 嵌入式 SQL 是 ANSI 和 IBM 标准的实现。

轻量级轮询器

在 MobiLink 服务器启动的同步中，轮询来自 MobiLink 服务器的推式通知的设备应用程序。

另请参见： “服务器启动的同步”一节第 779 页

全局临时表

一种临时表，在被显式地删除之前，其数据定义对所有用户都可见。全局临时表允许用户各自打开一个表的相同实例。缺省情况下，行在提交时被删除，并且始终是在连接结束时被删除。

另请参见：

- “临时表”一节第 785 页
- “局部临时表”一节第 784 页

日志文件

SQL Anywhere 所维护的事务日志。该日志文件用于确保在出现系统或介质故障时可以恢复数据库、提高数据库性能以及使用 SQL Remote 实现数据复制。

另请参见：

- “事务日志”一节第 789 页
- “事务日志镜像”一节第 790 页
- “完全备份”一节第 795 页

散列

散列是一种将索引条目转化为键的索引优化。索引散列旨在通过将足够的行实际数据与其行 ID 包括在一起，以避免进行先查找行、后装载行然后再将行解出才能得出索引值的高开销操作。

上载

同步过程的一个阶段，在此阶段数据从远程数据库传送到统一数据库。

设备跟踪

在 MobiLink 服务器启动的同步中，允许使用标识设备的 MobiLink 用户名来对消息进行寻址的功能。

另请参见： “服务器启动的同步”一节第 779 页

实例化视图

实例化视图是指已计算并已存储在磁盘上的视图。实例化视图同时具有视图的特征（使用查询说明进行定义）和表的特征（可以对其执行大多数表操作）。

另请参见：

- “基表”一节第 782 页
- “视图”一节第 789 页

世代号

MobiLink 中的一种机制，用于强制远程数据库先上载数据，然后再应用任何其它下载文件。

另请参见： “基于文件的下载”一节第 783 页

事件模型

MobiLink 中组成同步的事件（如 begin_synchronization 和 download_cursor）序列。如果为事件创建了脚本，则会调用事件。

视图

一种作为对象存储在数据库中的 SELECT 语句。它使用户能够看到一个或多个表中的行子集或列子集。每当用户使用特定表或表组合的视图时，都将利用存储在这些表中的信息重新计算视图。视图对确保安全以及定制数据库信息的外观来使数据访问简单明了有帮助。

事务

组成一个逻辑工作单元的 SQL 语句序列。事务要么全部得到处理，要么根本不做处理。SQL Anywhere 支持事务处理，并内置了锁定功能，使并发事务能够访问数据库而又不损坏数据。事务要么以 COMMIT 语句结束，该语句使对数据的更改成为永久性更改；要么以 ROLLBACK 语句结束，该语句撤消在事务执行过程中所做的全部更改。

事务日志

一种按进行更改的顺序存储对数据库所做全部更改的文件。它会提高性能并支持在数据库文件损坏时恢复数据。

事务日志镜像

同时维护的事务日志文件的完全相同副本（可选）。每当数据库更改写入事务日志文件时，也会同时写入事务日志镜像文件。

镜像文件应该与事务日志保留在不同的设备上，这样在任意设备出现故障时，日志的其它副本会确保数据可以安全地恢复。

另请参见： “事务日志”一节第 789 页

事务完整性

MobiLink 中对整个同步系统事务的有保证维护。要么同步整个事务，要么不对事务的任何部分进行同步。

生成的连接条件

一种自动生成的对连接结果的限制。有两种类型：关键和自然。指定 KEY JOIN 或指定关键字 JOIN 但不使用关键字 CROSS、NATURAL 或 ON 时，会生成关键连接。对于关键连接，所生成的连接条件取决于表之间的外键关系。指定 NATURAL JOIN 时会生成自然连接；所生成的连接条件基于两个表中的公用列名。

另请参见：

- “连接”一节第 785 页
- “连接条件”一节第 785 页

受保护的功能

数据库服务器启动时由 -sf 选项指定的功能，该数据库服务器上运行的任何数据库都无法使用该功能。

授权选项

一种权限级别，它允许用户向其他用户授予权限。

数据操作语言 (DML)

用于操作数据库中数据的 SQL 语句子集。DML 语句可以检索、插入、更新和删除数据库中的数据。

数据定义语言 (DDL)

用于定义数据库中数据结构的 SQL 语句子集。DDL 语句可以创建、修改和删除数据库对象（如表和用户）。

数据类型

数据的格式，如 CHAR 或 NUMERIC。在 ANSI SQL 标准中，数据类型也可以包括对大小、字符集和归类的限制。

另请参见： “域”一节第 799 页

数据立方体

一种多维结果集，每一维都以不同的方式对相同的结果进行分组和排序。数据立方体提供了有关数据的综合性信息，如果不使用数据立方体，要获得同样的信息就必须进行自连接查询和相关子查询。数据立方体是 OLAP 功能的一部分。

数据库

通过主键和外键关联的表的集合。表包含数据库中的信息。表和键一起定义数据库的结构。数据库管理系统会访问此信息。

另请参见：

- “外键”一节第 794 页
- “主键”一节第 802 页
- “数据库管理系统 (DBMS)”一节第 791 页
- “关系数据库管理系统 (RDBMS)”一节第 780 页

数据库对象

包含或接收信息的数据库组件。表、索引、视图、过程和触发器便是数据库对象。

数据库服务器

对所有针对数据库信息的访问进行管理的计算机程序。SQL Anywhere 提供了两种类型的服务器：网络服务器和个人服务器。

数据库管理系统 (DBMS)

用于创建和使用数据库的程序的集合。

另请参见： “关系数据库管理系统 (RDBMS)”一节第 780 页

数据库管理员 (DBA)

具有维护数据库所需权限的用户。DBA 通常负责对数据库模式的所有更改以及管理用户和组。数据库管理员角色自动内置于数据库中，其用户 ID 为 DBA，口令是 sql。

数据库连接

客户端应用程序与数据库之间的通信渠道。必须具有有效的用户 ID 和口令才能建立连接。为用户 ID 授予的特权决定了在连接过程中可以执行的操作。

数据库名称

服务器装载数据库时为数据库指定的名称。缺省数据库名是初始数据库文件的文件名（不含扩展名）。

另请参见： “数据库文件”一节第 792 页

数据库所有者 (dbo)

一种特殊的用户，他拥有不归 SYS 所有的系统对象。

另请参见：

- “数据库管理员 (DBA)”一节第 791 页
- “SYS”一节第 793 页

数据库文件

数据库保存在一个或多个数据库文件中。其中一个是初始文件，后面的文件称作 dbspace。每个表（包括其索引）都必须包含在单个数据库文件中。

另请参见： “dbspace”一节第 778 页

死锁

一组事务会进入的一种特殊状态，在该状态下这些事务都不能继续执行。

SQL

用于与关系数据库进行通信的语言。ANSI 定义了 SQL 的标准，其最新标准是 SQL-2003。SQL 的非官方全称是结构化查询语言。

SQL Anywhere

SQL Anywhere 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。SQL Anywhere 也是包含 SQL Anywhere RDBMS、UltraLite RDBMS、MobiLink 同步软件和其它组件的软件包的名称。

SQL Remote

一种基于消息的数据复制技术，用于在统一数据库与远程数据库之间进行双向复制。统一数据库和远程数据库必须是 SQL Anywhere。

SQL 语句

包含用于将指令传递给 DBMS 的 SQL 关键字的字符串。

另请参见：

- “模式”一节第 787 页
- “SQL”一节第 792 页
- “数据库管理系统 (DBMS)”一节第 791 页

锁定

一种在同时执行多个事务的过程中保护数据完整性的并发控制机制。SQL Anywhere 会自动应用锁以防止两个连接同时更改同一数据，并防止其它连接读取正接受更改的数据。

您可以通过设置隔离级别来控制锁定。

另请参见：

- “隔离级别”一节第 780 页
- “并发”一节第 775 页
- “完整性”一节第 795 页

索引

一组已排序的、与基表中的一个或多个列关联的键和指针。在表中一个或多个列上设置索引可以提高性能。

Sybase Central

一种数据库管理工具，通过图形用户界面提供 SQL Anywhere 数据库设置、属性和实用程序。Sybase Central 也可用于管理其它 Sybase 产品，其中包括 MobiLink。

SYS

一种拥有大多数系统对象的特殊用户。无法以 SYS 身份登录。

统一数据库

在分布式数据库环境中，是指用于存储数据主副本的数据库。出现冲突或差异时，将把统一数据库视为具有数据的主副本。

另请参见：

- “同步”一节第 793 页
- “复制”一节第 779 页

通信流

MobiLink 中 MobiLink 客户端与 MobiLink 服务器之间进行通信时所使用的网络协议。

通告程序

一种由 MobiLink 服务器启动的同步使用的程序。通告程序集成在 MobiLink 服务器中。它们会检查统一数据库是否有推式请求，并发送推式通知。

另请参见：

- “服务器启动的同步”一节第 779 页
- “监听器”一节第 783 页

同步

利用 MobiLink 技术在数据库之间复制数据的过程。

在 SQL Remote 中，同步专指以初始数据集初始化远程数据库的过程。

另请参见：

- “MobiLink”一节第 786 页
- “SQL Remote”一节第 792 页

推式请求

在 MobiLink 服务器启动的同步中，通告程序通过检查它来确定推式通知是否需要发送到设备的结果集中的一行值。

另请参见： “服务器启动的同步”一节第 779 页

推式通知

QAnywhere 中一种从服务器传送到 QAnywhere 客户端的特殊消息，用于提示客户端启动消息传输。在 MobiLink 服务器启动的同步中，从通告程序传送到包含推式请求数据和内部信息的设备的特殊消息。

另请参见：

- “QAnywhere”一节第 788 页
- “服务器启动的同步”一节第 779 页

UltraLite

一种针对小型设备、移动设备和嵌入式设备进行了优化的数据库。所面向的平台包括手机、传呼机和个人记事本。

UltraLite 运行时

一种过程中关系数据库管理系统，其中包括一个内置 MobiLink 同步客户端。每个 UltraLite 编程接口使用的库以及 UltraLite 引擎中都包括 UltraLite 运行时。

外表

包含外键的表。

另请参见： “外键”一节第 794 页

外部登录

与远程服务器通信时使用的替代登录名和口令。缺省情况下，SQL Anywhere 每次代表其客户端连接到远程服务器时都会使用这些客户端的名称和口令。但是，您可以通过创建外部登录来替换这一缺省设置。外部登录是指与远程服务器通信时使用的替代登录名和口令。

外键

一个表中复制另一个表中主键值的一个或多个列。外键建立表间的关系。

另请参见：

- “主键”一节第 802 页
- “外表”一节第 794 页

外键约束

对单个列或一组列的限制，指定表中的数据与某个其它表中数据的关系。对列集施加外键约束可使这些列成为外键。

另请参见：

- “约束”一节第 801 页
- “检查约束”一节第 783 页
- “主键约束”一节第 802 页
- “唯一约束”一节第 796 页

外连接

一种保留表中所有行的连接。SQL Anywhere 支持左、右和完全外连接。左外连接保留表中位于连接运算符左侧的行，当右表中的行不满足连接条件时，它将返回空值。完全外连接保留两个表中的所有行。

另请参见：

- “连接”一节第 785 页
- “内连接”一节第 787 页

完全备份

对整个数据库和事务日志（可选）的备份。完全备份包含数据库中的所有信息，因此可以在系统或介质出现故障时提供保护。

另请参见： “增量备份”一节第 801 页

完整性

遵守完整性规则的情况，完整性规则确保数据正确并准确，而且数据库的关系结构保持不变。

另请参见： “参照完整性”一节第 776 页

网关

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关如何发送用于服务器启动同步的消息的信息。

另请参见： “服务器启动的同步”一节第 779 页

网络服务器

从共享公共网络的计算机接受连接的数据库服务器。

另请参见： “个人服务器”一节第 780 页

网络协议

通信类型，如 TCP/IP 或 HTTP。

维护版本

维护版本是一套完整的软件，它升级已安装的具有相同主版本号的较早版本的软件（版本号格式是 *major.minor.patch.build*）。升级程序的发行说明中列出了错误修正软件和其它更改。

唯一约束

对某个列或一组列的限制，它要求所有非空值都各不相同。一个表可以有多个唯一约束。

另请参见：

- “外键约束”一节第 795 页
- “主键约束”一节第 802 页
- “约束”一节第 801 页

谓语

一种条件表达式，可以选择性地将其与逻辑运算符 AND 和 OR 组合在一起，以组成 WHERE 或 HAVING 子句中的条件集。在 SQL 中，求值结果为 UNKNOWN 的谓语将解释为 FALSE。

位数组

位数组是一种用于有效率地存储位序列的数组数据结构。位数组与字符串类似，不同的是其各个部分由 0（零）和 1（一）而不是字符组成。位数组通常用于保存一串布尔值。

Windows

Microsoft Windows 操作系统系列，如 Windows Vista、Windows XP 和 Windows 200x。

Windows CE

请参见“Windows Mobile”一节第 796 页。

Windows Mobile

Microsoft 为移动设备制造的操作系统系列。

文件定义数据库

MobiLink 中一种用于创建下载文件的 SQL Anywhere 数据库。

另请参见： “基于文件的下载”一节第 783 页

物理索引

索引存储在磁盘上的实际索引结构。

系统表

一种表，由 SYS 或 dbo 拥有，用于保存元数据。系统表也称作数据字典表，由数据库服务器创建并维护。

系统对象

由 SYS 或 dbo 拥有的数据库对象。

系统视图

存在于每一个数据库中的一种视图，它以易于理解的格式表示系统表中包含的信息。

下载

同步过程的一个阶段，在此阶段数据从统一数据库传送到远程数据库。

相关名

查询的 FROM 子句中使用的表或视图的名称—要么是表或视图的原始名称，要么是在 FROM 子句中定义的替代名称。

项目

在 MobiLink 或 SQL Remote 中，项目是表示整个表或表中行和列子集的数据库对象。项目在发布中组合在一起。

另请参见：

- “复制”一节第 779 页
- “发布”一节第 778 页

消息存储库

QAnywhere 中客户端和服务器设备上存储消息的数据库。

另请参见：

- “客户端消息存储库”一节第 784 页
- “服务器消息存储库”一节第 779 页

消息类型

SQL Remote 复制中指定远程用户与统一数据库发布者通信方式的数据库对象。一个统一数据库可能定义了几种消息类型，这样一来，不同的远程用户就可以使用不同的消息系统与统一数据库进行通信。

另请参见：

- “复制”一节第 779 页
- “统一数据库”一节第 793 页

消息日志

可存储来自数据库服务器或 MobiLink 服务器等应用程序的消息的日志。此类信息还可以出现在消息窗口中或记录到文件中。消息日志包括信息性消息、错误、警告以及来自 MESSAGE 语句的消息。

消息系统

SQL Remote 复制中用于在统一数据库与远程数据库之间交换消息的协议。SQL Anywhere 包括对以下消息系统的支持：FILE、FTP 和 SMTP。

另请参见：

- “复制”一节第 779 页
- “FILE”一节第 779 页

卸载

卸载数据库时会将数据库的结构和/或数据导出到文本文件（如果是结构，则导出到 SQL 命令文件中；如果是数据，则导出到 ASCII 逗号分隔文件中）。使用卸载实用程序来卸载数据库。

此外，您也可以使用 UNLOAD 语句卸载数据的选定部分。

性能统计

反映数据库系统性能的值。例如，CURRREAD 统计表示数据库服务器已发出但尚未完成的文件读取次数。

业务规则

基于实际要求的准则。通常，业务规则通过检查约束、用户定义数据类型以及事务的正确使用来实现。

另请参见：

- “约束”一节第 801 页
- “用户定义数据类型”一节第 799 页

引用对象

一种对象（如视图），其定义直接引用数据库中的另一个对象（如表）。

另请参见： “外键”一节第 794 页

用户定义数据类型

请参见“域”一节第 799 页。

游标

指向结果集的已命名链接，用于通过编程接口访问和更新行。在 SQL Anywhere 中，游标支持在查询结果中进行向前和向后移动。游标由两部分组成：游标结果集（通常由 SELECT 语句定义）和游标位置。

另请参见：

- “游标结果集”一节第 799 页
- “游标位置”一节第 799 页

游标结果集

与游标关联的查询所得到的行集。

另请参见：

- “游标”一节第 799 页
- “游标位置”一节第 799 页

游标位置

指向游标结果集中一个行的指针。

另请参见：

- “游标”一节第 799 页
- “游标结果集”一节第 799 页

语句级触发器

在整个触发语句完成后执行的触发器。

另请参见：

- “触发器”一节第 777 页
- “行级触发器”一节第 781 页

域

内置数据类型的别名，其中包括适用的精度值和小数位值，还可以选择是否包括 DEFAULT 值和 CHECK 条件。SQL Anywhere 中预定义了一些域，如货币数据类型。也称作用户定义数据类型。

另请参见： “数据类型”一节第 790 页

预订

MobiLink 同步中发布与 MobiLink 用户之间的客户端数据库中的一个链接，它使发布所描述的数据能够得到同步。

SQL Remote 复制中发布与远程用户之间的一种链接，它使用户能够与统一数据库交换该发布上的更新。

另请参见：

- “发布”一节第 778 页
- “MobiLink 用户”一节第 787 页

元数据

数据的数据。元数据描述其它数据的性质和内容。

另请参见： “模式”一节第 787 页

原子事务

保证成功完成或保证根本不予完成的事务。如果错误使原子事务的一部分无法完成，则将回退事务以防止数据库处于不一致的状态。

REMOTE DBA 特权

在 SQL Remote 中，消息代理 (dbremote) 所需的权限级别。MobiLink 中 SQL Anywhere 同步客户端 (dbmlsync) 所需的权限级别。当消息代理或同步客户端作为具有该权限的用户建立连接时，它将具有完全的 DBA 访问权。如果不是通过消息代理或同步客户端进行连接，则该用户 ID 将不具有附加权限。

另请参见： “DBA 权限”一节第 778 页

远程 ID

SQL Anywhere 和 UltraLite 数据库中一种由 MobiLink 使用的唯一标识符。远程 ID 初始情况下设置为 NULL，在数据库第一次同步期间将设置为 GUID。

远程数据库

MobiLink 或 SQL Remote 中一种与统一数据库交换数据的数据库。远程数据库可以共享统一数据库中的全部或部分数据。

另请参见：

- “同步”一节第 793 页
- “统一数据库”一节第 793 页

约束

对特定数据库对象（如表或列）中所包含值的限制。例如，列可以具有唯一性约束，该约束要求该列中的所有值互不相同。表可以具有外键约束，该约束指定该表中的信息与某个其它表中数据的关系。

另请参见：

- “检查约束”一节第 783 页
- “外键约束”一节第 795 页
- “主键约束”一节第 802 页
- “唯一约束”一节第 796 页

运营公司

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关供服务器启动的同步使用的公共运营公司的信息。

另请参见： “服务器启动的同步”一节第 779 页

增量备份

仅包含事务日志的备份，通常在两次完全备份之间使用。

另请参见： “事务日志”一节第 789 页

争用

为获取资源而竞争的行为。例如，就数据库而言，如果有两个或更多个用户试图编辑数据库的同一行，就会为获得编辑该行的权利而发生争用。

正则表达式

正则表达式是字符、通配符和运算符的序列，用于定义某种模式以在字符串内进行搜索。

直方图

直方图是列统计信息最重要的组成部分，是一种表示数据分布的方式。SQL Anywhere 维护直方图以为优化程序提供有关列值分布情况的统计信息。

直接行处理

MobiLink 中一种用于将表数据同步到 MobiLink 支持的统一数据库以外的数据源的方法。使用直接行处理时，上载和下载都可以实现。

另请参见：

- “统一数据库”一节第 793 页
- “基于 SQL 的同步”一节第 783 页

主表

包含外键关系中的主键的表。

主键

其值唯一标识表中各行的一个列或多个列。

另请参见： “外键”一节第 794 页

主键约束

一种对主键列的唯一性约束。一个表只能有一个主键约束。

另请参见：

- “约束”一节第 801 页
- “检查约束”一节第 783 页
- “外键约束”一节第 795 页
- “唯一约束”一节第 796 页
- “完整性”一节第 795 页

子查询

嵌套在 SELECT、INSERT、UPDATE 或 DELETE 语句或者其它子查询中的 SELECT 语句。

有两种类型的子查询：相关子查询和嵌套子查询。

字符串

字符串是以单引号围起的字符序列。

字符集

字符集是一组符号，包括字母、数字、空格和其它符号。字符集的一个例子是 ISO-8859-1，又称作 Latin1。

另请参见：

- “代码页”一节第 777 页
- “编码”一节第 775 页
- “归类”一节第 781 页

索引

其它

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

Y

Z

