新增功能公告

SAP Open Server™ 和 SDK for SAP ASE 16.0

Windows、Linux 和 UNIX

文档 ID： DC00571-01-1570100-01
最后修订日期： 2014 年 3 月

目录

产品平台和兼容性

以下平台支持 SAP® Open Server™ 和 SDK for SAP® ASE。

- HP-UX Itanium 32 位
- HP-UX Itanium 64 位
- IBM AIX 32 位
- IBM AIX 64 位
- Linux x86 32 位
- Linux x86-64 64 位
- Linux on POWER 32 位
- Linux on POWER 64 位
- Microsoft Windows x86 32 位
- Microsoft Windows x86-64 64 位
- Solaris SPARC 32 位
- Solaris SPARC 64 位
- Solaris x86 32 位
- Solaris x86-64 64 位

注意： 并不是所有 SAP Open Server 和 SDK for SAP ASE 组件都可以在上面列出的平台中使用。 有关每个平台中可用组件的完整列表，请参见“产品组件”。

SAP Open Server 和 SDK for SAP ASE 平台兼容性表

该表列出了构建和测试 SAP Open Server 和 SDK for SAP ASE 产品时所使用的平台、编译器和第三方产品。

平台	操作系统级别	C 和 C++ 编译器	COBOL 编译器	Kerberos 版本	轻量目录访问协议 (LDAP)	安全套接字层 (SSL)	Perl 版本	PHP 版本	Python 版本
HP-UX Itanium 32 位	HP 11.31	HP ANSI C A. 06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不可用	不可用

平台	操作系统级别	C 和 C++ 编译器	COBOL 编译器	Kerberos 版本	轻量目录访问协议 (LDAP)	安全套接字层 (SSL)	Perl 版本	PHP 版本	Python 版本
HP-UX Itanium 64 位	HP 11.31	HP ANSI C A. 06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
IBM AIX 32 位	AIX 6.1	XL C 10.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.1	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不可用	不可用
IBM AIX 64 位	AIX 6.1	XL C 10.1	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Linux x86 32 位	Red Hat En-terprise Linux 5.3	gcc 4.1.2 20060404 ker-nel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	不可用	不可用

平台	操作系统级别	C 和 C++ 编译器	COBOL 编译器	Kerberos 版本	轻量目录访问协议 (LDAP)	安全套接字层 (SSL)	Perl 版本	PHP 版本	Python 版本
Linux x86-64 64 位	Red Hat Enterprise Linux 5.3 (Nahant Update 4)	gcc 4.1.2 20060404 kernel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Linux on POWER 32 位	Red Hat Enterprise Linux 5.3	XL C 10.1	未计划	MIT 1.4.1	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不可用	不可用
Linux on POWER 64 位	Red Hat Enterprise Linux 5.3	XL C 10.1	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Microsoft Windows x86 32 位	Windows 2008 R2 Service Pack 1	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 4.0、MIT 2.6.4	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	不可用	不可用

平台	操作系统级别	C 和 C++ 编译器	COBOL 编译器	Kerberos 版本	轻量目录访问协议 (LDAP)	安全套接字层 (SSL)	Perl 版本	PHP 版本	Python 版本
Microsoft Windows x86-64 64 位	Windows 2008 R2 Service Pack 1	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 2.1	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	Active Perl 5.14.1 (DBI 1.616)	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Solaris SPARC 32 位	Solaris 10	Solaris Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.2	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不可用	不可用
Solaris SPARC 64 位	Solaris 10: 修补程序级别 144488-17 或更高, `SUNWlibC`: 修补程序级别 119963-24 或更高	Solaris Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.2	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Solaris x86 32 位	Solaris 10	Solaris Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不可用	不可用

平台	操作系统级别	C 和 C++ 编译器	COBOL 编译器	Kerberos 版本	轻量目录访问协议 (LDAP)	安全套接字层 (SSL)	Perl 版本	PHP 版本	Python 版本
Solaris x86 64 位	Solaris 10	Solaris Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.31（含有 OpenSSL 1.0.1b）	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不可用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
图例：不可用 = 脚本在该平台上不可用或者无法与 SDK for SAP ASE 一起工作。									

注意： 有关最新的 SAP Open Server 和 SDK for SAP ASE 证书支持的信息，请参见 SAP® Sybase® 平台证书页面 http://certification.sybase.com/ucr/search.do

Microsoft 已结束对 Visual Studio 2005 的主流支持。尽管 SDK for SAP ASE 当前仍支持 Visual Studio Compiler 2005 及更高版本，但 SAP 建议您尽快升级到 Visual Studio 2010。

Solaris SPARC 64 位修补程序级别

对于 Solaris SPARC 64 位平台，Solaris 10 操作系统内核的修补程序级别必须为 144488-17 或更高（2011 年 6 月 30 日及以后的修补程序包）。

此外，还必须将 119963-24 或更高级别的修补程序应用于 `SUNWlibC` 软件包。

FIPS 兼容平台支持

SAP Open Server 和所有 SDK for SAP ASE 组件（SAP® jConnect™ for JDBC (SAP jConnect) 除外）均符合 FIPS 标准并可在这些平台上使用。

平台	FIPS 合规性
HP-UX Itanium 32 位	15.7 SP121
HP-UX Itanium 64 位	15.7 SP120
IBM AIX 32 位	无
IBM AIX 64 位	无
Linux x86 32 位	15.7 SP121

平台	FIPS 合规性
Linux x86-64 64 位	15.7 SP120
Linux on POWER 32 位	15.7 SP120
Linux on POWER 64 位	无
Microsoft Windows x86 32 位	15.7 SP120
Microsoft Windows x86-64 64 位	15.7 SP121
Solaris SPARC 32 位	15.7 SP120
Solaris SPARC 64 位	15.7 SP121
Solaris x86 32 位	15.7 SP121
Solaris x86 64 位	15.7 SP121

SAP jConnect for JDBC FIPS 合规性

从 16.0 版开始，SAP jConnect 将在产品中附带 FIPS 140-2 认证的 Java Cryptography Extension (JCE) 提供程序。 该 JCE 提供程序将在缺省情况下使用并提供符合 FIPS 140-2 标准的加密服务。

产品组件

这里将介绍 SAP Open Server 和 SDK for SAP ASE 的产品组件。

SAP Open Server 16.0 和 SDK for SAP ASE 16.0 还支持将 Perl、PHP 和 Python 脚本化语言用于 SAP® Adaptive Server® Enterprise。

SAP Open Server

SAP Open Server 是一组 API 和支持工具，可用于创建自定义服务器以响应通过 SAP® Open Client™ 或 SAP jConnect 例程提交的客户端请求。

表 1. SAP Open Server 组件和支持的平台

Open Server 组件	平台
SAP Open Server Server-Library	所有平台
SAP Open Server Client-Library	所有平台
语言模块	所有平台

SDK for SAP ASE

SDK for SAP ASE 是一组用于开发客户端应用程序的库和实用程序。

表 2. SDK for SAP ASE 组件和支持的平台

SDK for SAP ASE 组件	平台
SAP Open Client Client-Library	平台
SAP® Open Client DB-Library™	所有平台
SAP® Embedded SQL™/C (ESQL/C)	所有平台

SDK for SAP ASE 组件	平台
Embedded SQL/COBOL (ESQL/COBOL)	• HP HP-UX Itanium 32 位 • HP HP-UX Itanium 64 位 • IBM AIX 64 位 • Linux x86 32 位 • Linux x86-64 64 位 • Linux on POWER 32 位 • Linux on POWER 64 位 • Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位 • Solaris SPARC 32 位 • Solaris SPARC 64 位 • Solaris x86 32 位 • Solaris x86-64 64 位
扩展体系结构 (XA)	• HP HP-UX Itanium 32 位 • HP HP-UX Itanium 64 位 • IBM AIX 32 位 • IBM AIX 64 位 • Linux x86-64 64 位 • Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位 • Solaris SPARC 32 位 • Solaris SPARC 64 位 • Solaris x86 32 位 • Solaris x86-64 64 位
SAP jConnect	所有平台
SAP Adaptive Server Enterprise ODBC 驱动程序	• HP HP-UX Itanium 64 位 • IBM AIX 64 位 • Linux on POWER 64 位 • Linux x86 32 位 • Linux x86-64 64 位 • Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位 • Solaris SPARC 64 位 • Solaris x86-64 64 位
SAP Adaptive Server Enterprise ADO.NET 数据提供程序	• Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位

SDK for SAP ASE 组件	平台
语言模块	所有平台
用于 Python 的 SAP Adaptive Server Enterprise 扩展模块	• HP-UX Itanium 64 位 • IBM AIX 64 位 • Linux x86-64 64 位 • Linux on POWER 64 位 • Microsoft Windows x86-64 64 位 • Solaris SPARC 64 位 • Solaris x86-64 64 位
用于 PHP 的 SAP Adaptive Server Enterprise 扩展模块	• HP-UX Itanium 64 位 • IBM AIX 64 位 • Linux x86-64 64 位 • Linux on POWER 64 位 • Microsoft Windows x86-64 64 位 • Solaris SPARC 64 位 • Solaris x86-64 64 位
用于 Perl 的 SAP Adaptive Server Enterprise 数据库驱动程序	• HP-UX Itanium 32 位 • IBM AIX 32 位 • Linux x86-64 64 位 • Linux on POWER 32 位 • Microsoft Windows x86-64 64 位 • Solaris SPARC 32 位 • Solaris x86 32 位
dbisql 工具	• HP HP-UX Itanium 64 位 • IBM AIX 64 位 • Linux on POWER 64 位 • Linux x86-64 64 位 • Solaris SPARC 64 位 • Solaris x86-64 64 位 • Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位
用于 DB-Library 的 Kerberos Authentication	• Linux x86 32 位 • Microsoft Windows x86 32 位 • Solaris SPARC 32 位 • Solaris SPARC 64 位

16.0 版的新增功能

16.0 版为 SAP Open Client 16.0、SAP Open Server 16.0、SDK for SAP ASE 16.0 和用于 PHP 16.0 的 SAP Adaptive Server Enterprise 扩展模块引入了更新的功能，同时停用了一些支持并进行了换代变更。

SAP Open Client 16.0 和 SAP Open Server 16.0 功能

SAP Open Client 和 SAP Open Server 为 `ocs.cfg` 文件引入了一个新属性，供用户应用配置设置。

命名应用程序的 NAMED_APP_DEFAULT 部分

用户可通过 `NAMED_APP_DEFAULT` 部分将一组配置设置应用到没有在 `ocs.cfg` 文件中显示的所有命名应用程序。

可以在 `NAMED_APP_DEFAULT` 部分设置 `ocs.cfg` 运行时配置文件中不带特定部分的命名应用程序属性。例如，要为`ocs.cfg`中不带特定部分的命名应用程序开启 FIPS 模式，可指定以下内容：

```
[NAMED_APP_DEFAULT]
CS_PROP_FIPSMODE = CS_TRUE ;
```

SDK DB-Library Kerberos Authentication Option 变更

从 16.0 版开始，DB-Library Kerberos Authentication Option 不再是独立的可购选项，而是作为组件随 SDK for SAP ASE 一起提供。

有关 SDK for SAP ASE 组件的详细信息，请参见“产品组件”>“SDK for SAP ASE”。

要安装并启用 SDK DB-Library Kerberos Authentication 组件，请参见所用平台的《SDK for SAP ASE 和 SAP Open Server 16.0 安装指南》(SDK for SAP ASE and SAP Open Server 16.0 Installation Guide)。

有关已知问题，请参见所用平台的 SDK for SAP ASE 和 SAP Open Server 16.0 版发行公告。

SDK for SAP ASE 16.0 中针对 Adaptive Server Enterprise 驱动程序和提供程序的功能

16.0 版为 SAP jConnect 和 SAP Adaptive Server ADO.NET 数据提供程序引入了更新的功能。

SAP jConnect for JDBC 符合 FIPS 140-2 标准

从 16.0 版开始，SAP jConnect 将在产品中附带 FIPS 140-2 认证的 Java Cryptography Extension (JCE) 提供程序。 此 JCE 提供程序将缺省用于口令加密和 SSL 连接。

如果设置 `JCE_PROVIDER_CLASS` 属性，则将使用通过该属性指定的 JCE 提供程序。 如果要求符合 FIPS 140-2 标准，请确保指定的 JCE 提供程序经过 FIPS 140-2 认证。

要强制使用符合 FIPS 140-2 标准的加密，请将 `ENABLE_FIPS` 连接属性设置为 true。

适用于 16.0 版 SAP ASE ADO.NET 的 .NET Framework 组件

16.0 版 SAP ASE ADO.NET 支持 3.5 和 4.0 版的 .NET Framework。

表 3. SDK for SAP ASE 附带的 3.5 和 4.0 版 .NET Framework 组件

.NET Framework 版本	组件
3.5	• `Sybase.AdoNet35.AseClient.dll`（ADO.NET 提供程序） • `Sybase.VSIntegration35.ASE.dll`（DDEX 提供程序） • `Sybase.AdoNet35.AseDestination.dll`（SSIS 组件） • `Sybase.AdoNet35.AseReportingServices.dll`（SSRS 组件） • `Sybase.AdoNet35.EnterpriseLibrary.dll`（Enterprise Library 组件） • `AseGacUtility35.exe` • `AseRegistar35.exe`

.NET Framework 版本	组件
4.0	• Sybase.AdoNet4.AseClient.dll（ADO.NET 提供程序） • Sybase.VSIntegration4.ASE.dll（DDEX 提供程序） • Sybase.AdoNet4.AseDestination.dll（SSIS 组件） • Sybase.AdoNet4.AseReportingServices.dll（SSRS 组件） • Sybase.AdoNet4.EnterpriseLibrary.dll（Enterprise Library 组件） • AseGacUtility4.exe • AdoNetRegistrar4.exe

另请参见

- 废弃的功能 （第 14 页）

用于 PHP 的 SAP Adaptive Server Enterprise 扩展模块

用于 PHP 的 SAP ASE 扩展模块得到了增强，现可支持 SAP ASE 结果集数据类型。

PHP 驱动程序的扩展数据类型支持

从 16.0 版开始，用于 PHP 的 SAP ASE 驱动程序支持语言查询结果集中的所有 SAP ASE 数据类型。

同样从 16.0 版开始，用于 PHP 的 SAP ASE 驱动程序还支持远程过程调用参数的扩展数据类型。 已对 `sybase_rpc_bind_param_ex()` API 进行更新，现接受这些数据类型。

用于 PHP 的 SAP ASE 驱动程序使用 PHP datetime ISO 8601 分析函数，该函数可通过 PHP 驱动程序扩展获得。 在某些平台上，您可能需要显式指示 PHP 构建过程才能导出日期/时间函数。 例如，可在以下平台中使用如下设置：

- Windows - 向 `ext\date\php_date.def` 文件添加 *PHP_DLL_DEF_SOURCES* 变量的引用。 Makefile 变量是对 `ext\date\php_date.def` 的引用，其中包含 PHP 日期核心扩展函数的 EXPORTS。
- Linux - 在使用 gcc 编译器构建 PHP 环境时，请确保配置脚本没有在 CFLAGS 中设置“--fvisibility=hidden”。
- AIX - 在使用 xlc 编译器时，请确保 CFLAGS 中包含“-bexpall”，然后再编译 PHP 环境。

有关受支持的数据类型的详细信息，请参见《用于 PHP 的 Adaptive Server Enterprise 扩展模块程序员指南》。

Microsoft Windows x64 中通过 VC9/VS2008 编译 PHP 驱动程序

从 16.0 版开始，用于 PHP 的 SAP ASE 驱动程序在 Microsoft Windows x64 平台上通过 Microsoft Visual C++ 9/Visual Studio 2008 进行编译。

因此，PHP 驱动程序只能通过用 VC9/VS2008 编译的 64 位 PHP 5.3.6 版运行时进行装载。

注意： Visual Studio 2008 附带 Microsoft Visual C++ 9 编译器。

SDK for ASE SDK 16.0 的 dbisql 工具支持

SDK for SAP ASE 16.0 版本中增加了用于启动 Interactive SQL 的 **dbisql**。

请参见《Adaptive Server Enterprise 15.7 SP100 实用程序指南》中的 第 7 章：“在图形模式下使用 Interactive SQL”。

废弃的功能

已从 16.0 版的 SAP Open Server 和 SDK for SAP ASE 中删除对若干驱动程序的支持。

从 16.0 版开始，SDK for SAP ASE 和 SAP Open Server 不再支持以下内容：

- .NET 2.0 和 3.0 Framework
- OLE DB 提供程序

从 16.0 版的 SDK for SAP ASE 中删除的目录和组件如下：

- `DataAccess\ADONET\dll\Sybase.AdoNet2.AseClient.dll`
- `DataAccess\ADONET\dll\Sybase.AdoNet2.AseReportingServices.dll`
- `DataAccess\ADONET\dll\Sybase.EnterpriseLibrary.AseClient.dll`
- `DataAccess\ADONET\dll\Sybase.VSIntegration.ASE.dll`
- `DataAccess\ADONET\dll\AdoNetRegistrar.exe`
- `DataAccess\ADONET\dll\AseGacUtility.exe`
- `DataAccess\ADONET\dll\Microsoft.VC80.ATL`
- `DataAccess\ADONET\dll\Microsoft.VC80.CRT`
- `DataAccess\ADONET\dll\Microsoft.VC80.MFC`
- `DataAccess\ADONET\dll\Microsoft.VC80.MFCLOC`
- `DataAccess\ADONET\dll\Microsoft.VC80.OpenMP`
- `DataAccess64\ADONET\dll\Sybase.AdoNet2.AseClient.dll`
- `DataAccess64\ADONET\dll\Sybase.AdoNet2.AseDestination.dll`

- DataAccess64\ADONET\dll\Sybase.AdoNet2.AseReportingServices.dll
- DataAccess64\ADONET\dll\Sybase.EnterpriseLibrary.AseClient.dll
- DataAccess64\ADONET\dll\Sybase.VSIntegration.ASE.dll
- DataAccess64\ADONET\dll\AdoNetRegistrar.exe
- DataAccess64\ADONET\dll\AseGacUtility.exe
- DataAccess64\ADONET\dll\Microsoft.VC80.ATL
- DataAccess64\ADONET\dll\Microsoft.VC80.CRT
- DataAccess64\ADONET\dll\Microsoft.VC80.MFC
- DataAccess64\ADONET\dll\Microsoft.VC80.MFCLOC
- DataAccess64\ADONET\dll\Microsoft.VC80.OpenMP

- DataAccess\OLEDB\dll
- DataAccess\OLEDB\dll\locales
- DataAccess\OLEDB\dll\Microsoft.VC80.ATL
- DataAccess\OLEDB\dll\Microsoft.VC80.CRT
- DataAccess\OLEDB\dll\Microsoft.VC80.MFC
- DataAccess\OLEDB\dll\Microsoft.VC80.MFCLOC
- DataAccess\OLEDB\dll\Microsoft.VC80.OpenMP
- DataAccess\OLEDB\dll\sybdrvoledb.dll
- DataAccess\OLEDB\samples
- DataAccess\OLEDB\sp
- DataAccess\bin\sybdrvadm.exe
- DataAccess\bin\sybdrvadm.ico
- DataAccess\bin\dsnmigrate.exe

- DataAccess64\OLEDB
- DataAccess64\OLEDB\dll\locales
- DataAccess64\OLEDB\dll\Microsoft.VC80.ATL
- DataAccess64\OLEDB\dll\Microsoft.VC80.CRT
- DataAccess64\OLEDB\dll\Microsoft.VC80.MFC
- DataAccess64\OLEDB\dll\Microsoft.VC80.MFCLOC
- DataAccess64\OLEDB\dll\Microsoft.VC80.OpenMP
- DataAccess64\OLEDB\dll\sybdrvoledb64.dll
- DataAccess64\OLEDB\samples
- DataAccess64\OLEDB\sp
- DataAccess64\bin\sybdrvadm.exe
- DataAccess64\bin\sybdrvadm.ico
- DataAccess64\bin\dsnmigrate.exe

SP121 的新增功能

SP121 增加了其它符合 OpenSSL FIPS 标准的平台。

- HP-UX Itanium 32 位
- Linux x86 32 位
- Microsoft Windows x86-64 64 位
- Solaris SPARC 64 位
- Solaris x86 32 位
- Solaris x86 64 位

另请参见

- FIPS 兼容平台支持 （第 5 页）

SP120 的新增功能

SP120 中替换了 Certicom 加密服务，同时为 Open Client 15.7、Open Server 15.7 和用于 PHP 15.7 的 Adaptive Server Enterprise 扩展模块引入了新的功能和属性并更新了一些功能和属性。

Certicom 的替代软件

SAP® Sybase 产品不再支持为敏感信息的存储和传输提供保护的加密服务软件 Certicom。 这些服务已由备用提供程序替代。

现在，Open Server 和所有 SDK 组件（jConnect for JDBC 除外）均支持 OpenSSL。

现在，jConnect for JDBC 使用 Java VM 中可支持请求算法的可用 JCE 提供程序。 例如，SUN JCE 提供程序通常与 Oracle Java VM 一起使用。 实际使用的提供程序取决于 Java VM 的安全性配置。

ODBC、OLE DB、ADO.NET、Open Client 和 Open Server 中的 OpenSSL

现在，Open Server 和所有 SDK 组件（jConnect for JDBC 除外）均支持 OpenSSL。

引入此更改后，将不再支持下列证书实用程序：

- **certreq**
- **certauth**
- **certpk12**

作为替代，Open Server 和 SDK 增加了 **openssl** 实用程序，该实用程序位于：

- (UNIX) `$SYBASE/$SYBASE_OCS/bin`
- (Windows) `%SYBASE%\%SYBASE_OCS%\bin`

可使用 **openssl** 工具执行先前由 **certreq**、**certauth** 和 **certpk12** 执行的所有证书管理任务。

请参见 http://www.openssl.org/docs/apps/openssl.html。

启用 FIPS 合规性

从 15.7 SP120 开始，在缺省情况下，客户端库不启用严格的 FIPS 合规性，因此，应用程序应启用合规性。

FIPS 模式下的 OpenSSL 由 OpenSSL 安全设置严格控制。 因此，在客户端库上启用 FIPS 合规性前，请确保服务器 SSL 证书符合 FIPS 要求。 否则，在启用 FIPS 模式时，与服务器的连接将失败。

这还意味着在使用 OpenSSL 的情况下，曾与 Certicom FIPS 模块配合使用的某些证书可能不再适用。

针对服务器 SSL 证书的 FIPS 140-2 要求：

- MD5 算法不符合 FIPS 140-2 标准；必须将 MD5 替换为符合 FIPS 标准的算法。
- 私有密钥必须为 pkcs8 格式并且通过符合 FIPS 140-2 标准的 OpenSSL 算法加密。
- 如果对数字签名使用 RSA 加密算法，则 RSA 密钥大小必须至少为 1024 位。

有关详细信息，请参见《Adaptive Server Enterprise SP60 新增功能指南》。

在 ODBC、OLE DB 和 ADO.NET 驱动程序中启用 FIPS 合规性

为了能够在 OpenSSL 中启用或禁用 FIPS 模式，在 ODBC、OLE DB 和 ADO.NET 驱动程序中添加了新的连接属性 `EnableFIPS`。

- 要在 ODBC 中启用 FIPS 模式，则将 `EnableFIPS=1` 添加到连接字符串。
- 要在 OLE DB 中启用 FIPS 模式，则将 `EnableFIPS=true` 添加到提供程序字符串。
- 要在 ADO.NET 中启用 FIPS 模式，则将 `EnableFIPS=true` 添加到连接字符串。

在缺省情况下，`EnableFIPS` 属性处于禁用状态（设置为 false 或 0）。 每个客户端进程只能打开一种类型的连接。 如果某个连接必须是 FIPS，那么所有连接均必须使用 FIPS。

注意： Sybase 专有口令加密使用的算法不符合 FIPS 标准。 因此，当启用 FIPS 模式时，需确保服务器支持 RSA 口令加密模式。

在以下情况下，可能会出现设置安全上下文时出错：

与 FIPS 无关	SSL 连接的 `trusted.txt` 文件格式错误。 检查 `trusted.txt` 路径和文件。
启用 FIPS 模式	• 在 Microsoft Windows 上，驱动程序未获得其首选基址，因此 OpenSSL 核内指纹检查失败。 您可以使用 Microsoft Process Explorer 查看正在运行的进程的基址。 • OpenSSL 因未知原因导致指纹检查失败。

在 Microsoft Windows 上，驱动程序有一个首选基址，需要装载到内存中。 首选基址如下所示：

- ODBC： `0xF800000`
- OLE DB： `0xF500000`
- ADO.NET： `0xF200000`

在 Open Client 和 Open Server 上启用 FIPS 合规性

可以在 Open Client 和 Open Server 上启用 FIPS 140-2 合规性。

- 应用程序必须在 `ocs.cfg` 文件中将上下文属性 CS_PROP_FIPSMODE 设置为 CS_TRUE，或
- 将环境变量 SYBOCS_FIPS_MODE 设置为 1。

在 Microsoft Windows 上启用 FIPS 合规性时，要在 Open Client 的内存中装载的首选基址为 `0xFB00000`。 这是为了避免因 OpenSSL 核内指纹完整性检查而引起基址冲突。

注意： 如果首选基址不可用，会出现“FIPS 指纹检查失败”(FIPS fingerprint check failed) 错误，Open Client 初始化将失败。

在 Perl、Python 和 PHP 上启用 FIPS 合规性

可以在 Perl、Python 和 PHP 上启用 FIPS 140-2 合规性。

要在 Perl、Python 和 PHP 上启用 FIPS 合规性，请将 `FIPSMode` 连接属性设置为 true。

在 Microsoft Windows 上启用 FIPS 合规性时，要在内存中装载的首选基址为 `0xFB00000`。 这是为了避免因 OpenSSL 核内指纹完整性检查而引起基址冲突。

注意： 如果首选基址不可用，会出现“FIPS 指纹检查失败”(FIPS fingerprint check failed) 错误，初始化将失败。

FIPS 平台可用性

取代 Certicom 后，随 SP120 一起提供的符合 FIPS 140-2 标准的 OpenSSL 加密模块在以下平台上均可用。

- HP-UX Itanium 64 位
- Linux x86-64 64 位
- Linux on POWER 32 位
- Microsoft Windows x86 32 位
- Solaris SPARC 32 位

另请参见

- FIPS 兼容平台支持 （第 5 页）

jConnect for JDBC 使用的 JCE 提供程序

现在，jConnect 使用 Java VM 中可支持请求算法的可用 JCE 提供程序。

例如，SUN JCE 提供程序通常与 Oracle Java VM 一起使用。 实际使用的提供程序取决于 Java VM 的安全性配置。

jConnect 配置以使用特定 JCE 提供程序

对 jConnect 进行配置以使用特定 JCE 提供程序。
将 JCE_PROVIDER_CLASS 连接属性设置为提供程序的字符串类名称。

注意： 确保 JCE 提供程序 jar 文件位于 CLASSPATH 中。

在 jConnect for JDBC 中启用 FIPS 合规性

通常情况下，随 Java VM 一起提供的缺省 JCE 提供程序没有经过 FIPS 140-2 认证。要设置符合 FIPS 140-2 标准的连接，您需要访问 FIPS 140-2 认证的 JCE 提供程序。

访问 FIPS 140-2 认证的 JCE 提供程序后，您可以通过设置以下属性为 FIPS 140-2 合规性配置 jConnect：

1. 将 `ENABLE_FIPS` 连接属性布尔值设置为 TRUE。
2. 将 `JCE_PROVIDER_CLASS` 连接属性设置为 FIPS 140-2 认证的 JCE 提供程序的字符串类名称。

Open Client 15.7 和 Open Server 15.7 功能

Open Client 15.7 和 Open Server 15.7 可通过更新后的功能与新的连接属性来支持 Client-Library。

新 isql 参数可以提高性能

使用新的 `--fast` 命令行参数可以加速列类型仅为数字的（大型）数据集的检索。

`--fast` 参数包括 Sybase Adaptive Server® Enterprise 和 Open Client 支持的所有整数类型。

注意： `--fast` 参数不会影响 string、date、time 和 datetime 数据类型。

`--fast` 将更改列输出的格式，并且不与标准输出兼容。 仍保留标准 **isql** 输出格式。此外，如果使用 `--fast`，则不会发生换行，这意味着除非在 `--fast` 中加入 `-w` 标志，否则不会保留缺省列宽 80。

下面的示例将在所选输出的第 80 列换行：

```
isql -U sa -P --fast -w80
```

下面的示例则不会换行：

```
isql -U sa -P --fast
```

isql 和 bcp 的 --filemode 选项

（仅限 UNIX）`--filemode` 选项用于为通过 **isql** 生成的文件（输出文件）和通过 **bcp** 生成的文件（所有通过 **bcp** 生成的文件）设置文件权限。 `--filemode` 优先于这些文件的所有缺省权限设置。

通过 `--filemode <nnn>` 选项，用户可以为通过 **bcp** 和 **isql** 生成的一些文件指定比缺省设置更加宽松的权限设置。

- 对于 **isql**，使用 `--filemode <nnn>` 选项设置宽松的文件权限将影响通过 **isql** 生成的输出文件（-o 选项）和重定向输出（使用“go > filename”方法）。 例如，要创建一个 **isql** 输出文件，要求“user”必须可对该文件进行读取和写入，“group”只能对该文件进行读取，“other”不能对该文件进行读取和写入。用户可指定以下命令：

  ```
  isql -U sa -P secret -o myoutput --filemode 640
  ```
- 对于 **bcp**，使用 `--filemode <nnn>` 选项设置宽松的文件权限将影响所生成的以下 **bcp** 文件：
  - 数据文件（含 **bcp** 输出数据的文件）
  - 输出文件（使用 -o 选项）
  - 格式文件（使用 -f 选项）
  - 错误文件（使用 -e 选项）

有关所生成的所有数据和输出文件的缺省权限，请参见 ESD #4 的“适用于 Open Client 和 Open Server 文件的更为严格的权限（仅限 UNIX）”部分。

另请参见

- 适用于 Open Client 和 Open Server 文件（仅限 UNIX）的更为严格的权限 （第 93 页）

连接字符串属性中的新关键字

Client Library 具有 API 例程 **ct_connect_string()** 的新关键字。

ct_connect_string()

名称	说明	值
DataOrigin	启用数据源加戳。 数据源加戳可验证数据是由客户端还是服务器发送的。	布尔值。 缺省值为 false。
FIPSMode	**FIPSMode** 可以确定符合 FIPS 标准的 SSL 加密算法的使用。 另请参见上下文属性 **CS_PROP_FIPSMODE**。	布尔值。 缺省值为 false。
HAFailover	确定高可用性故障切换。	布尔值。 缺省值为 false。
PasswordEncryptionOnRetry	关键字 **PasswordEncryption** 可为某个连接同时启用 **CS_SEC_ENCRYPTED_PASSWORD** 和 **CS_SEC_EXTENDED_ENCRYPTED_PASSWORD**。 **PasswordEncryptionOnRetry** 可确保当客户端应用程序连接到较低版本的服务器时，如果服务器不支持新版本，该客户端服务器仍然可以使用较低版本形式的口令加密。	布尔值。 缺省值为 false。

示例：

```
CS_SEC_DATAORIGIN    Boolean;
CS_PROP_FIPSMODE    Boolean;
CS_HAFAILOVER    Boolean;
CS_SEC_NON_ENCRYPTION_RETRY    Boolean;
```

用于 PHP 的 Adaptive Server Enterprise 扩展模块

用于 PHP 的 Adaptive Server Enterprise 扩展模块得到了增强，现可支持在非调试时间装载 PHP 驱动程序。

在非调试 PHP 运行时装载 PHP 调试驱动程序

在 Sybase SDK 15.7 SP 120 版中，您可在除 Linux 和 Windows 以外的所有支持平台上在非调试 PHP 运行时装载 PHP 驱动程序的常规变体和调试变体。

调试 PHP 驱动程序将不再在调试 PHP 运行时装载。

在 Linux 和 Windows 上，常规 PHP 驱动程序仍可在常规 PHP 运行时装载，且调试 PHP 驱动程序可在调试 PHP 运行时装载。

SP110 的新增功能

SP110 为 Open Client 15.7、Open Server 15.7、SDK 15.7、适用于 Perl 15.7 的 Adaptive Server Enterprise 数据提供程序和适用于 Python 15.7 的 Adaptive Server Enterprise 扩展模块引入了新的功能和属性并更新了一些功能和属性。

Open Client 15.7 和 Open Server 15.7 功能

Open Client 15.7 和 Open Server 15.7 可通过更新后的功能与新的连接属性来支持 Client-Library。

Open Server 中新增函数 srv_msgq_set_blocking_threshold

利用新的 API 函数 **srv_msgq_set_blocking_threshold()** 可以设置在不阻塞发送线程的情况下，消息队列中可存储的消息数的阈值。

语法

```
CS_RETCODE srv_msgq_set_blocking_threshold(SRV_OBJID mqid, CS_INT
threshold)
```

参数

- *mqid*
  要设置阻塞阈值的消息队列的标识符。
- *threshold*
  在不阻塞发送线程的情况下，消息队列中可存储的最大消息数；如果不指定任何阈值，则使用 CS_NO_LIMIT。

返回值

返回内容	表示
CS_SUCCEED	阈值设置正确。
CS_FAIL	阈值设置不正确。

用法示例

```
/*
** We want the threads to block if there are already 10 messages
** in the queue.
*/
ret = srv_msgq_set_blocking_threshold(mqid, 10);
```

注释

- 缺省值（在没有调用 **srv_msgq_set_blocking_threshold()** 时）为 **CS_NO_LIMIT**。
- 在较低版本的 Open Server 中，将消息队列行为的阈值设置为 **CS_NO_LIMIT**。 当服务器范围内所存储的消息数达到最大时，**srv_putmsgq()** 不会阻塞，而是会失败。 服务器范围内的最大消息数是通过 **SRV_S_MSGPOOL** 属性指定的。
- 将阈值设置为 0（零）会导致每次针对此消息队列调用 **srv_putmsgq()** 时遭到阻塞，直至通过 **srv_getmsgq()** 检索到消息。
- 不可将阈值设置为 **CS_NO_LIMIT** 以外的负值。
- 所设置的阈值不得大于消息队列中可存储的服务器范围内最大消息数。 服务器范围内的最大消息数是通过 **SRV_S_MSGPOOL** 属性指定的。
- 如果所设置的阈值小于队列中的当前消息数，那么添加新的消息会阻塞调用线程，直至从队列中删除了足够的消息并达到新的限制。
- 如果所设置的阈值大于队列中的当前消息数，那么当从队列中删除消息时，将逐个取消阻塞之前遭到阻塞的线程。
- 调用带 *SRV_M_WAIT* 标志的 **srv_putmsgq()** 不算在阈值中。 使用此标志已经会导致调用程序阻塞，因为其将一直等到再次从队列中检索到消息本身。

CS_DATAFMT 格式说明符

向 **CS_DATAFMT** 结构的“格式”位屏蔽元素中添加了一个新的格式说明符 **CS_FMT_SUBS_ILL_CHAR**，可利用此说明符对 Adaptive Server 发送的非法字符进行转换。

当 Adaptive Server 使用 **enable permissive unicode** 配置参数时，客户端可能会收到非法的 unicode 字符。 设置 **CS_FMT_SUBS_ILL_CHAR** 可以实现非 Unicode 数据的成功转换。

低于 15.7 的版本在遇到非法字符时会报告错误。

新的连接属性

通过新的 Open Client 和 Open Server 连接属性，可在连接时指定缺省数据库。

- **CS_PROP_INITIAL_DATABASE** - 在连接时用于设置初始数据库。 在连接期间，成功登录之后会将参数化的 **use database** 命令发送到服务器。 即使 **use database** 命令失败，连接也会成功。 如果在行内执行错误处理，则使用 **ct_diag()** 来检查是否存在指示 **use database** 命令成功或是失败的高速缓存错误消息。 **ct_diag()** 是在完成 **ct_connect()** 之后调用的。 如果安装有客户端消息回调处理程序，则该处理程序将作为 **use database** 命令的结果进行调用。 该处理程序将检查生成的消息，并决定要如何处理失败的 **use database** 命令。 它可以返回 **CS_FAIL** 以终止连接，也可以返回 **CS_SUCCEED** 来指示可以忽略此次失败。
- **CS_PROP_CURRENT_DATABASE** - 在 **ct_connect()** 完成后，包含连接曾使用的上次报告的数据库。 此属性是在客户端库查看来自服务器的 *ENVCHANGE* 数据库令牌时设置的。

- **CS_PROP_USE_LAST_DATABASE** - 布尔属性，与 **CS_HAFAILOVER** 一起用于将故障切换后的数据库设置为最近的 **use database** 命令的结果。 当为 true 时，会导致 **CS_PROP_INITIAL_DATABASE** 更新为服务器在 ENVCHANGE 数据库令牌流中发送的所报告数据库的名称。 在故障切换时，此更新后的值将设置连接数据库。

新增服务器属性 SRV_S_ADJUSTRECVPARAMLEN

srv_descfmt API 可以利用 **SRV_S_ADJUSTRECVPARAMLEN** 属性返回从客户端收到的调整后参数数据的最大长度。

在 15.7 SP 110 版中，Open Server 应用程序可将 **SRV_S_ADJUSTRECVPARAMLEN** 属性设置为 *CS_TRUE*，这样 **srv_descfmt** 就可以检索并调整从客户端收到的参数的最大长度，该长度需足以在 Open Server 对收到的数据进行字符集转换时存储参数数据。

为了维护与现有应用程序的向后兼容性，**SRV_S_ADJUSTRECVPARAMLEN** 属性的缺省值为 *CS_FALSE*。

SDK 15.7 中针对 Adaptive Server Enterprise 驱动程序和提供程序的功能

SP110 为 Adaptive Server ODBC 驱动程序 15.7 和 jConnect for JDBC 7.07 引入了更新的功能。

对 Adaptive Server ODBC 驱动程序进行共享内存诊断

Adaptive Server ODBC 驱动程序允许应用程序用户和管理员监控驱动程序和数据库的性能。

通过应用程序编程，或者使用新提供的实用程序 **aseodbcstatus**.均可访问此信息。 要使用 aseodbcstatus 实用程序，必须将检测配置为使用共享内存。

在不修改应用程序的情况下启用 Adaptive Server ODBC 驱动程序检测
要启用 ODBC 检测，可以设置以下环境变量之一:

- SYBASE_ODBC_FORCE_INSTRUMENTATION=1 - 将 Adaptive Server ODBC 驱动程序配置为启用检测。 如果没有设置此环境变量值或者将其设置为 1 以外的值，则可通过编程方式启用检测。
- SYBASE_ODBC_INSTRUMENTATION_FINE=1 - 将 Adaptive Server ODBC 驱动程序配置为在语句级别监控网络通信量。 在设置此变量后，将通过当前正在执行的语句来保存网络时间。 将此环境变量设置为 1 表示应用程序无法使用多个线程访问 Sybase ODBC 库。 如果没有设置环境变量，则会在应用程序级别收集网络时间。

配置共享内存检测
借助共享内存，可以通过 **aseodbcstatus** 实用程序获得检测数据。 共享内存段是通过 SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM 环境变量进行启用和配置的。

```
SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM=<number of diagnostic
sections to put in one shared memory segment>
(example 512)
```

要启用共享内存检测，请将 SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM 设置为大于零的值（缺省值）。

将环境变量设置为较小的数值可能会导致 Adaptive Server ODBC 驱动程序使用许多共享内存段，进而可能会对操作系统性能造成影响，影响程度取决于具体的操作系统。将环境变量设置为较大的数值可能会导致 Adaptive Server ODBC 驱动程序使用比所需的共享内存段更大的段。

如果大概知道应用程序所用的语句数量，请将 SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM 值设置为比该数值略大的值。 例如，如果应用程序使用的语句数量在 250 到 350 之间，则将 SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM 的值设置为 360。 如果应用程序使用的语句数量范围较大（例如，语句数量在 100 到 10000 之间），那么使用最大语句数量可能会导致在应用程序没有使用所有语句时占用过多内存。 请改为使用较小的值并允许增加内存块的数量。 在这种情况下，请尝试使用应用程序通常使用的最少语句数量的二倍。

使用 ***aseodbcstatus*** *实用程序检索检测数据*

要检索检测数据，请使用 **aseodbcstatus** 实用程序，该程序可连接到共享内存段并显示检测数据。

aseodbcstatus 接受以下参数：

- **-help** – 显示有效参数列表。
- **-check <memory_area> <pid>** – 针对给定进程 ID 检查指定内存区的可用性。 如果内存区不可用，**aseodbcstatus** 将以非零状态退出。
- **-print <memory_area> <pid>** – 针对给定进程 ID 输出指定内存区中包含的检测数据。如果数据是未知的，（例如，所用 **asedobcstatus** 的版本低于 ODBC 驱动程序的版本）**asedobcstatus** 将以非零状态退出。
- **-statement_diagnostics <pid> <sid> <filter | all>** - 输出指定语句 ID 的检测数据 **(<sid>)** 为语句 ID 传递 -1 将输出所有语句的数据。 如果传入 **filter**，则仅显示含非零计数的检测数据。

aseodbcstatus 实用程序具有若干用于控制所检索数据的内存区。 可能的值有：

- InstrumentationTimes - 特定 ID 的全局检测数据。 这是该进程所用所有连接和语句的组合数据。
- InstrumentationTimesName – InstrumentationTimes 和 **statement_diagnostics** 检测数据中各行的名称列表，按顺序显示。
- StatementIDs – 列出用于 **statement_diagnostics** 的语句 ID。

通过编程方式使用检测
应用程序可以直接使用环境、连接和语句属性来启用和访问检测。 环境和连接属性是相同的，两者均可在应用程序的全局范围内应用。 如果应用程序使用的驱动程序管理器不支持自定义环境属性，则该应用程序可以使用连接属性。 这些属性包括：

- **SQL_ATTR_INSTRUMENTATION** - 控制检测行为。 支持的值包括：
  - **SQL_INSTRUMENTATION_ENABLE** - 开启检测数据收集。
  - **SQL_INSTRUMENTATION_DISABLE** - 关闭检测数据收集。
  - **SQL_INSTRUMENTATION_CLEAR** - 这是语句属性支持的唯一值。 在环境或连接属性中设置时，**SQL_INSTRUMENTATION_CLEAR** 将清除全局检测数据。 在语句属性中设置时，**SQL_INSTRUMENTATION_CLEAR** 将清除该语句的检测数据。
  - **SQL_INSTRUMENTATION_CLEAR_ALL** - 清除全局检测和所有语句检测。
  - **SQL_INSTRUMENTATION_FINE** - 启用更为详细的检测数据收集，其中包括锁、网络、**select** 语句以及批处理的各个方面。
- **SQL_ATTR_INSTRUMENTATION_LOG** - 检索格式为 **SQLWCHAR** 字符串的检测数据。 当在环境或连接中使用时，**SQL_ATTR_INTRUMENTATION_LOG** 将检索全局检测数据。 当在语句中使用时，**SQL_ATTR_INTRUMENTATION_LOG** 仅检索该语句的检测数据。 该字符串的格式为以分号分隔的列表。 每一项的格式为：

  ```
  <instrumentation name>:<time in us>,<count >
  ```

  例如：

  ```
  Unknown:0,0; SocketRetrieve:75,19;
  Waiting for lock XATransactionManager:0,0;
  Holding lock XATransactionManager:0,
  0; SQLAllocHandle:149,20;
  ```

64 位 Linux 中支持的 Sybase iAnywhere ODBC 驱动程序管理器

Adaptive Server Enterprise ODBC 驱动程序 15.7 SP 110 版在 Linux x86_64 和 Linux Power 64 位中支持 Sybase iAnywhere ODBC 驱动程序管理器 16.0 版。

有关支持平台的信息，请参见《Sybase Adaptive Server Enterprise ODBC 驱动程序用户指南 15.7》。

注意： Microsoft Windows 中不支持 Sybase iAnywhere ODBC 驱动程序管理器 16.0 版。

jConnect 中的 PRE_CACHE_DATATYPE_INFO 连接属性

jConnect 使用 **PRE_CACHE_DATATYPE_INFO** 连接属性在登录时对数据类型元数据进行高速缓存，这样可以提高后续使用时的数据访问性能。

如果重复使用 **Statement** 或其派生接口来获取数据类型元数据，那么将 **PRE_CACHE_DATATYPE_INFO** 设置为 true 可以提高性能。

如果将 **PRE_CACHE_DATATYPE_INFO** 设置为 true，连接时会对服务不同 `ResultsetMetadata` API（如 `isCaseSensitive` 和 `isSearchable`）的所有用

户定义数据类型的相关信息进行高速缓存。 然后，可通过高速缓存对此信息进行后续访问。

如果 **PRE_CACHE_DATATYPE_INFO** 为 false（缺省值），jConnect 不会高速缓存任何用户定义的数据类型信息。

注意： 根据要获取其连接的数据库中所存在的用户定义数据类型数，建立连接的时间可能会有所增加。

用于 Perl 的 Adaptive Server Enterprise 扩展模块

用于 Perl 的 Adaptive Server Enterprise 扩展模块支持 Kerberos 连接、数据源名称样式（DSN 样式）连接的更新属性和方法、区域设置和字符集配置以及更新的数据库句柄属性。

Perl 驱动程序的 DSN 样式连接属性

在 Perl 驱动程序中添加了若干新 DSN 属性，并对一些属性进行了更改。

下面是 15.7 SP 110 版中当前支持的属性及其值的权威列表。

SybaseASE 驱动程序连接语法
对于 **DBI connect()** 方法，在建立属性和值对时以下规则适用。 该方法的 DSN 字符串参数必须包含 **dbi:SybaseASE:**，后接一个或多个以分号 (;) 分隔的字符串，其格式为 *name=value*，其中：

- **名称** - 该值不区分大小写，可由等号 (=) 或分号 (;) 分隔。 一个属性可以具有多个同义词。 例如，server 和 servername 表示同一个属性。
- **等号 (=)** - 表示开始将值分配给名称。 如果没有等号，则名称将被视为布尔类型，且值为 true。
- **值** - 以分号 (;) 终止的字符串。 如果值中已经包含分号或反斜杠 (\)，请再使用一个反斜杠。 值的类型可以是布尔型、整数型或字符串型。 布尔类型的有效值为 true、false、on、off、1 和 0。

  注意： 如果存在没有值的布尔类型名称，则布尔类型将被设置为 true。

有效的属性名称和值
下表列出了 **dsn** 关键字参数的统一属性名称和值。

名称	说明	值
ANSINull	确定在 SQL 的等于 (=) 或不等于 (!=) 比较计算中，NULL 值操作数的求值是否符合 ANSI 标准。 如果值为 true，Adaptive Server 将 ANSI 行为强制为 *= NULL* 和 *is NULL* 不等效。 在标准 Transact-SQL® 中，*= NULL* 和 *is NULL* 被视为等效。 此选项还将以相类似的方式影响 *<> NULL* 和 *is not NULL* 行为。	布尔值。 缺省值为 false。
BulkLogin	确定是否启用连接以执行批量复制操作。	布尔值。 缺省值为 false。
ChainXacts	如果值为 true，Adaptive Server 将采用链式事务行为，即每个服务器命令都将视为不同的事务。 Adaptive Server 将在以下任意语句之前隐式执行一个 begin transaction 命令： **delete**、**fetch**、**insert**、**open**、**select** 和 **update**。 仍必须显式结束事务或回退事务。 如果值为 false，则应用程序必须指定与 **commit** 或 **rollback** 语句成对出现的显式 **begin transaction** 语句。	布尔值。 缺省值为 false。
Charset	指定用于此连接的**字符集**。	字符串值。 缺省字符集现已设置为 **iso_1**。
Confidentiality	是否已对连接执行数据加密服务。	布尔值。 缺省值为 false。
CredentialDelegation	确定是否允许服务器使用用户的委托证书连接到另一个服务器。	布尔值。 缺省值为 false。
DetectReplay	确定连接的安全机制是否检测回放的传输。	布尔值。 缺省值为 false。
DetectOutOfSequence	确定连接的安全机制是否检测到达顺序混乱的传输。	布尔值。 缺省值为 false。

名称	说明	值
Hostname	客户端计算机的主机名。	字符串值
HostPort	指定要连接的服务器的主机和端口组合。	字符串值。 字符串格式为 “hostname portnumber” 或 “host-name:portnumber” 。
Integrity	确定连接的安全机制是否执行数据完整性检查。	布尔值。 缺省值为 false。
Interfaces	interfaces 文件的路径和名称。	字符串值。
Keytab	连接的安全机制会从一个文件中读取与 *username* 值相配的安全密钥，此属性表示该文件的路径和名称。	字符串值。 缺省值为 NULL，即用户必须在连接之前创建证书。
Locale	确定消息、数据类型转换以及日期时间格式所使用的语言和字符集。	字符串值。
Language	确定消息、数据类型转换以及日期时间格式所使用的语言集。	字符串值。
LoginTimeout	指定登录超时值。	整数值。
MaxConnect	指定某上下文可同时打开的最大连接数量。	整数值。 缺省值为 25。 不允许负值和零值。
MutualAuthen-tication	确定服务器是否必须向客户端验证自身。	布尔值。 缺省值为 false。
NetworkAu-thentication	确定连接的安全机制是否执行基于网络的用户验证。	布尔值。 缺省值为 false。
PacketSize	指定 TDS 包大小。	整数值。
Password	指定用于登录服务器的口令。	字符串值。
PasswordEn-cryption **EncryptPass-word**	确定连接是否使用非对称口令加密。	布尔值。 缺省值为 false。

名称	说明	值
SecurityMechanism	指定执行连接安全服务的网络安全机制名称。	字符串值。 缺省值取决于安全驱动程序配置。
Server **Servername**	指定连接到的服务器的名称。	字符串值。
ServerPrincipalName **Kerberos**	指定已打开连接的服务器的网络安全主体名。	字符串值。 缺省值为 NULL，这意味着此连接假定服务器的主体名称与其 *ServerName* 值相同。
ScriptName	登录到服务器时使用的应用程序名。	字符串值。
SslCAFile	受托 CA 证书所在文件的路径。	字符串值。
TDSKeepalive	确定是否使用 KEEPALIVE 选项。	布尔值。 缺省值为 true。
Timeout	指定连接超时值。	整数值。
UID **User** **Username**	指定用于登录服务器的用户名称。	字符串值。

用于 **Python** 的 **Adaptive Server Enterprise** 扩展模块

用于 Python 的 Adaptive Server Enterprise 扩展模块支持用于进行批量操作和批量复制 LOB 列的新属性。

为批量复制操作设置属性

在启动批量复制操作之前，应用程序可以设置特定的批量属性。

使用 `blkcursor` 对象的 **copy()** 方法来设置属性。

该方法接受以下参数：

- **name** - 要执行批量复制操作的表的名称。
- **direction** - 关键字参数，其值包括：in 和 out。
- **properties** - 操作的属性。 该参数是一个以分号分隔的字符串，其格式为“名称=值”：

- **名称** - 该值不区分大小写，可由等号 (=) 或分号 (;) 分隔。 一个属性可以具有多个同义词。
- **等号 (=)** - 表示开始将值分配给名称。 如果没有等号，则名称将被视为布尔类型，且值为 true。
- **值** - 以分号 (;) 终止的字符串。 如果值中已经包含分号或反斜杠 (\)，请再使用一个反斜杠。 值的类型可以是布尔型、整数型或字符串型。 布尔类型的有效值为 true、false、on、off、1 和 0。

注意： 如果存在没有值的布尔型名称，则必须将布尔类型设置为 true。

示例：

```
blk.copy(name="mytable", direction="in",
properties="IdStartNum=21")
```

有效的属性名称和值

属性关键字参数的有效属性名称和值。

名称	说明	值
Identity	是否为要插入的每一行显式指定表中 identity 列的值。 如果已为批量拷入操作设置 IdStartNum 属性，则不能将此属性设置为 true。	布尔型；缺省值为 false。
IdStart-Num	被插入的行中 identity 列的起始值。 第一个被插入的行将使用此值，而且后续每行中的值都是递增的。 如果已将批量拷入操作的 Identity 属性设置为 true，则不能设置此属性。	整数值；无缺省值。

在含 Identity 列的表中执行批量复制操作

在含 identity 列的表中执行批量复制操作。

当在涉及 identity 列的批量拷入操作中传输行时，在缺省情况下将不指定 identity 列的值。 这些值由服务器生成。

例如：

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulklogin=true;chainxacts=off")
cur = conn.cursor()
cur.execute("create table mytable (empid int identity, empname
varchar(20))")
            cur.close()
            blk = conn.blkcursor()

# Start bulk copy in operation. Do not specify values for identity
columns.
# Values will be generated by the server.
blk.copy(name="mytable", direction="in")
blk.rowxfer(["Joanne"])
blk.rowxfer(["John"])
blk.done()
```

- **IdStartnum** 属性用于指定 identity 列的起始值。
  例如：

```
# Specify starting identity column value of 11 for the copy
operation.
blk.copy(name="mytable", direction="in",
properties="IdStartNum=11")
blk.rowxfer(["Max"])
blk.rowxfer(["Danny"])
blk.done()
```

- **Identity** 属性供应用程序用于显式指定 identity 列的值。
  例如：

```
# Values for identity columns will have to be specified.
blk.copy(name="mytable", direction="in",
properties="identity=on")
blk.rowxfer([21, "Maya"])
blk.rowxfer([22, "Uma"])
blk.done()
```

LOB 列的批量复制

Python 模块现支持涉及 text 和 image (LOB) 列的批量复制操作。

LOB 对象的构造方法、类型和方法
应用程序提供了特殊的构造方法和类型，用来创建具有特殊值的对象。 应用程序必须使用一种构造方法将 Python 对象作为 text 或 image 列进行绑定才能执行批量拷入操作。 传递到 `blkcursor` 方法时，模块便可以为输入参数检测合适的类型，并进行相应绑定。

大对象 (LOB) 支持

Python 支持使用大对象 (LOB) 数据类型 - *text* 和 *image*。

构造方法：

Lob(type, obj) - 创建具有 LOB 值的对象。

它可以采用以下参数：

- `type` - LOB 对象的类型。 其值可以是 TEXT 或 IMAGE，如下方指定。
- `obj` – Python 缓冲区对象。 它可以是支持缓冲协议的任意对象。 此类对象包含内置 bytearray。

类型

TEXT 类型 - 描述数据库中的 text 列。

IMAGE 类型 - 描述数据库中的 image 列。

LOB 对象方法

.readinto(bytearray) – 必须用于批量拷出操作，以便为绑定到 text 或 image 列的 LOB 对象获取数据。 此方法返回读取的字节数。 如果返回 None 对象，则表示列值已完

全复制。 应用程序必须重复调用此方法，直到返回 None 为止。 每个块中所读取的字节数由 **bytearray** 的大小决定。 此方法采用以下参数：

bytearray - 列中的数据已读取并复制到该数组。

批量拷入操作中的 LOB 列

对于批量拷入操作，应用程序必须使用 `LOB()` 构造方法标记 Python 对象，以便传输 text 或 image 列。

例如：

```
conn = sybpydb.connect(dsn="user=sa;bulklogin=true;chainxacts=off")
cur = conn.cursor()cur.execute("create table mytable (id int, t text,
i image)")
cur.close()
blk = conn.blkcursor()
blk.copy("mytable", direction="in")

# Transfer text and image data using a bytearray.
arr1 = bytearray(b"hello this is some text data")
lob1 = sybpydb.Lob(sybpydb.TEXT, arr1)
arr2 = bytearray(b"hello this is some image data")
lob2 = sybpydb.Lob(sybpydb.IMAGE, arr2)
blk.rowxfer([1, lob1, lob2])
```

在 Python 中，可通过多种方式对文件进行打开和读取。 下面的示例显示使用内存映射在批量复制操作中传输文件：

```
# Transfer data from a file using memory maps.
fh1 = open(“file1”, "rb")
mp1 = mmap.mmap(fh1.fileno(), 0 , access=mmap.ACCESS_READ)
arr1 = bytearray(mp1)
lob1 = sybpydb.Lob(sybpydb.TEXT, arr1)
fh2 = open(“file2”, "rb")
mp2 = mmap.mmap(fh2.fileno(), 0 , access=mmap.ACCESS_READ)
arr2 = bytearray(mp2)
lob2 = sybpydb.Lob(sybpydb.IMAGE, arr2)
blk.rowxfer([2, lob1, lob2])
```

批量拷出操作中的 LOB 列

对 text 和 image 列进行批量拷出操作时，所要传输的 text 和 image 列必须位于行的结尾。

text 和 image 列的数据将作为 LOB 对象返回。 表中各行必须逐个进行传输。 在所有行均传输完毕后，必须检索行中每个 LOB 对象的数据。 必须重复调用 `readinto()` 方法，直到返回表示完整列值已复制的 None 对象。

例如：

```
# Method to read data from a lob object
def getlobdata(lob):
    outarr = bytearray()
    chunk = bytearray(1024)
    while True:
        len = lob.readinto(chunk);
```

```
        if (len == None):
             break
        outarr.extend(chunk[:len])
return outarr

blk.copy("mytable", direction="out")
# The rows should be transferred one by one.
row = blk.rowxfer()
print(row[0])
# Now read the lob data for the text column column
arr1 = getlobdata(row[1])
print(arr1.decode())
# Now read the lob data for the text column column
arr2 = getlobdata(row[2])
print(arr2.decode())
```

SP100 的新增功能

SP100 为 Open Client 15.7、Open Server 15.7 和 SDK 15.7 引入了版本号更改并更新了一些功能。

发行版本号更改

目前 Sybase® 客户熟知的在主要或次要版本之后的 ESD (Electronic Software Deliveries) 软件修补程序现在称为 SP（支持包），其编号最多为三位数字。

有关所有主要软件版本，请参见 SAP® 发行策略，网址为： https://service.sap.com/releasestrategy。此次版本号更改不会造成升级或降级过程的更改。

安装程序更改

对 SDK 和 Open Server 安装程序的版本兼容性和向后兼容性进行了增强。

- SDK 和 Open Server 安装程序现在会检查您所安装的版本是否与目标目录中的版本兼容并能够在其基础上安装。
  如果所安装的版本中不含目标目录版本所需的缺陷修复，即会将安装视为不兼容。
  如果已安装的版本兼容，那么将正常进行安装。
  如果已安装的版本与您正在安装的版本不兼容，那么将停止安装过程。 您可以：
  - 覆盖错误继续操作，或
  - 中止安装。 访问软件下载网站以查看是否有可用的兼容版本。
- 对于向后兼容性，安装程序将安装从 15.7 GA 到 15.7 SP100 的所有安全和目录驱动程序文件版本。

Open Client 15.7 和 Open Server 15.7 功能

Open Client 15.7 和 Open Server 15.7 支持新的 MIT Kerberos 库。

新的 MIT Kerberos 库支持 Sybase Kerberos 驱动程序

适用于 Windows 64 位的 4.0.1 版 MIT Kerberos 新库可用于 Sybase Kerberos 驱动程序，`libsybskrb64.dll`。

要在 Windows 64 位上使用 MIT Kerberos GSS 库，请将此条目添加到 `%SYBASE%\OCS-15_0\ini\libtcl64.cfg` 文件的 SECURITY 部分：

```
[SECURITY]
csfkrb5=libsybskrb64.dll secbase=@MYREALM libgss=C:
\Kerberos_winx64\bin\gssapi64.dll
```

此处 *C:\Kerberos_winx64* 是 MIT Kerberos 的安装位置。

注意： Kerberos gssapi 库的路径不能包含空格。

SDK 15.7 中针对 Adaptive Server Enterprise 驱动程序和提供程序的功能

SP100 为 Adaptive Server ODBC 驱动程序 15.7、jConnect 7.07 和 Adaptive Server ADO.NET 数据提供程序 15.7 引入了新功能。

WindowsCharsetConverter 连接属性

（仅限 Microsoft Windows）从 15.7 SP 100 版开始，用户可通过新连接属性 **WindowsCharsetConverter** 来选择要使用的转换库是 Sybase Unicode Infrastructure Library (Unilib) 还是 Microsoft Unicode 转换例程。

在 15.5 及更高版本中，Windows 平台上的 Adaptive Server Enterprise ADO.NET 数据提供程序、Adaptive Server Enterprise OLEDB 提供程序和 Adaptive Server Enterprise ODBC 驱动程序使用 Sybase Unicode Infrastructure Library (Unilib) 进行字符集转换。

低于 15.5 的版本使用 Microsoft Unicode 转换例程。

两种转换库执行转换的方式稍有不同。

在连接字符串中，将 **WindowsCharsetConverter** 设置为以下值时所用的转换库不同：

- 0 -（缺省值）使用 Unilib 库。
- 1 - 使用 Microsoft Unicode 转换例程。

注意： 如果应用程序依赖于不同于 Unilib 的特定转换差异，则使用 Microsoft Unicode 转换例程。

在非 Windows 操作系统中，仅支持使用 Unilib 进行字符集转换；将 **WindowsCharsetConverter** 设置为 1 不起作用。

使用 SSIS 自定义数据流目标组件可使 SQL Server 2012 向 Adaptive Server 进行更快的数据传输

Adaptive Server ADO.NET 数据提供程序分布中包括与 SQLServer 2012 兼容的 SQL Server 集成服务 (SSIS) 自定义数据流目标组件，该组件可使用批量插入协议向 Adaptive Server Enterprise 更快地传输数据。

自定义数据流目标组件使用 `AseBulkCopy` 类支持的 Adaptive Server 批量插入协议。此组件名为 `Sybase.AdoNet4.AseDestination.dll`，随 Adaptive Server ADO.NET 数据提供程序一起安装，在 32 位系统和 64 位系统中分别安装在 `%SYBASE%\DataAccess\ADONET\dll` 和 `%SYBASE%\DataAccess64\ADONET\dll` 中。

有关与 SQLServer 2008 兼容的自定义数据流目标组件的版本，请参见 ESD #5 中的“新 SSIS 自定义数据流目标组件可向 Adaptive Server 更快地传输数据”部分。

注意： 用于从 SQL Server 2008 进行数据传输的 SSIS 目标组件已从 `Sybase.AdaptiveServerAdoNetDestination.dll` 重命名为 `Sybase.AdoNet2.AseDestination.dll`。

配置 Adaptive Server ADO.NET 目标 SSIS 组件

Adaptive Server ADO.NET 目标 SSIS 组件可以向 Adaptive Server 目标更快地传输数据。

1. 将 `Sybase.AdoNet4.AseDestination.dll` 复制到 `C:\Program Files\Microsoft SQL Server\110\DTS\PipelineComponents` 和 `C:\Program Files (x86)\Microsoft SQL Server\110\DTS\PipelineComponents`。
2. 在步骤 1 中所用本地驱动器中的任一 Microsoft SQL Server 目录中，使用 SDK 安装中提供的 `AseGacUtility4.exe` 来注册 `Sybase.AdoNet4.AseDestination.dll`。
3. 要在 Windows 中启动 SQLServer 2012 Data Tools 或 SQL Server 2012 Data，请选择**“开始”** > **“所有程序”** > **“Microsoft SQL Server 2012”** > **“SQL Server Data Tools”**。
4. 选择**“文件”** > **“新建”** > **“项目”** > **“Integration Services 项目”**。SSIS 工具箱中将自动出现 Sybase 目标组件。
5. 从**“控制流项”**工具箱中拖放一个“控制流”对象。
6. 选择**“数据流目标”**选项卡，然后选择**“数据流源工具箱”** (Data Flow Sources Toolbox) 选项卡，再将 **“Sybase AdoNet4 ASE 目标”** (Sybase AdoNet4 ASE Destination) 和 **“ADO NET 源组件”** (ADO NET Source Component) 拖放到“数据流”选项卡。
7. 如果“连接管理器”窗口中没有可用的源或目标连接，则右键单击“连接管理器”窗口并选择**“新建 ADO.NET 连接”**。 如果已经存在现有数据连接，则将其选中，或单击**“新建”**。
8. 要创建与目标 Adaptive Server 的新连接，请在“配置 ADO.NET 连接管理器”窗口中单击**“新建”**，然后选择 **“Sybase Adaptive Server Enterprise 数据提供程序”** (Sybase Adaptive Server Enterprise Data Provider)。
9. 在“连接管理器”窗口中输入连接属性。
10. 要启用批量插入，请在“其它连接属性”(Additional Connection Props) 文本框中输入：

    ```
    enablebulkload=1
    ```

    注意： 有关使用批量插入的更多详细信息，请参见《Adaptive Server Enterprise ADO.NET 数据提供程序用户指南》。
11. 单击**“确定”**。

12. 对于数据流中的 ADO.NET 源，设置连接和数据访问模式。 在从 ADO.NET 源连接数据流路径后，右键单击 **“Sybase AdoNet4 ASE 目标”** (Sybase AdoNet4 ASE Destination)，并选择**“显示高级编辑”** (Show Advanced Edit)。
13. 在**“连接管理器”**选项卡中，从“连接管理器”字段中选择 ASE 连接。 在**“组件属性”**选项卡中，将 TableName 属性设置为目标表名。
14. 选择**“输入列”**选项卡，然后选择**“名称”**。 这将选择源表指定的所有列。
15. 单击**“确定”**以建立连接。

    有关使用 SQL Server 集成服务进行数据传输的详细信息，请参见 Microsoft SSIS 文档。

Adaptive Server ADO.NET 数据提供程序支持 SSRS

Adaptive Server ADO.NET 数据提供程序分布中包括 Microsoft SQL Server 报告服务 (SSRS) 自定义数据扩展组件，用户可通过该组件在报告服务器中存储证书。

Adaptive Server SSRS 组件支持：

- Microsoft SQL Server 2008
- Microsoft SQL Server 2008 R2

此组件名为 Sybase.AdoNet2.AseReportingServices，随 Adaptive Server ADO.NET 数据提供程序一起安装在以下位置：在 32 位系统中，`%SYBASE%\DataAccess\ADONET\dll`；在 64 位系统中，`%SYBASE%\DataAccess64\ADONET\dll`。

配置 Adaptive Server ADO.NET SSRS 组件

配置 Adaptive Server ADO.NET SSRS 组件。

1. 将 `Sybase.AdoNet2.AseReportingServices.dll` 复制到 `C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies`。
2. 使用文本编辑器打开 `C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies` 中的 `RSReportDesigner.config`。
   - 在“数据”部分输入以下内容：

     ```
     <Extension Name="Sybase"
     Type="Sybase.AdoNet2.AseReportingServices.SybaseClientConnectionWrapper,Sybase.AdoNet2.AseReportingServices"/>
     ```
   - 在“设计器”部分输入以下内容：

     ```
     <Extension Name="Sybase"
     Type="Microsoft.ReportingServices.QueryDesigners.GenericQueryDesigner,Microsoft.ReportingServices.QueryDesigners"/>
     ```
3. 保存 `RSReportDesigner.config` 文件。

针对 Adaptive Server Enterprise 驱动程序和提供程序的 LDAPS 功能

当在 LDAP URL 中指定 `ldaps` 而不是 `ldap` 时，将与 LDAP 服务器建立 SSL 连接。

UNIX

以下示例显示了必须为 `odbc.ini`（或连接字符串）中的 DSN 指定的属性：

```
DSURL = ldaps://huey:636/dc=sybase,dc=com????
bindname=cn=Manager,dc=Sybase,dc=com?secret
DSServiceName = myAse
TrustedFile = /usr/u/sybase/config/trusted.txt
```

用于对 LDAP 服务器证书进行签名的证书发放机构签名证书必须附加到 `trusted.txt` 文件。

Windows

以下示例显示了必须在连接字符串中指定的属性：

```
DSURL = ldaps://huey:636/dc=sybase,dc=com????
bindname=cn=Manager,dc=Sybase,dc=com?secret
DSServiceName = myAse
```

用于对 LDAP 服务器证书进行签名的证书发放机构签名证书必须安装在 Microsoft 证书存储区中。

SAP jConnect 中的 SSL 支持

要在低于 15.7 SP 100 版的 SAP jConnect 中使用 SSL 套接字，必须创建“**SybSocketFactory**” 接口的实现并通过设置 **SYBSOCKET_FACTORY** 连接属性来使用该接口的实现。

SAP jConnect 具有对使用 SSL 套接字连接到 SAP Adaptive Server 的内置支持。 在设置为以下值时，新的连接属性 **ENABLE_SSL** 如下：

- false -（缺省值）SAP jConnect 将不使用 SSL 套接字。
- true - SAP jConnect 将使用 SSL 套接字，且目标 SAP Adaptive Server 必须启用 SSL 套接字连接。 SAP jConnect 将忽略 **SYBSOCKET_FACTORY** 连接属性。

注意： SAP 建议使用 **DriverManager.setLoginTimeout** 属性设置登录超时，从而允许在未启用 SSL 的 SAP Adaptive Server 上尝试建立 SSL 连接时出现连接超时。

SSL 套接字功能取决于以下标准 Java 属性：

- **javax.net.ssl.keyStore**
- **javax.net.ssl.keyStorePassword**
- **javax.net.ssl.trustStore**
- **javax.net.ssl.trustStorePassword**
- **javax.net.ssl.trustStore**

- **javax.net.ssl.trustStoreType**

有关 Java 标准属性的详细信息，请参见 Java J2SE 6 文档。

ESD #7 的新增功能

ESD #7 为 Open Client 15.7、Open Server 15.7、SDK 15.7、用于 Python 15.7 的 Adaptive Server Enterprise 扩展模块以及用于 PHP 15.7 的 Adaptive Server Enterprise 扩展模块引入了更新的功能。

Open Client 15.7 和 Open Server 15.7 功能

Open Client 15.7 和 Open Server 15.7 得到了增强，现支持 Client-Library 连接字符串属性、远程口令加密和用于 Windows 64 位的 `libsybsspiwrapper64.dll`。

Client-Library 支持连接字符串属性

Client-Library 现支持 API 例程 **ct_connect_string()**。

ct_connect_string()
通过指定连接字符串连接到服务器。

ct_connect_string() 函数提供的功能与 **ct_connect()** 相同。 它还提供了用于在连接时设置特定属性的机制。

语法

```
CS_RETCODE ct_connect_string(connection, connection_string, length)
CS_CONNECTION    *connection;
CS_CHAR                  *connection_string;
CS_INT                       length;
```

参数

- *connection* – 指向 CS_CONNECTION 结构的指针。 CS_CONNECTION 结构中包含有关特定客户端/服务器连接的信息。 使用 **ct_con_alloc** 可以指派 CS_CONNECTION 结构。
- *connection_string* - 包含属性名称和值的字符串。
- *length* - **connection_string* 的长度（以字节为单位）。 如果 **connection_string* 以空值终止，将以 CS_NULLTERM 作为长度进行传递。 如果 connection_string 为 NULL，将以 0 或 CS_UNUSED 作为长度进行传递。

返回值
ct_connect 将返回以下值：

返回内容	表示
CS_SUCCEED	例程已成功完成。

返回内容	表示
CS_FAIL	例程失败。
CS_PENDING	异步网络 I/O 有效。 请参见《Open Client Client-Library/C 参考手册》中的“异步编程”部分。
CS_BUSY	该连接已有一个待执行的异步操作。 请参见《Open Client Client-Library/C 参考手册》中的“异步编程”部分。

该连接字符串是一个以分号分隔的字符串，其格式为“名称=值”：

1. 名称 - 该值不区分大小写，可由等号 (=) 或分号 (;) 分隔。 一个属性可以具有多个同义词。 例如，**server** 和 **servername** 表示同一个属性。
2. 等号 (=) - 表示开始将值分配给名称。 如果没有等号，则名称将被视为布尔类型，且值为 true。
3. 值 - 以分号 (;) 终止的字符串。 如果值中已经包含分号或反斜杠 (\)，请再使用一个反斜杠。 值的类型可以是布尔型、整数型或字符串型。 布尔类型的有效值为 true、false、on、off、1 和 0。

注意： 如果存在没有值的布尔型名称，则必须将布尔类型设置为 true。

例如：

```
ct_connect_string(conn, "Username=me; Password=mypassword;
Servername=ASE", CS_NULLTERM);
```

有效的属性名称和值

下表列出了 **dsn** 关键字参数的有效属性名称和值。

名称	说明	值
ANSINull	确定在 SQL 的等于 (=) 或不等于 (!=) 比较计算中，NULL 值操作数的求值是否符合 ANSI 标准。 如果值为 true，Adaptive Server 将 ANSI 行为强制为 *= NULL* 和 *is NULL* 不等效。 在标准 Transact-SQL® 中，*= NULL* 和 *is NULL* 被视为等效。 此选项还将以相类似的方式影响 *<> NULL* 和 *is not NULL* 行为。	布尔值。 缺省值为 false。
BulkLogin	确定是否已启用连接以执行批量复制操作。	布尔值。 缺省值为 false。

名称	说明	值
ChainXacts	如果值为 true，Adaptive Server 将采用链式事务行为，即每个服务器命令都将视为不同的事务。 Adaptive Server 将在以下任意语句之前隐式执行一个 begin transaction 命令： **delete**、**fetch**、**insert**、**open**、**select** 和 **update**。 仍必须显式结束事务或回退事务。 如果值为 false，则应用程序必须指定与 commit 或 rollback 语句成对出现的显式 begin transaction 语句。	布尔值。 缺省值为 false。
Charset	指定用于此连接的**字符集**。	字符串值。
Confidentiality	是否已对连接执行数据加密服务。	布尔值。 缺省值为 false。
CredentialDelegation	确定是否允许服务器使用用户的委托证书连接到另一个服务器。	布尔值。 缺省值为 false。
DetectReplay	确定连接的安全机制是否检测回放的传输。	布尔值。 缺省值为 false。
DetectOutOfSequence	确定连接的安全机制是否检测到达顺序混乱的传输。	布尔值。 缺省值为 false。
Integrity	确定连接的安全机制是否执行数据完整性检查。	布尔值。 缺省值为 false。
Interfaces	interfaces 文件的路径和名称。	字符串值。
Keytab	连接的安全机制会从一个文件中读取与 *username* 值相配的安全密钥，此属性表示该文件的名称和路径。	字符串值。 缺省值为 NULL，即用户必须在连接之前创建证书。
Locale	确定消息、数据类型转换以及日期时间格式所使用的语言和字符集。	字符串值。
Language	确定消息、数据类型转换以及日期时间格式所使用的语言集。	字符串值。
LoginTimeout	指定登录超时值。	整数值。
MaxConnect	指定某上下文可同时打开的最大连接数量。	整数值。 缺省值为 25。 不允许负值和零值。

名称	说明	值
MutualAuthentication	确定服务器是否必须向客户端验证自身。	布尔值。 缺省值为 false。
NetworkAuthentication	确定连接的安全机制是否执行基于网络的用户验证。	布尔值。 缺省值为 false。
PacketSize	指定 TDS 包大小。	整数值。
Password	指定用于登录服务器的口令。	字符串值。
PasswordEncryption	确定连接是否使用非对称口令加密。	布尔值。 缺省值为 false。
SecurityMechanism	指定执行连接安全服务的网络安全机制名称。	字符串值。 缺省值取决于安全驱动程序配置。
Server **Servername**	指定连接到的服务器的名称。	字符串值。
ServerPrincipalName	指定已打开连接的服务器的网络安全主体名。	字符串值。 缺省值为 NULL，这意味着此连接假定服务器主体名与其 *ServerName* 值相同。
TDS_Keepalive	确定是否使用 KEEPALIVE 选项。	布尔值。 缺省值为 true。
Timeout	指定连接超时值。	整数值。
UID **User** **Username**	指定用于登录服务器的用户名称。	字符串值。

远程口令加密

Open Server 支持使用扩展增强加密口令 (EPEP) 检索连接的远程口令对。

检索属性（包括 SRV_T_NUMRMTPWDS 和 SRV_T_RMTPWDS）与 **srv_thread_props()** 配合使用。 如果客户端支持 EPEP 协议，SRV_T_NUMRMTPWDS 属性将返回解密远程口令对的数量，SRV_T_RMTPWDS 属性将返回口令对。

用于 Windows 64 位的 libsybsspiwrapper64.dll

通过使用 `libsybsspiwrapper64.dll` 包装库可允许 Kerberos 安全驱动程序使用 Windows 64 位平台上的 Windows 安全支持提供程序接口 (SSPI) 例程。

要使用此功能，必须编辑 `libtcl64.cfg` 以加入 `libsybsspiwrapper64.dll`。例如：

```
[SECURITY]csfkrb5=LIBSYBSKRB64 secbase=@MYREALM libgss=C:\Sybase
\release\OCS-15_0\lib3p64\libsybsspiwrapper64.dll
```

注意： 此库存储在 `%SYBASE%\OCS-15_0\lib3p64` 目录中。

SDK 15.7 中针对 Adaptive Server Enterprise 驱动程序和提供程序的功能

ESD #7 为 Adaptive Server ODBC 驱动程序 15.7 和 Ribo 实用程序引入了新功能。

Adaptive Server ODBC 驱动程序的新连接属性 CancelQueryOnFreeStmt

如果某 Microsoft Access 表单使用 Adaptive Server ODBC 驱动程序执行返回大结果集的查询，并且在整个结果集处理完毕之前该表单被关闭，则在 ODBC 驱动程序处理完整个结果集之前，Microsoft Access 将保持无响应状态。

在 15.7 ESD #7 版中，新连接属性 **CancelQueryOnFreeStmt** 解决了这一问题。 如果将此连接属性设置为 1，则无论何时关闭表单，Adaptive Server ODBC 驱动程序都会立即取消所有待执行的结果并将控制权交还 Microsoft Access 应用程序。 如果设置为 0（缺省值），Adaptive Server ODBC 驱动程序中的行为将没有变化。

新增的设置客户端连接属性的有效方法

在 15.7 ESD #7 版中，Adaptive Server ODBC 驱动程序开始支持使用 ODBC **SQLSetConnectAttr** API 高效设置客户端连接属性。 属性值集在 Adaptive Server **sysprocesses** 表中可见，有助于区分不同的客户端连接。

要在低于 15.7 ESD #7 的版本中设置这些属性，应用程序必须显式调用 **set** 语句以设置导致服务器上其它执行的相应属性。 使用 **SQLSetConnectAttr** API 时，驱动程序会推迟执行 **set** 语句，并将其附加到下一个将要执行的语句中。

注意： 由于调用 **SQLSetConnectAttr** API 后不会立即执行 **set** 语句，因此在执行下一语句前，此值集在 Adaptive Server 中不可见。

SQLSetConnectAttr 支持以下属性：

- **SQL_ATTR_CLIENT_NAME** - 使用命令集 *clientname<value>* 设置客户端名称。

- **SQL_ATTR_CLIENT_HOST_NAME** - 使用命令集 *clienthostname <value>* 设置客户端主机名。
- **SQL_ATTR_CLIENT_APPL_NAME** - 使用命令集 *clientapplname <value>* 设置客户端应用程序名称。

这些属性值被截断为 30 字节。 可使用 ODBC **SQLGetConnectAttr** 检索这些属性值。但是，在此接口外对服务器值所做的任何更改均不会反映出来。

对 Adaptive Server ODBC 驱动程序中 data-at-exec 功能的增强支持

在 Adaptive Server ODBC 驱动程序 15.7 ESD #7 版中，对 data-at-exec 功能进行了增强，该功能现可以支持批处理操作，从而降低了内存使用率、提升了应用程序性能。

在较低版本中，调用 **SQLBulkOperations** 或执行批处理前必须完全装载绑定参数的所有数据。 在 ESD #7 中，应用程序无需预装载任何参数数据，可使用 **SQLPutData** 以块的形式进行发送。 使用 Adaptive Server ODBC 驱动程序批处理协议时（**SQLExecute**/**SQLExecDirect** 和 **SQL_ATTR_BATCH_PARAMS**），只要 **SQL_ATTR_PARAMSET_SIZE** 设置为 1，便支持 data-at-exec。 服务器必须支持 LOB 参数，才能对 LOB 列使用 data-at-exec。

Ribo 实用程序中的新命令行选项 -n

Ribo 实用程序得到了增强，现可使用新命令行选项 `-n` 将原始 `.tds` 转储文件转换为便于管理的多个小文件。

在低于 15.7 ESD #7 的版本中，Ribo 实用程序将整个原始 `.tds` 转储文件转换为单个转换文件，不考虑转换文件的大小。 Ribo 实用程序得到了增强，现可使用新命令行选项 `-n` 将原始 `.tds` 转储文件转换为便于管理的多个小文件。 使用 `-n` 选项指定单个转换文件的最大大小（以 KB 为单位）。 当转换输出文件的大小超出通过 `-n` 选项指定的值时，将创建一个新文件。

输出文件名称遵循以下命名约定：

<output_file_part1_of_5> <output_file_part2_of_5>

其中 **<output_file>** 是用户指定的文件，其后附有 **partX_ofY**，其中 *X* 是当前部分，*Y* 是转换输出所分割的部分数。

注意： 执行转换时，`-n` 标志会生效。

用于 Python 的 Adaptive Server Enterprise 扩展模块

用于 Python 的 Adaptive Server Enterprise 扩展模块得到了增强，可支持数据源名称样式 (DSN-style) 连接属性、新示例程序以及 **blklib**。

支持 DSN 样式连接字符串属性

connect() 方法增加了对 DSN 样式连接属性的支持。

connect()
构建一个连接对象，用以代表与数据库的连接。

此方法接受以下关键字参数：

- **user** - 连接用来登录到服务器的用户登录名。
- **password** - 连接在登录到服务器时使用的口令。
- **servername** - 定义客户端程序连接到的 Adaptive Server 名称。 如果不指定 **servername**，则 DSQUERY 环境变量会定义 Adaptive Server 名称。
- **dsn** - 数据源名称。 该数据源名称是一个以分号分隔的字符串，其格式为“名称=值”：
  - 名称 - 该值不区分大小写，可由等号 (=) 或分号 (;) 分隔。 一个属性可以具有多个同义词。 例如，**server** 和 **servername** 表示同一个属性。
  - 等号 (=) - 表示开始将值分配给名称。 如果没有等号，则名称将被视为布尔类型，且值为 true。
  - 值 - 以分号 (;) 终止的字符串。 如果值中已经包含分号或反斜杠 (\)，请再使用一个反斜杠。 值的类型可以是布尔型、整数型或字符串型。 布尔类型的有效值为 true、false、on、off、1 和 0。

  注意： 如果存在没有值的布尔型名称，则必须将布尔类型设置为 true。

  例如：

```
sybpydb.connect(user='name', password='password string',
                          dsn=’servername=Sybase;timeout=10’)
```

有效的属性名称和值

下表列出了 **dsn** 关键字参数的有效属性名称和值。

名称	说明	值
ANSINull	确定在 SQL 的等于 (=) 或不等于 (!=) 比较计算中，NULL 值操作数的求值是否符合 ANSI 标准。 如果值为 true，Adaptive Server 将 ANSI 行为强制为 *= NULL* 和 *is NULL* 不等效。 在标准 Transact-SQL 中，*= NULL* 和 *is NULL* 被视为等效。 此选项还将以相类似的方式影响 *<> NULL* 和 *is not NULL* 行为。	布尔值。 缺省值为 false。
BulkLogin	确定是否已启用连接以执行批量复制操作。	布尔值。 缺省值为 false。
ChainXacts	如果值为 true，Adaptive Server 将采用链式事务行为，即每个服务器命令都将视为不同的事务。 Adaptive Server 将在以下任意语句之前隐式执行一个 begin transaction 命令： **delete**、**fetch**、**insert**、**open**、**select** 和 **update**。 仍必须显式结束事务或回退事务。 如果值为 false，则应用程序必须指定与 commit 或 rollback 语句成对出现的显式 begin transaction 语句。	布尔值。 缺省值为 false。
Charset	指定用于此连接的**字符集**。	字符串值。
Confidentiality	是否已对连接执行数据加密服务。	布尔值。 缺省值为 false。
CredentialDelegation	确定是否允许服务器使用用户的委托证书连接到另一个服务器。	布尔值。 缺省值为 false。
DetectReplay	确定连接的安全机制是否检测回放的传输。	布尔值。 缺省值为 false。
DetectOutOfSequence	确定连接的安全机制是否检测到达顺序混乱的传输。	布尔值。 缺省值为 false。
Integrity	确定连接的安全机制是否执行数据完整性检查。	布尔值。 缺省值为 false。
Interfaces	interfaces 文件的路径和名称。	字符串值。

名称	说明	值
Keytab	连接的安全机制会从一个文件中读取与 *username* 值相配的安全密钥，此属性表示该文件的名称和路径。	字符串值。 缺省值为 NULL，即用户必须在连接之前创建证书。
Locale	确定消息、数据类型转换以及日期时间格式所使用的语言和字符集。	字符串值。
Language	确定消息、数据类型转换以及日期时间格式所使用的语言集。	字符串值。
LoginTimeout	指定登录超时值。	整数值。
MaxConnect	指定某上下文可同时打开的最大连接数量。	整数值。 缺省值为 25。不允许负值和零值。
MutualAuthentication	确定服务器是否必须向客户端验证自身。	布尔值。 缺省值为 false。
NetworkAuthentication	确定连接的安全机制是否执行基于网络的用户验证。	布尔值。 缺省值为 false。
PacketSize	指定 TDS 包大小。	整数值。
Password	指定用于登录服务器的口令。	字符串值。
PasswordEncryption	确定连接是否需要使用非对称密钥对口令进行加密。	布尔值。 缺省值为 false。
SecurityMechanism	指定执行连接安全服务的网络安全机制名称。	字符串值。 缺省值取决于安全驱动程序配置。
Server **Servername**	指定连接到的服务器的名称。	字符串值。
ServerPrincipalName	指定已打开连接的服务器的网络安全主体名。	字符串值。 缺省值为 NULL，这意味着此连接假定服务器主体名与其 *ServerName* 值相同。
Keepalive	确定是否使用 KEEPALIVE 选项。	布尔值。 缺省值为 true。
Timeout	指定连接超时值。	整数值。

名称	说明	值
UID **User** **Username**	指定用于登录服务器的名称。	字符串值。

新示例程序

用于 Python 的 Adaptive Server Enterprise 扩展模块中增加了若干新示例。

dsnconnect
演示如何使用 **dsn** 连接到服务器。

blk
使用批量复制例程将数据复制到服务器表。 随即检索并显示该数据。

blkmany
使用批量复制例程同时复制数据和多个行。

blkiter
演示如何使用 Python 迭代协议批量拷出表行。

blktypes
演示如何将不同的 Python 对象类型（缺省值、NULL 值等）作为批量操作中的值来使用。

blklib 支持

blklib 功能是 Python DB-API 的扩展，用于批量复制行。 **blklib** 功能包括对象接口、方法和属性。

BulkCursor 对象构造方法

此 Python 扩展模块提供用于与数据库建立连接的连接对象。 该连接对象包括用于创建新 `BulkCursor` 对象的方法，此方法可对批量操作的上下文进行管理。

只有连接对象所对应的连接的批量操作属性已标记时，才能根据其来构造 `BulkCursor`。

用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10))")
blk = conn.blkcursor()
```

close()

`BulkCursor` 对象的 **close()** 方法可关闭批量操作。 一旦调用该方法，将无法使用批量游标对象。 **close()** 不使用任何参数。

用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
blk = conn.blkcursor()
bblk.close()
```

copy()

`BulkCursor` 对象的 **copy()** 方法可初始化批量操作。

该方法接受以下参数：

- **tablename** - 用于指定批量操作表名称的字符串。
- **direction** - 关键字参数，其值包括： *in* 和 *out*。

用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10))")
blk = conn.blkcursor()
blk.copy("mytable", direction="out")
```

done()

`BulkCursor` 对象的 **done()** 方法标记批量操作的完成情况。 若要开始另一操作，请调用 **copy()** 方法。

用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10)
blk = conn.blkcursor()
blk.copy("mytable", direction="in")
...
blk.done()
blk.copy("mytable", direction="out")
...
blk.done()
blk.close()
```

用于 PHP 的 Adaptive Server Enterprise 扩展模块

用于 PHP 的 Adaptive Server Enterprise 扩展模块得到了增强，现可支持 DSN 样式连接属性。

支持 DSN 样式连接属性

sybase_connect() 和 **sybase_pconnect()** API 支持 DSN 样式连接属性。

仅使用 **servername** 参数调用 **sybase_connect()** 或 **sybase_pconnect()** API 时，**servername** 必须包含有效的 DSN（数据源名称）字符串。 该数据源名称是一个以分号 (;) 分隔的字符串，其格式为“名称=值”，如下所示：

1. 名称 - 该值不区分大小写，可由等号 (=) 或分号 (;) 分隔。 一个属性可以具有多个同义词。 例如，**server** 和 **servername** 表示同一个属性。
2. 等号 (=) - 表示开始将值分配给名称。 如果没有等号，则名称将被视为布尔类型，且值为 true。
3. 值 - 以分号 (;) 终止的字符串。 如果值中已经包含分号或反斜杠 (\)，请再使用一个反斜杠。 值的类型可以是布尔型、整数型或字符串型。 布尔类型的有效值为 true、false、on、off、1 和 0。

注意： 如果存在没有值的布尔型名称，则必须将布尔类型设置为 true。

例如：

```
Username=name;Password=pwd;Timeout=10
```

有效的属性名称和值

下表列出了 **dsn** 关键字参数的有效属性名称和值。

名称	说明	值
ANSINull	确定在 SQL 的等于 (=) 或不等于 (!=) 比较计算中，NULL 值操作数的求值是否符合 ANSI 标准。 如果值为 true，Adaptive Server 将 ANSI 行为强制为 *= NULL* 和 *is NULL* 不等效。 在标准 Transact-SQL 中，*= NULL* 和 *is NULL* 被视为等效。 此选项还将以相类似的方式影响 *<> NULL* 和 *is not NULL* 行为。	布尔值。 缺省值为 false。
BulkLogin	确定是否已启用连接以执行批量复制操作。	布尔值。 缺省值为 false。

名称	说明	值
ChainXacts	如果值为 true，Adaptive Server 将采用链式事务行为，即每个服务器命令都将视为不同的事务。 Adaptive Server 将在以下任意语句之前隐式执行一个 begin transaction 命令： **delete**、**fetch**、**insert**、**open**、**select** 和 **update**。 仍必须显式结束事务或回退事务。 如果值为 false，则应用程序必须指定与 commit 或 rollback 语句成对出现的显式 begin transaction 语句。	布尔值。 缺省值为 false。
Charset	指定用于此连接的**字符集**。	字符串值。
Confidentiality	是否已对连接执行数据加密服务。	布尔值。 缺省值为 false。
CredentialDelegation	确定是否允许服务器使用用户的委托证书连接到另一个服务器。	布尔值。 缺省值为 false。
DetectReplay	确定连接的安全机制是否检测回放的传输。	布尔值。 缺省值为 false。
DetectOutOfSequence	确定连接的安全机制是否检测到达顺序混乱的传输。	布尔值。 缺省值为 false。
Integrity	确定连接的安全机制是否执行数据完整性检查。	布尔值。 缺省值为 false。
Interfaces	interfaces 文件的路径和名称。	字符串值。
Keytab	连接的安全机制会从一个文件中读取与 *username* 值相配的安全密钥，此属性表示该文件的名称和路径。	字符串值。 缺省值为 NULL，即用户必须在连接之前创建证书。
Locale	确定消息、数据类型转换以及日期时间格式所使用的语言和字符集。	字符串值。
Language	确定消息、数据类型转换以及日期时间格式所使用的语言集。	字符串值。
LoginTimeout	指定登录超时值。	整数值。
MaxConnect	指定某上下文可同时打开的最大连接数量。	整数值。 缺省值为 25。 不允许负值和零值。

名称	说明	值
MutualAuthentication	确定服务器是否必须向客户端验证自身。	布尔值。 缺省值为 false。
NetworkAuthentication	确定连接的安全机制是否执行基于网络的用户验证。	布尔值。 缺省值为 false。
PacketSize	指定 TDS 包大小。	整数值。
Password	指定用于登录服务器的口令。	字符串值。
PasswordEncryption	确定连接是否使用非对称口令加密。	布尔值。 缺省值为 false。
SecurityMechanism	指定执行连接安全服务的网络安全机制名称。	字符串值。 缺省值取决于安全驱动程序配置。
Server **Servername**	指定连接到的服务器的名称。	字符串值。
ServerPrincipalName	指定已打开连接的服务器的网络安全主体名。	字符串值。 缺省值为 NULL，这意味着此连接假定服务器主体名与其 *ServerName* 值相同。
Keepalive	确定是否使用 KEEPALIVE 选项。	布尔值。 缺省值为 true。
Timeout	指定连接超时值。	整数值。
UID **User** **Username**	指定用于登录服务器的用户名称。	字符串值。

dsnconnect.php 示例程序

`dsnconnect.php` 示例程序使用 DSN 连接字符串连接到服务器。 该示例程序还可以输出服务器名称、用户帐户和当前数据库。

ESD #6 的新增功能

ESD #6 为 Open Client 15.7 和 Open Server 15.7 更新了功能，为用于 Python 15.7 的 Adaptive Server Enterprise 扩展模块和用于 Perl 15.7 的 Adaptive Server Enterprise 扩展模块引入了数据源名称 (DSN) 连接属性支持。

Open Client 15.7 和 Open Server 15.7 功能

Open Client 15.7 和 Open Server 15.7 得到了增强，现支持 LOB 数据类型批量拷入、新的环境变量 SYBOCS_IFILE、LDAP 和 SSL 版本、参数格式抑制、扩展增强加密口令和 BCP --quoted-fname 选项。

对 LOB 数据类型执行批量拷入

在 ESD #6 中，可在使用 **blk_textxfer()** API 调用后使用 **blk_rowxfer()**。

在以前的版本中，如果将某 LOB 列标记为通过 **blk_textxfer()** API 进行传输以将 LOB 数据复制到由行内值和行外值构成的数据库表中，则该数据类型的后续所有列也必须标记为使用 **blk_textxfer()** API 进行传输，而不能使用 **blk_rowxfer()**。 ESD#6 中消除了此限制，现在可在使用 **blk_textxfer()** API 调用后使用 **blk_rowxfer()**。

新环境变量 SYBOCS_IFILE

可使用 SYBOCS_IFILE 指定 interfaces 文件的位置，而不必使用缺省路径 `$SYBASE/interfaces`。

如果应用程序在 CT-Library 中设置 CS_IFILE 属性，则该属性设置将优先执行。

LDAP 和 SSL 版本支持

Sybase 提供的 OpenLDAP 库 (`libsybaseldap.so/dll`) 使用 OpenLDAP 2.4.31 版和 OpenSSL 1.0.1b 版与 LDAP 服务器建立连接。

参数格式抑制

现在，Open Client 和 Open Server 支持对 Adaptive Server Enterprise 中的动态语句进行参数格式抑制。

注意： 从 ESD #3 开始，Open Client 即已支持参数格式抑制。 而 Open Server 对参数格式抑制的支持直到 ESD #6 才被引入。

Open Server 对扩展增强加密口令的支持

如果客户端连接支持扩展增强加密口令 (EPEP)，Open Server 将处理包括口令解密在内的登录协商。

登录协商发生在调用 SRV_CONNECT 处理程序前。 在 SRV_CONNECT 事件处理程序中，应用程序只能使用现有 SRV_T_PWD 属性检索口令，要检查所用的口令加密协议，则需要使用新属性。

要尝试使用 Open Server 口令加密，可使用带有 `-x` 选项的 **isql** 连接到“lang”示例，这将打开 **isql** 中的口令加密。

注意： 从 15.0 版本起，Open Client 即已开始支持强登录口令加密。 而 Open Server 自 ESD#6 起才开始支持强登录口令加密。

SRV_T_PWD
此属性与 **srv_thread_props()** 一起用于检索口令。 如果客户端支持 EPEP 协议，SRV_T_PWD 将自动返回解密口令。

SRV_PWD_ENCRYPT_VERSION
Open Server 中新增的公共枚举类型具有以下值：

- SRV_NOENCRYPT_PWD (0)
- SRV_ENCRYPT_PWD (1)（未在 Open Server 中实现）
- SRV_EXTENDED_ENCRYPT_PWD (2)（未在 Open Server 中实现）
- SRV_EXTENDED_PLUS_ENCRYPT_PWD (3)

SRV_T_PWD_ENCRYPT_VERSION
将此新增的只读属性与 **srv_thread_props()** 函数结合使用，对检索口令的口令加密协议版本进行检索。 SRV_PWD_ENCRYPT_VERSION 中描述了此属性的类型和可能的值。

注意： 使用此属性时无法避免以明文方式传输口令。 Open Server 读取客户端支持的口令加密版本时，口令可能已经以明文形式进行传输。 但是，可使用此属性验证是否所有客户端应用程序均使用所需的口令加密算法。

SRV_S_DISABLE_ENCRYPT
可使用 SRV_S_DISABLE_ENCRYPT 属性禁用对本机口令协商的支持。 如果设置此属性，Open Server 将不会启动口令协商协议。 SRV_S_DISABLE_ENCRYPT 的缺省值为 CS_FALSE。

BCP --quoted-fname 选项

BCP 命令行参数的当前语法为“--quoted-fname”。

本系统接受字符串之间没有空格的字符串“quoted-fname”。 可将新参数放置在命令行参数列表中数据文件后的任意位置。

若要使用包含特殊字符的数据文件名称，除使用此选项外，还应将文件名称用双引号括起来，并在每个文件前放置一个反斜线 (\")。 如果文件名称包含双引号，则在每个文件名称的双引号前放置一个反斜线。

表 4. 示例

数据文件名称	更新的语法
fnamepart1,fnamepart2	\” fnamepart1,fnamepart2\”
fnamepart1”fnamepart2	\” fnamepart1\”fnamepart2\”
“fnamepart1”fnamepart2”	\”\” fnamepart1\”fnamepart2\”\”

用于 Python 的 Adaptive Server Enterprise 扩展模块

用于 Python 的 Adaptive Server Enterprise 扩展模块得到了增强，现可支持 DSN 样式连接属性。

支持 DSN 样式连接属性

connect() 方法接受名为 **dsn** 的新关键字参数。

该关键字参数是指定连接信息的字符串。 **dsn** 字符串的语法为：

```
name1=value1;name2=value2;...
```

此处的 name1 通常指连接属性或选项。

名称字符串不包含转义字符。 若要在值字符串中显示等号和分号，应通过在每个字符串前放置反斜线来转义字符。

用于 Perl 的 Adaptive Server Enterprise 扩展模块

用于 Perl 的 Adaptive Server Enterprise 扩展模块得到了增强，现可支持新属性和方法、新的 Perl 数据库和语句句柄属性、多个语句、动态 SQL、绑定参数、存储过程、私有驱动程序方法、文本和图像数据处理以及错误处理。

支持 DSN 样式连接属性

驱动程序使用 DSN 机制以允许在连接时设置某些属性。

DSN 属性的语法与 Open Source DBD::Sybase 驱动程序的语法相同。 因此，无需更改 Perl 脚本或维护不同版本的 DBD::Sybase 和 DBD::SybaseASE。 但是，DBD::SybaseASE 不支持某些被认为已过时的属性。 请参见“当前不支持的 DSN 语法”。

SybaseASE 驱动程序连接语法
dbi:SybaseASE: 部分用于获取驱动程序的包名称，以便其可以按以下语法进行装载。

```
DBI->connect("dbi:SybaseASE:attr=value;attr=value", $user_id,
$password, %attrib);
```

当 DSN 传递到驱动程序时，系统将删除此部分，而剩余字符串将保留要分解的键和值对。

注意： *$user_id* 和 *$password* 是单独的 API 参数；它们不是 DSN 字符串的一部分。

%attrib 参数是可选的以逗号分隔的键值对的字串，用于在连接时设置选项。 将在 **connect()** 调用期间将其传递到驱动程序并进行处理。 例如：

```
DBI->connect("dbi:SybaseASE:server=mumbles; user, password,
PrintError => 1, AutoCommit = 0);
```

属性和方法

当连接到服务器时，以下属性当前受到支持。

属性	说明
server	指定要连接到的服务器。 驱动程序当前假定已设置此选项。 如果未指定服务器，则使用 ENV{"DSQUERY"} 机制获取服务器名称。
database	指定服务器内在连接时充当目标数据库的数据库。 如果未指定数据库，则使用 master 数据库。
hostname	指定此进程的 `sysprocesses` 表的值部分中存储的主机名。 如果未指定主机名，则使用执行 Perl 应用程序的主机。
language	指定用于此连接的区域设置。 如果未指定语言，将使用名为 CS_LC_ALL 的内部缺省区域设置。
charset	指定用于此连接的字符集。 如果未指定字符集，将使用内部缺省值 **utf8**。
host; port	指定要使用的主机和端口组合，而不是依靠 interfaces 文件条目。 **注意：** 在 Perl DSN 语法中，host 和 port 是不同的选项。 当前不支持如下备用 DSN 形式： `host:port=mumbles:1234` 如果由于不想使用 interfaces 文件而指定 DSN 选项 host 和 port，则该 host 和 port 必须能够满足连接条件。 如果除 host 和 port 组合外还提供了 DSN 属性“server=”，则连接将失败。 因此，要么使用主机和端口建立连接，要么使用服务器建立连接。 两个 DSN 属性（server 和 host/port）互相排斥。
timeout	指定连接超时值。 设置为 0 或负值表示无超时值。
loginTimeout	指定登录超时值（秒）。 缺省值是 60 秒。 设置 **loginTimeout=*value in seconds*** 以启用此属性。
tds_keepalive	指定用于连接的 KEEP_ALIVE 属性。 设置 **tds_keepalive=*1*** 以启用此属性。

属性	说明
packetSize	指定用于连接的 TDS 包大小。 缺省情况下，驱动程序中设置的下限为 2048. 最大值由服务器确定，不在驱动程序中设置。
maxConnect	增加或减少允许的连接数。 值范围为 1 到 128；缺省值为 25。
encryptPassword	指定是否使用口令加密。 设置 **encryptPassword=*1*** 以启用此属性。
sslCAFile	指定 `trusted.txt` 文件的替代位置。 指定最多为 256 个字符的绝对路径。
scriptName	指定驱动应用程序的顶级 Perl 脚本的选定名称。 此名称将作为应用程序名称显示在 `sysprocesses` 表中。 如果不指定该值，将使用从 Perl 内部环境获得的缺省应用程序名称。 此值最多可包含 256 个字符。 **注意：** 送入 SybaseASE 驱动程序的应用程序名称或通过 DSN **scriptName** 选项进行设置，或派生自 Perl 内部环境。
interfaces	指定 Sybase interfaces 文件的替代位置。 **sslCAFile** 和 **scriptName** 选项也具有同样约束。

属性值可以反复使用，只要驱动程序能识别即可。 非法属性可导致 **DBI->connect()** 调用失败。

注意： 属性名称跟在 Open Source Sybase Perl 驱动程序后面。

特定于 DSN 的示例：

```
$dbh = DBI->connect("dbi:SybaseASE:server=mumbles", $user, $passwd);
```

或者使用 DSQUERY 环境变量：

```
my $srv = $ENV{"DSQUERY"};
$dbh = DBI->connect("dbi:SybaseASE:server=$srv",  $user, $passwd);
$dbh = DBI-
>connect("dbi:SybaseASE:host=tzedek.sybase.com;port=8100", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:maxConnect=100", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:database=sybsystemprocs", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:charset=iso_1", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:language=us_english", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:packetSize=8192", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:interfaces=/opt/sybase/
interfaces", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:loginTimeout=240", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:timeout=240", $user, $passwd);
$dbh = DBI->connect("dbi:Sybase:scriptName=myScript", $user,
$password);
$dbh = DBI->connect("dbi:SybaseASE:hostname=pedigree", $user,
```

```
$password);
$dbh = DBI->connect("dbi:SybaseASE:encryptPassword=1", $user,
$password);
$dbh = DBI>connect("dbi:SybaseASE:sslCAFile=/usr/local/sybase/
trusted.txt", $user, $password,
AutoCommit => 1);
```

特定于 DSN 的示例组合：

```
$dbh = DBI->connect("dbi:SybaseASE:server=mumbles,
database=tempdb;packetSize=8192;
language=us_english;charset=iso_1;encryptPassword=1", $user, $pwd,
AutoCommit=>1, PrintError => 0);
```

当前不支持的 DSN 语法
当前不支持以下 DSN 语法：

- **tdsLevel**
- **kerberos**；例如：

  ```
  $dbh = DBI->connect("dbi:SybaseASE:kerberos=$serverprincipal",
  '', '');
  ```
- **bulkLogin**；例如：

  ```
  $dbh = DBI->connect("dbi:SybaseASE:bulkLogin=1", $user,
  $password);
  ```
- **serverType**

当前支持的数据库句柄属性

下表列出了当前支持的数据库句柄属性。

属性	说明	缺省值
dbh->{AutoCommit} = (0|1);	禁用或启用 AutoCommit。	0（关闭）
dbh->{LongTruncOK} = (0|1);	禁用或启用 `text` 和 `image` 类型的截断。	0
dbh->{LongReadLen}=(int);	设置 `text` 和 `image` 数据的缺省读取块大小。 例如： **dbh->{LongReadLen} = 64000**。	32767
dbh->{syb_show_sql} =(0|1);	设置后，当前语句会包含在 **$dbh->errstr** 机制返回的错误字符串中。	0
dbh->{syb_show_eed} = (0|1);	设置后，扩展错误信息会包含在 **$dbh->errstr** 返回的错误字符串中。	0

属性	说明	缺省值
dbh->{syb_chained_txn} = (0|1);	设置后，AutoCommit 关闭时将使用 CHAINED 事务。 仅在调用 **connect()** 时使用此属性： `$dbh = DBI->connect("dbi:SybaseASE:", $user, $pwd, {syb_chained_txn => 1});` 在关闭 AutoCommit 的情况下，任何时候使用 **syb_chained_txn** 都将强制在当前句柄上执行提交。 设置为 0 时，会根据需要发出显式 **BEGIN TRAN**。	0
dbh->{syb_use_bin_0x} = (0|1);	设置后，结果字符串中的 `BINARY` 和 `VARBINARY` 值会带有“0x”前缀。	0
dbh->{syb_binary_images} = (0|1);	设置后，将以原始二进制格式返回 `image` 数据。 否则，`image` 数据会转换为十六进制字符串。	0
dbh->{syb_quoted_identifier} =(0|1);	在使用“标识符”进行引用时，允许使用与 Sybase 保留字冲突的标识符。	0
dbh->{syb_row-count}=(int);	如果设置为非零值，则 **SELECT** 返回的行数，或受 **UPDATE** 或 **DELETE** 语句影响的行数将限制为 *rowcount* 值。 将其重置为 0 会清除此限制。	0
dbh->{syb_flush_finish} = (0|1);	设置后，驱动程序会通过实际读取当前命令的所有剩余结果来清除它们。 可使用此设置替代驱动程序发出的 **ct_cancel()** 命令。	0
dbh->{syb_date_fmt} = datefmt string	此私有方法设置缺省日期转换和显示格式。 请参见“缺省日期转换和显示格式”。	
dbh->{syb_err_handler}	可以创建该 Perl 子例程以在进行常规错误处理前执行错误处理程序或进行报告。 对某类警告很有用。 请参见“错误处理”。	0（不存在）
dbh->{syb_failed_db_fatal} = (0|1)	如果 DSN 具有 **database=*mumbles*** 属性/值对，并且此数据库在连接时不存在，则 **DBI->connect()** 调用将失败。	0
dbh->{syb_no_child_con} =(0|1);	设置后，驱动程序将不允许在 **dbh** 上存在多个活动语句句柄。 在这种情况下，可以预准备某个语句，但必须执行完该语句才能尝试准备另一语句。	0
dbh->{syb_cancel_request_on_error}= (0|1);	设置后，当执行多个语句集且其中一个语句失败时，**sth->execute()** 也将失败。	1（打开）
dbh->{syb_bind_empty_string_as_null}= (0|1);	设置后，NULLABLE 列属性将返回一个空字符串（一个空格）用来表示 NULL 字符。	0

属性	说明	缺省值
dbh->{syb_disconnect_in_child} = (0|1);	处理分叉上已关闭的连接。 如果子连接已断开，则 DBI 将导致连接关闭。	0
dbh->{syb_enable_utf8} = (0|1);	设置后，UNICHAR、UNIVARCHAR 和 UNITEXT 将转换为 **utf8**。	0
sth->syb_more_results} = (0|1);	请参见“多个结果集”。	
sth->{syb_result_type} = (0|1);	设置后，将返回数值结果编号，而不是符号式 CS_ version。	0
sth->{syb_no_bind_blob} = (0|1);	设置后，在执行 **sth->{fetch}** 或其它变化形式时将不会返回 image 或 text 列。 请参见“文本和图像数据处理”。	0
sth->{syb_do_proc_status} = (0|1);	强制 **$sth->execute()** 读取在 SQL 流中执行的存储过程的返回状态。 如果返回状态非零，则 **$sth->execute()** 会返回 undef（即失败）。 设置此属性不会影响现有语句句柄。 但是，将会对设置后创建的语句句柄产生影响。 要恢复现有 **$sth** 句柄的行为，请执行： **$sth->{syb_do_proc_status} = 0**;	0

不支持的数据库句柄选项

不支持以下数据库句柄选项。

- **dbh->{syb_dynamic_supported}**
- **dbh->{syb_ocs_version}**
- **dbh->{syb_server_version}**
- **dbh->{syb_server_version_string}**
- **dbh->{syb_has_blk}**

注意： 如果 Perl 脚本尝试使用这些选项，将生成错误。

Perl 支持的数据类型

Perl 驱动程序当前支持字符串、数值以及日期和时间数据类型。

字符串类型	数值类型	日期和时间数据类型
char	integer	datetime
varchar	smallint	date
binary	tinyint	time
varbinary	money	bigtime
text	smallmoney	bigdatetime
image	float	
unichar	real	
univarchar	double	
	numeric	
	decimal	
	bit	
	bigint	

注意： Perl 将 numeric 和 decimal 类型以字符串形式返回。 其它数据类型按各自的格式返回。

Sybase ASE 驱动程序使用的缺省时间/日期格式是简写格式，例如，Aug 7 2011 03:05PM。

此格式基于 C（缺省）区域设置。 有关支持的其它日期和时间格式，请参见“缺省日期转换和显示格式”。

使用多个语句

Adaptive Server 可在单个批处理中处理多语句 SQL。

例如：

```
my $sth = $dbh->prepare("
    insert into publishers (col1, col2, col3) values (10, 12, 14)
    insert into publishers (col1, col2, col3) values (1, 2, 4)
    insert into publishers (col1, col2, col3) values (11, 13, 15)
    ");
my $rc = $sth->execute();
```

如果这些语句中的任何语句失败，则 **sth->execute()** 将返回 `undef`。 如果 **AutoCommit** 处于启用状态，则成功完成的语句可能已在表中插入数据，这可能并不是您所期望的结果。

多个结果集

Perl 驱动程序允许您通过一次调用准备多个语句，并通过另外一次调用执行这些语句。例如，执行包含多个 select 的存储过程将返回多个结果集。

通过一次调用准备的多个语句的结果将作为单个数据流返回客户端。 将每个不同的结果集视为常规的单个结果集，这意味着语句句柄的 **fetch()** 方法将在每个结果集结束时返回 `undef`。

CT-Lib API **ct_fetch()** 返回 `CS_END_RESULTS`，检索完最后几行后，驱动程序会将其转换为 `undef`。

此驱动程序允许应用程序通过检查 **sth->{syb_result_type}** 获取结果类型。 然后便可使用 **sth->{syb_more_results}** 语句句柄属性确定是否存在其它要返回的结果集。 由 **sth->{syb_results_type}** 返回的（数字）值为以下内容之一：

- `CS_MSG_RESULT`
- `CS_PARAM_RESULT`
- `CS_STATUS_RESULT`
- `CS_COMPUTE_RESULT`
- `CS_ROW_RESULT`

多个结果集示例：

```
do {
    while($a = $sth->fetch) {
    ..for example, display data..
    }
} while($sth->{syb_more_results});
```

如果您需要多个结果集，Sybase 建议您使用此功能。

注意： Perl 驱动程序当前不支持使用 **ct_cursor()** API 的游标。 因此，驱动程序不会报告 CS_CURSOR_RESULT。

在 DatabaseHandle (dbh) 上使用多个活动语句

如果某个数据库上已有一个活动语句句柄，则可通过在 **$dbh->prepare()** 方法中打开新连接来允许在 **$dbh** 上存在多个活动语句句柄。

dbh->{syb_no_child_con} 属性控制此功能的状态为打开还是关闭。 缺省情况下，DatabaseHandle 设置为 off，这表示支持多个语句句柄。 如果设置为 on，则会禁用此功能，即不允许同一数据库句柄上存在多个语句。

注意： 如果 AutoCommit 设置为 off，则不支持单个 **$dbh** 上存在多个语句句柄。 这样便可避免潜在的死锁问题。 此外，同时使用多个语句句柄无法提供事务完整性，因为需要使用不同的物理连接。

支持的字符长度

不同类型标识符支持的字符长度。

Sybase 标识符（如表和列）的名称可以在长度上超过 255 个字符。

受 TDS 协议限制的登录名、应用程序名和口令的长度不能超过 30 个字符。

配置区域设置和字符集

可使用 DSN 属性 **charset** 和 **language** 配置 CT-Library 区域设置和字符集的 Perl 驱动程序。

驱动程序的缺省字符集为 *UTF8*，缺省区域设置为 *CS_LC_ALL*。

动态 SQL 支持、占位符和绑定参数

Perl 驱动程序支持动态 SQL（包括参数用法）。

例如：

```
$sth = $dbh->prepare("select * from employee where empno = ?");

# Retrieve rows from employee where empno = 1024:
$sth->execute(1024);
while($data = $sth->fetch) {
    print "@$data\n";
}
# Now get rows where empno = 2000:
$sth->execute(2000);
while($data = $sth->fetch) {
    print "@$data\n";
}
```

注意： Perl 驱动程序支持“?”样式的参数，但不支持“:1”占位符类型的参数。 无法使用占位符绑定 `text` 或 `image` 数据类型。

DBD::SybaseASE 针对 **prepare()** 方法使用 API 的 Open Client **ct_dynamic()** 系列。 有关“?”样式占位符约束和常规动态 SQL 用法的信息，请参见《Sybase Open Client C 程序员指南》。

以下为展示动态 SQL 支持的另一个示例：

```
my $rc;
my $dbh;
my $sth;


# call do() method to execute a SQL statement.
#
$rc = $dbh->do("create table tt(string1 varchar(20), date datetime,
    val1 float, val2 numeric(7,2))");
```

```
$sth = $dbh->prepare("insert tt values(?, ?, ?, ?)");
$rc = $sth->execute("test12", "Jan 3 2012", 123.4, 222.33);


# alternate way, call bind_param() then execute without values in the
# execute statement.
$rc = $sth->bind_param(1, "another test");
$rc = $sth->bind_param(2, "Jan 25 2012");
$rc = $sth->bind_param(3, 444512.4);
$rc = $sth->bind_param(4, 2);
$rc = $sth->execute();

# and another execute, with args.....
$rc = $sth->execute("test", "Feb 30 2012", 123.4, 222.3334);
```

注意： 由于日期无效，最后一条语句抛出扩展错误信息 (EED)。 在 Perl 脚本中，在执行前设置 **dbh->{syb_show_eed} =*1*** 以将 Adaptive Server 错误消息写入 **dbh->errstr**。

以下为演示“?”样式占位符的另一个示例：

```
$sth = $dbh->prepare("select * from tt where date > ? and val1 > ?");
$rc = $sth->execute('Jan 1 2012', 120);

# go home....
$dbh->disconnect;
exit(0);
```

存储过程对占位符的支持

用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序支持带有输入和输出参数的存储过程。

存储过程的处理方式与其它任何 Transact-SQL 语句的处理方式相同。 但是，Sybase 存储过程会返回一个额外的结果集，其中包括与存储过程代码中的返回语句对应的返回状态。 此名为 CS_STATUS_RESULT、数值为 4043 的额外结果集是单独一行且始终最后返回。

驱动程序可使用特殊属性 **$sth->{syb_do_proc_status}** 处理存储过程。 如果设置了此属性，驱动程序将处理这一额外结果集，并将返回状态值放置在 **$sth->{syb_proc_status}** 中。 如果结果集的值不是 0，则将生成一条错误。

示例

```
$sth = $dbh->prepare("exec my_proc \@p1 = ?, \@p2 = ?");
    $sth->execute('one', 'two');
```

以下示例说明了位置参数的用法：

```
$sth = $dbh->prepare("exec my_proc ?, ?");
  $sth->execute('one', 'two');
```

不能在同一准备语句中混合使用位置参数和命名参数；例如，下列语句在第一个参数处会失败：

```
$sth = $dbh->prepare("exec my_proc \@p1 = 1, \@p2 = ?");
```

如果存储过程使用输出参数返回数据，则必须先对其进行声明：

```
$sth = $dbh->prepare(qq[declare @name varchar(50) exec getname abcd,
@name output]);
```

不能调用带有绑定参数的存储过程，如下所示：

```
$sth = $dbh->prepare("exec my_proc ?");
  $sth->execute('foo');
```

工作方式如下：

```
$sth = $dbh->prepare("exec my_proc 'foo'");
  $sth->execute('foo');
```

由于存储过程通常会返回多个结果集，因此请使用循环，直到 syb_more_results 为 0 为止：

```
do {
    while($data = $sth->fetch) {
        do something useful...
    }
 } while($sth->{syb_more_results});
```

参数示例

```
declare @id_value int, @id_name char(10)
    exec my_proc @name = 'a_string', @number = 1234,
        @id = @id_value OUTPUT, @out_name = @id_name OUTPUT
```

如果存储过程仅返回 OUTPUT 参数，可使用：

```
$sth = $dbh->prepare('select * .......');
    $sth->execute();
@results = $sth->syb_output_params();  #  this method is available in
SybaseASE.pm
```

这将为过程调用中的所有 OUTPUT 参数返回一个数组，并忽略其它任何结果。 如果不存在 OUTPUT 参数或存储过程失败，则不会定义此数组。

通用示例

```
$sth = $dbh->prepare("declare \@id_value int, \@id_name
        OUTPUT, @out_name = @id_name OUTPUT");
    $sth->execute();
    {
        while($d = $sth->fetch) {
            # 4042 is CS_PARAMS_RESULT
        if ($sth->{syb_result_type} == 4042) {
            $id_value  = $d->[0];
            $id_name  = $d->[1];
        }
    }
    redo if $sth->{syb_more_results};
}
```

OUTPUT 参数将作为特殊结果集中的单独一行返回。

参数类型

驱动程序不会尝试确定每个参数的正确参数类型。 所有参数的缺省值都缺省为 ODBC 样式 SQL_CHAR 值，除非使用类型值设置为支持的绑定类型的 **bind_param()**。

驱动程序支持以下 ODBC 样式的绑定类型：

- SQL_CHAR
- SQL_VARCHAR
- SQL_VARBINARY
- SQL_LONGVARCHAR
- SQL_LONGVARBINARY
- SQL_BINARY
- SQL_DATETIME
- SQL_DATE
- SQL_TIME
- SQL_TIMESTAMP
- SQL_BIT
- SQL_TINYINT
- SQL_SMALLINT
- SQL_INTEGER
- SQL_REAL
- SQL_FLOAT
- SQL_DECIMAL
- SQL_NUMERIC
- SQL_BIGINT
- SQL_WCHAR
- SQL_WLONGVARCHAR

ODBC 类型在驱动程序中映射为等效的 Adaptive Server 数据类型。 请参见《Sybase Adaptive Server Enterprise ODBC 驱动程序用户指南 15.7》。

执行存储过程 **sp_datatype_info** 来获取特定 Adaptive Server 支持类型的完整列表。 例如：

```
$sth = $dbh->prepare("exec my_proc \@p1 = ?, \@p2 = ?");
    $sth->bind_param(1, 'one', SQL_CHAR);
    $sth->bind_param(2, 2.34, SQL_FLOAT);
    $sth->execute;
    ....
    $sth->execute('two', 3.456);
    etc...
```

注意： 一旦为参数设置了列类型，除非释放并重试语句句柄，否则将无法对其进行更改。 绑定 SQL_NUMERIC 或 SQL_DECIMAL 数据时，如果标度或精度超过目标参数定义的大小，则可能会发生致命的转换错误。

以下面的存储过程定义为例：

```
declare proc my_proc @p1 numeric(5,2) as...
    $sth = $dbh->prepare("exec my_proc \@p1 = ?");
    $sth->bind_param(1, 3.456, SQL_NUMERIC);
```

将生成以下错误：

```
DBD::SybaseASE::st 执行失败： 服务器消息号=241 严重级=16 状态=2 行=0 过程=my_proc 文本=在 NUMERIC 值 '3.456' 到 NUMERIC 域的隐式转换期间出现标度错误。
```

如下所示设置 **arithabort** 选项以忽略这些错误：

```
$dbh->do("set arithabort off");
```

请参见 Adaptive Server 参考文档。

支持的私有驱动程序方法

dbh->syb_isdead() 返回 true 或 false 来表示连接状态。 返回值 false 可表示特定类或连接上存在错误，或表示该连接已失败。

$sth->syb_describe() 返回一个包含当前结果集每个输出列的描述的数组。 数组的每个元素都是对描述列的散列的引用。

可设置 NAME、TYPE、SYBTYPE、SYBMAXLENGTH、MAXLENGTH、SCALE、PRECISION 和 STATUS 等说明字段，如下所示：

```
$sth = $dbh->prepare("select name, uid from sysusers");
    $sth->execute;
    my @description = $sth->syb_describe;
    print "$description[0]->{NAME}\n";                  # prints name
    print "$description[0]->{MAXLENGTH}\n";     # prints 30
    etc, etc.
     ....
    while(my $row = $sth->fetch) {
    ....
}
```

注意： STATUS 字段是可对以下值进行测试的字符串： CS_CANBENULL、CS_HIDDEN、CS_IDENTITY、CS_KEY、CS_VERSION_KEY、CS_TIMESTAMP、CS_UPDATABLE、CS_UPDATECOL 和 CS_RETURN。

请参见 Open Client 文档。

缺省日期转换和显示格式

可以使用 **syb_data_fmt()** 私有方法设置您自己的缺省日期转换和显示格式。

Sybase 日期格式取决于客户端的区域设置。 缺省日期格式基于 'C' 区域设置，例如 Feb 16 2012 12:07PM。

此缺省区域设置还支持下列其它几种输入格式：

- 2/16/2012 12:07PM
- 2012/02/16 12:07
- 2012-02-16 12:07
- 20120216 12:07

使用 **dbh->{syb_date_fmt}** 并以字符串作为参数来更改日期的输入和输出格式。

表 5. 支持的日期/时间格式

日期格式	示例
LONG	Nov 15 2011 11:30:11:496AM
SHORT	Nov 15 2011 11:30AM
DMY4_YYYY	Nov 15 2011
MDY1_YYYY	11/15/2011
DMY1_YYYY	15/11/2011
DMY2_YYYY	15.11.2011
DMY3_YYYY	15-11-2011
DMY4_YYYY	15 November 2011
HMS	11:30:11
LONGMS	Nov 15 2011 11:30:33.532315PM

用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序支持 15.7 及之前版本所支持的所有日期和时间值。

文本和图像数据处理

用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序支持 image 和 `text` 类型的 LONG/BLOB 数据。 每种类型最多可容纳 2GB 的二进制数据。

文本/图像数据的缺省大小限制为 32KB。 此限制通过调用 **fetch()** API 进行设置，可使用 **LongReadLen** 属性进行更改。

不能使用绑定参数插入文本或图像数据。

使用常规 SQL 时，图像数据通常转换为十六进制字符串，但可使用 **syb_binary_images** 句柄属性来更改此行为。 或者，也可以使用类似 **$binary = pack("H*", $hex_string);** 的 Perl 函数执行转换。

由于 DBI 不支持通过 API 处理 BLOB 样式 (`text`/`image`) 类型，`SybaseASE.pm` 文件中提供了一组函数，可安装并在应用程序级别的 Perl 代码中使用来执行 Open Client **ct_get_data()** 样式调用。 **syb_ct_get_data()** 和 **syb_ct_send_data()** 调用是 Open Client 函数的包装，该函数可向/从 Adaptive Server 传输 `text` 和 `image` 数据。

示例

```
$sth->syb_ct_get_data($col, $dataref, $numbytes);
```

可使用 **syb_ct_get_data()** 调用按原始格式读取图像/文本数据，既可一次性全部读取也可分块读取。 要启用此调用，请将 **dbh->{syb_no_bind_blob}** 语句句柄设置为 *1*。

syb_ct_get_data() 调用带有以下参数：查询的列编号（从 1 开始）、标量参考以及字节数。 字节数 0 表示将读取尽可能多的字节。 图像/文本列必须位于 select 列表的末尾，才可以使用此调用。

调用顺序如下：

```
$sth = $dbh->prepare("select id, img from a_table where id = 1");
    $sth->{syb_no_bind_blob} = 1;
    $sth->execute;
    while($d = $sth->fetchrow_arrayref) {
        # The data is in the second column
        $len = $sth->syb_ct_get_data(2, \$img, 0);
}
```

syb_ct_get_data() 返回已读取的字节数，如果要分块读取数据，可使用：

```
while(1) {
$len = $sth->syb_ct_get_data(2, $imgchunk, 1024);
... do something with the $imgchunk ...
 last if $len != 1024;
}
```

其它 TEXT/IMAGE API

syb_ct_data_info() API 用于读取或更新您要更新的图像/文本数据项的 CS_IODESC 结构。

例如：

```
$stat = syb_ct_data_info($action, $column, $attr)
```

- *$action* - CS_SET 或 CS_GET。
- *$column* - 活动 select 语句的列号（CS_SET 操作忽略此值）。
- *$attr* - 在此结构中设置值的散列引用。

必须首先调用带有 CS_GET 的 **syb_ct_data_info()** 来读取想要更新的图像/文本数据项的 CS_IODESC 结构。 然后将 **total_txtlen** 结构元素的值更新为将要插入的图像/文本数据的长度（字节）。 将 **log_on_update** 设置为 true 以启用操作的完全日志记录。

如果正在读取 CS_IODESC 的图像/文本数据为 NULL，则调用带有 CS_GET 的 **syb_ct_data_info()** 将失败。 检索 CS_IODESC 条目前，使用标准 SQL 将 NULL 值更新为非 NULL 值（例如，空字符串）。

在此示例中，考虑更新 id 列为 1 的图像列中的数据：

1. 为数据查找 CS_IODESC 数据：

   ```
   $sth = $dbh->prepare("select img from imgtable where id = 1");
           $sth->execute;
   ```

```
    while($sth->fetch) {    # don't care about the data!
        $sth->syb_ct_data_info('CS_GET', 1);
    }
```

2. 使用 CS_IODESC 值进行更新：

```
$sth->syb_ct_prepare_send();
```

3. 设置将要插入的新数据项的大小，并不对此操作进行记录：

```
$sth->syb_ct_data_info('CS_SET', 1, {total_txtlen
=> length($image), log_on_update => 0});
```

4. 要在单个块中传输数据，请执行：

```
$sth->syb_ct_send_data($image, length($image));
```

5. 要提交操作，请执行：

```
$sth->syb_ct_finish_send();
```

错误处理

所有来自用于 Perl 和 CT-Lib 的 Adaptive Server 数据库驱动程序的错误都会传播到 DBI 层。

但驱动程序启动期间必须报告的错误或警告除外，因为这时尚无可用的上下文。

当启用 **PrintError** 属性时，DBI 层会执行基本错误报告。可使用 DBI 跟踪方法启用 DBI 操作跟踪，以跟踪程序或系统级别的问题。

下面的示例显示了如何添加更多的详细错误消息（服务器消息）：

- 在活动 **dbh** 上设置 **dbh->{syb_show_sql} =*1*** 以在 **$dbh->errstr** 返回的字符串中包含当前 SQL 语句。
- 在活动 **dbh** 上设置 **dbh->{syb_show_eed} =*1*** 以将扩展错误信息 (EED)（如重复插入失败和无效日期格式）添加到由 **$dbh->errstr** 返回的字符串。
- 使用 **syb_err_handler** 属性设置即席错误处理程序回调（即 Perl 子例程），该回调在常规处理程序执行其处理之前进行调用。如果该子例程返回 0，则将忽略此错误。这对于处理 Transact-SQL 中的 **PRINT** 语句以及 **showplan** 输出和 **dbcc** 输出非常有用。

  该子例程使用包含以下内容的参数进行调用以表示类型：Sybase 错误号、严重级、状态、SQL 批处理中的行号、服务器名称（如果有）、存储过程名称（如果有）、消息文本、SQL 文本和字符串“client”或“server”。

配置安全服务

使用 `ocs.cfg` 和 `libtcl.cfg` 文件配置安全选项。

1. 对于连接，使用 `ocs.cfg` 设置目录和安全属性。

   注意： 在 `ocs.cfg` 文件中，针对应用程序名称添加条目以便设置特定于驱动程序的选项。

2. 编辑 `libtcl.cfg` 以装载安全和目录服务驱动程序。

3. 要加密口令，请使用 **encryptPassword** DSN 选项。 例如：

```
DBI-
>connect("dbi:SybaseASE:server=mumbles;encryptPassword
=1", $user, $pwd);
```

示例

使用示例程序查看存储过程的基本用法，并从 pubs2 authors 表检索行。

示例 1

使用示例程序查看 Perl 中的存储过程的基本用法。

此程序首先连接到服务器，创建两个存储过程，调用预准备语句，绑定或执行过程，将结果输出到 STDOUT，然后断开连接并退出程序。

```
use strict;

use DBI qw(:sql_types);
use DBD::SybaseASE;

require_version DBI 1.51;

my $uid = "sa";
my $pwd = "";
my $srv = $ENV{"DSQUERY"} || die 'DSQUERY appears not set';
my $dbase = "tempdb";

my $dbh;
my $sth;
my $rc;

my $col1;
my $col2;
my $col3;
my $col4;

# Connect to the target server.
#
$dbh = DBI->connect("dbi:SybaseASE:server=$srv;database=$dbase",
    $uid, $pwd, {PrintError => 1});

# One way to exit if things fail.
#
if(!$dbh) {
    warn "Connection failed, check if your credentials are set
correctly?\n";
    exit(0);
}

# Ignore errors on scale for numeric. There is one marked call below
# that will trigger a scale error in ASE. Current settings suppress
# this.
#
$dbh->do("set arithabort off")
```

```
        || die "ASE response not as expected";

# Drop the stored procedures in case they linger in ASE.
#
$dbh->do("if object_id('my_test_proc') != NULL drop proc
my_test_proc")
    || die "Error processing dropping of an object";

$dbh->do("if object_id('my_test_proc_2') != NULL drop proc
my_test_proc_2")
    || die "Error processing dropping of an object";

# Create a stored procedure on the fly for this example. This one
# takes input args and echo's them back.
#
$dbh->do(qq{
create proc my_test_proc \@col_one varchar(25), \@col_two int,
    \@col_three numeric(5,2), \@col_four date
as
    select \@col_one, \@col_two, \@col_three, \@col_four
}) || die "Could not create proc";

# Create another stored procedure on the fly for this example.
# This one takes dumps the pubs2..authors table. Note that the
# format used for printing is defined such that only four columns
#appear in the output list.
#
$dbh->do(qq{
create proc my_test_proc_2
as
    select * from pubs2..authors
}) || die "Could not create proc_2";

# Call a prepare stmt on the first proc.
#
$sth = $dbh->prepare("exec my_test_proc \@col_one = ?, \@col_two
= ?,
    \@col_three = ?, \@col_four = ?")
        || die "Prepare exec my_test_proc failed";

# Bind values to the columns. If SQL type is not given the default
# is SQL_CHAR. Param 3 gives scale errors if arithabort is disabled.
#
$sth->bind_param(1, "a_string");
$sth->bind_param(2, 2, SQL_INTEGER);
$sth->bind_param(3, 1.5411111, SQL_DECIMAL);
$sth->bind_param(4, "jan 12 2012", SQL_DATETIME);

# Execute the first proc.
#
$rc = $sth->execute || die "Could not execute my_test_proc";

# Print the bound args
#
dump_info($sth);
```

```
# Execute again, using different params.
#
$rc = $sth->execute("one_string", 25, 333.2, "jan 1 2012")
    || die "Could not execute my_test_proc";

dump_info($sth);

# Enable retrieving the proc status.
$sth->{syb_do_proc_status} = 1;

$rc = $sth->execute(undef, 0, 3.12345, "jan 2 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute("raisin", 1, 1.78, "jan 3 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute(undef, 0, 3.2233, "jan 4 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute(undef, 0, 3.2234, "jan 5 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute("raisin_2", 1, 3.2235, "jan 6 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute(undef, 0, 3.2236, "jan 7 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

# End of part one, generate blank line.
#
print "\n";

# Undef the handles (not really needed but...).
#
undef $sth;
undef $rc;

# Prepare the second stored proc.
#
$sth = $dbh->prepare("exec my_test_proc_2")
        || die "Prepare exec my_test_proc_2 failed";

# Execute and print
#
$rc = $sth->execute || die "Could not execute my_test_proc_2";
dump_info($sth);

#
# An example of a display/print function.
#
```

```
sub dump_info {
    my $sth = shift;
    my @display;

    do {
    while(@display = $sth->fetchrow) {
     foreach (@display) {
        $_ = '' unless defined $_;
    }
     $col1 = $display[0];
     $col2 = $display[1];
     $col3 = $display[2];
     $col4 = $display[3];

    # Proc status is suppressed, assume proc
    # execution was always successful. Enable
    # by changing the write statement.
    #
    #write;
    write unless $col1 eq 0;
    }
} while($sth->{syb_more_results});
}

#
# The FORMAT template for this example.
#
format STDOUT_TOP =

Column1            Column2            Column3            Column4
------             ------             ------             ------
.

# Treat all data as left-justified strings
#
format STDOUT =
@<<<<<<<<<<<<   @<<<<<<<<<<<< @<<<<<<<<<<<<
@<<<<<<<<<<<<
$col1, $col2, $col3, $col4
.

# The End.....
#
$dbh->do("drop proc my_test_proc");
$dbh->do("drop proc my_test_proc_2");
$dbh->disconnect;
```

示例 2

使用示例程序从 `pubs2 authors` 表检索行，将其插入到 `tempdb`，然后附加新行以进行批量插入。 然后，此程序将更新的 `authors` 表输出到 STDOUT，断开连接并退出。

```
use strict;
```

```
use DBI ();
use DBD::SybaseASE ();

require_version DBI 1.51;

# trace(n) where n ranges from 0 - 15.
# use 2 for sufficient detail.
#DBI->trace(2); # 0 - 15, use 2 for sufficient detail

# Login credentials, handles and other variables.
#
my $uid = "sa";
my $pwd = "";
my $srv = $ENV{"DSQUERY"} || die 'DSQUERY appears not set';
my $dbase = "tempdb";
my $temp_table = "$dbase..authors";

my $rows;
my $col1;
my $col2;
my $dbh;
my $sth;
my $rc;

# Connect to the target server:
#
$dbh = DBI->connect("dbi:SybaseASE:server=$srv;database=$dbase",
    $uid, $pwd, {PrintError => 0, AutoCommit => 0})
    || die "Connect failed, did you set correct credentials?";

# Switch to the pubs2 database.
#
$rc = $dbh->do("use pubs2") || die "Could not change to pubs2";

# Retrieve 2 columns from pubs2..authors table.
#
$sth = $dbh->prepare(
    "select au_lname, city from authors where state = 'CA'")
    || die "Prepare select on authors table failed";

$rc = $sth->execute
    || die "Execution of first select statement failed";

# We may have rows now, present them.
#
$rows = dump_info($sth);
print "\nTotal # rows: $rows\n\n";

# Switch back to tempdb, we take a copy of pubs2..authors
# and insert some rows and present these.
#
$rc = $dbh->do("use $dbase") || die "Could not change to $dbase";

# Drop the authors table in tempdb if present
#
$rc = $dbh->do("if object_id('$temp_table') != NULL drop table
```

```
$temp_table")
    || die "Could not drop $temp_table";

# No need to create a tempdb..authors table as the select into will
# do that.

$rc = $dbh->do("select * into $temp_table from pubs2..authors")
    || die "Could not select into table $temp_table";

# Example of a batch insert...
#
$sth = $dbh->prepare("
    insert into $temp_table
    (au_id, au_lname, au_fname, phone, address, city, state,
        country, postalcode) values
    ('172-39-1177', 'Simpson', 'John', '408 496-7223',
        '10936 Bigger Rd.', 'Menlo Park', 'CA', 'USA', '94025')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode) values
('212-49-4921', 'Greener', 'Morgen', '510 986-7020',
    '309 63rd St. #411', 'Oakland', 'CA', 'USA', '94618')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('238-95-4766', 'Karson', 'Chernobyl', '510 548-7723',
    '589 Darwin Ln.', 'Berkeley', 'CA', 'USA', '94705')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('267-41-4394', 'OLeary', 'Mich', '408 286-2428',
    '22 Cleveland Av. #14', 'San Jose', 'CA', 'USA', '95128')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('274-80-4396', 'Straight', 'Shooter', '510 834-2919',
    '5420 College Av.', 'Oakland', 'CA', 'USA', '94609')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('345-22-1785', 'Smiths', 'Neanderthaler', '913 843-0462',
    '15 Mississippi Dr.', 'Lawrence', 'KS', 'USA', '66044')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('405-56-7012', 'Bennetson', 'Abra', '510 658-9932',
    '6223 Bateman St.', 'Berkeley', 'CA', 'USA', '94705')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
```

```
    country, postalcode ) values
('427-17-2567', 'Dullest', 'Annie', '620 836-7128',
    '3410 Blonde St.', 'Palo Alto', 'CA', 'USA', '94301')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('527-72-3246', 'Greene', 'Mstar', '615 297-2723',
    '22 Graybar House Rd.', 'Nashville', 'TN', 'USA', '37215')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('672-91-3249', 'Yapan', 'Okiko', '925 935-4228',
    '3305 Silver Ct.', 'Walnut Creek', 'CA', 'USA', '94595')
");

$rc = $sth->execute || die "Could not insert row";

# Retrieve 2 columns from tempdb..authors table and present these
#
$sth = $dbh->prepare(
    "select au_lname, city from $temp_table where state = 'CA'")
    || die "Prepare select on $temp_table table failed";

$rc = $sth->execute
    || die "Execution of second select statement failed";

# Output
#
$rows = dump_info($sth);
print "\nTotal # rows: $rows";
print "\n";

sub dump_info {
    my $sth = shift;
    my @display;
    my $rows = 0;

while(@display = $sth->fetchrow) {
    $rows++;
    foreach (@display) {
    $_ = '' unless defined $_;
    }
    $col1 = $display[0];
    $col2 = $display[1];
    write;
    }
    $rows;
}

# The FORMAT template for this example.
#
format STDOUT_TOP =

Lastname            City
```

```
--------              -------
.

format STDOUT =

@<<<<<<<<<<<<     @<<<<<<<<<<<<
$col1, $col2
.

$dbh->disconnect;
```

ESD #5 的新增功能

ESD #5 为 jConnect 7.07、Adaptive Server ODBC 驱动程序 15.7 和 Adaptive Server ADO.NET 数据提供程序 15.7 引入了新功能。

Adaptive Server ADO.NET 数据提供程序支持带有 COMPUTE 子句的 Transact-SQL 查询

Adaptive Server ADO.NET 数据提供程序现支持包括 **COMPUTE** 子句的 Transact-SQL 查询。

使用 **COMPUTE** 子句可在单个 **select** 语句中包括明细和摘要结果。 摘要行显示在特定组的明细行的下面，如下所示：

```
select type, price, advance from titles order by type compute
sum(price), sum(advance) by type
```

```
type          price      advance
------------  ---------  ----------
UNDECIDED     NULL       NULL
Compute Result:
----------------------- ------------------------
NULL                     NULL
type          price      advance
------------  ---------  ----------
business      2.99       10,125.00
business      11.95      5,000.00
business      19.99      5,000.00
business      19.99      5,000.00
Compute Result:
----------------------- ------------------------
54.92                    25,125.00
...
...

(24 rows affected)
```

当 Adaptive Server ADO.NET 数据提供程序执行包含 **COMPUTE** 子句的 **select** 语句时，该提供程序会向客户端返回多个结果集。 结果集的数量取决于可用的唯一分组的数量。每个组包含一个用于明细行的结果集和一个用于摘要的结果集。 客户端必须处理所有结果集才能完全处理返回的行，否则返回的第一个结果集中将只包括第一个数据组的明细行。

有关 **COMPUTE** 子句的详细信息，请参见《Adaptive Server Enterprise Transact-SQL 用户指南》。

有关处理多个结果集的详细信息，请参见 Microsoft 网站上的《ADO.NET 程序员指南》。

向 Adaptive Server 快速传输数据的新 SSIS 自定义数据流目标组件

Adaptive Server ADO.NET 数据提供程序分发版现增加一个 SQL Server 集成服务 (SSIS) 自定义数据流目标组件，该组件可将数据快速传输到 Adaptive Server 目标。

快速数据传输使用 **AseBulkCopy** 类支持的 Adaptive Server 批量插入协议。此组件名为 SybaseAdaptiveServerAdoNetDestination，随 Adaptive Server ADO.NET 数据提供程序和程序集文件一起安装在以下位置：`%SYBASE%\DataAccess\ADONET\SybaseAdaptiveServerAdoNetDestination.dll`（32 位系统）和 `%SYBASE%\DataAccess64\ADONET\SybaseAdaptiveServerAdoNetDestination.dll`（64 位系统）。

针对 SQLServer 2008 配置 Adaptive Server ADO.NET 目标 SSIS 组件

配置 Adaptive Server ADO.NET 目标 SSIS 组件。

1. 将 `Sybase.AdoNet2.AseDestination.dll` 复制到 `C:\Program Files\Microsoft SQL Server\100\DTS\PipelineComponents` 和 `C:\Program Files (x86)\Microsoft SQL Server\100\DTS\PipelineComponents`。
2. 在本地驱动器中的任一 Microsoft SQL Server 目录中，使用 SDK 安装中提供的 AseGacUtility 来注册 `Sybase.AdoNet2.AseDestination.dll`。
3. 启动 SQL Server Business Intelligence Studio。
4. 在“工具箱”选项卡上，右键单击“数据流目标”，然后选择“选择项”。随即显示“选择工具箱项”窗口。
5. 选择“SSIS 数据流项”选项卡。单击“Sybase Adaptive Server Enterprise ADO NET 目标”(Sybase Adaptive Server Enterprise ADO NET Destination)，然后单击“确定”。选择**“工具箱”** > **“数据流目标”**以查看 Sybase Adaptive Server ADO NET 目标组件。
6. 要创建 SSIS 项目，请选择**“文件”** > **“新建”** > **“项目”** > **“Integration Services 项目”**菜单。从“控制流项”工具箱创建或拖放“控制流”对象。
7. 在“数据流目标”和“数据流源工具箱”(Data Flow Sources Toolbox) 选项卡中，将 Sybase Adaptive Server ADO NET 目标组件和 ADO NET 源组件拖放到“数据流”选项卡。
8. 如果“连接管理器”窗口中没有可用的源或目标连接，则右键单击“连接管理器”窗口并选择“新建 ADO.NET 连接”。选择已存在的数据连接，或单击“新建”。

9. 要创建与目标 Adaptive Server 的新连接，请在“配置 ADO.NET 连接管理器”窗口中单击“新建”按钮，然后选择“Sybase Adaptive Server Enterprise 数据提供程序”(Sybase Adaptive Server Enterprise Data Provider)。
10. 在“连接管理器”窗口中输入连接属性。
11. 要启用批量插入，请在“其它连接属性”(Additional Connection Props) 文本框中输入： `enablebulkload=1`

    注意： 有关使用批量插入功能的更多详细信息，请参见《Adaptive Server Enterprise ADO.NET 数据提供程序用户指南》中的 **AseBulkCopy**。
12. 单击“确定”。
13. 针对“数据流”中的 ADO.NET 源，设置连接和数据访问模式。 从 ADO.NET 源连接数据流路径后，右键单击“Sybase Adaptive Server ADO NET 目标组件”(Sybase Adaptive Server ADO NET Destination Component)，然后选择“显示高级编辑”(Show Advanced Edit)。
14. 在“连接管理器”选项卡中，从“连接管理器”字段选择 ASE 连接。 在“组件属性”选项卡中，将 TableName 属性设置为目标表名。
15. 选择“输入列”选项卡，然后选中“名称”复选框。 这将选择源表指定的所有列。
16. 单击“确定”。

    注意： 用于从 SQL Server 2008 进行数据传输的 SSIS 目标组件已从 `Sybase.AdaptiveServerAdoNetDestination.dll` 重命名为 `Sybase.AdoNet2.AseDestination.dll`。

连接已建立。 有关数据传输的详细信息，请参见 Microsoft SSIS 文档。

jConnect 动态记录级别

jConnect 已得到增强，现在允许应用程序用户将消息粒度设置为 Level.FINE、Level.FINER 和 Level.FINEST。

例如：

- 如果用户将 **SybConnection** 类的记录级别设置为 Level.FINE，jConnect 将报告：**Dr1_Col setClientInfo**(Properties)
- 如果将 **SybConnection** 类的记录级别设置为 Level.FINER，将报告：**Dr1_Co1 setClientInfo**(Properties.size = [3])
- 如果将 **SybConnection** 类的记录级别设置为 Level.FINEST，将报告：**Dr1_Co1 setClientInfo**(Properties = [[ClientUserValue, ApplicationNameValue, ClientHostnameValue]])

请参见《jConnect for JDBC 程序员参考》。

jConnect 中转换程序类的包名称发生更改

在 jConnect 7.07 中，所有字符集转换程序类的包名称和文件路径都已发生更改。

字符集转换程序类文件已从 `com/sybase/jdbc4/utils` 移动到 `com/sybase/jdbc4/charset`。 jConnect 7.07 中字符集转换程序类的包名称更改包括：

- **com.sybase.jdbc4.utils.TruncationConverter** 已更改为 **com.sybase.jdbc4.charset.TruncationConverter**
- **com.sybase.jdbc4.utils.PureConverter** 已更改为 **com.sybase.jdbc4.charset.PureConverter**

注意： 如果您为使用完整包名称而声明了用于扩展字符集转换程序类的类，则必须将包名称从 **com.sybase.jdbc4.utils** 更改为 **com.sybase.jdbc4.charset**。

建议您使用通配符导入而不是编码类引用。 例如：

```
import com.sybase.jdbc4.charset.*;

import com.sybase.jdbc4.utils.*;
```

包名称的转换程序类引用由导入语句来解决。

jConnect 中增加的 PreparedStatement 参数限制

在以前的版本中，**PreparedStatement** 参数的最大数量限制为 2048。jConnect 7.07 现在支持 32767 个参数，当连接到 Adaptive Server 时，还可支持更大的限制。

Adaptive Server ODBC 驱动程序的新连接属性 SkipRowCountResults

SkipRowCountResults 连接属性可用于控制 ODBC 驱动程序如何处理返回行计数结果的语句。

UPDATE、**INSERT** 和 **DELETE** 语句返回行计数结果。 **SELECT** 语句返回结果集。 ODBC 应用程序可执行一批使用混合语句返回行计数或结果集的语句。

当 **SkipRowCountResults** 设置为 1（缺省值）时，Adaptive Server ODBC 驱动程序将跳过任意个行计数结果。 使用 **SQLExecDirect** 或 **SQLExecute** 执行一批语句后，ODBC 应用程序将定位在第一个结果集上。 后续调用 **SQLMoreResults** 时将跳过行计数结果，并且应用程序将定位在下一个可用的结果集上。

当 **SkipRowCountResults** 设置为 0 时，Adaptive Server ODBC 驱动程序将在每个结果集或行计数上停留。 使用 **SQLExecDirect** 或 **SQLExecute** 执行一批语句后，应用程序将定位在第一个可用的结果上，该结果可以是结果集，也可以是行计数。 ODBC 应

用程序可使用 **SQLFetch** 检索结果集，或使用 **SQLRowCount** 检索行计数结果。 后续调用 **SQLMoreResults** 时会将应用程序定位到下一个可用结果，该结果可以是结果集，也可以是行计数。

对 Adaptive Server ODBC 驱动程序中 AF_UNIX 套接字的支持

Adaptive Server ODBC 驱动程序现在支持 **AF_UNIX** 套接字，以便与 Adaptive Server 通信。

此支持当前限制为 Linux x86-64 64 位平台。 当 ODBC 应用程序和 Adaptive Server 位于相同主机上，且均配置为使用 AF_UNIX 套接字，则可以使用 **AF_UNIX** 套接字。 AF_UNIX 套接字可提供比 TCP/IP 套接字更好的性能。 若要从 ODBC 启用 AF_UNIX 套接字，请设置以下连接字符串属性：

- **networklibraryname=afunix** – 用于通知 Adaptive Server ODBC 驱动程序使用 AF_UNIX 套接字。
- **server=<管道的完整路径>** – AF_UNIX 套接字的路径。 例如，`/tmp/test/demo_socket`。

有关配置 Adaptive Server 以使用 AF_UNIX 套接字的详细信息，请参见 Sybase Adaptive Server Enterprise 文档。

Adaptive Server ODBC 驱动程序的 AdjustLargePrecisionAndScale 连接属性

在低于 15.7 的版本中，Adaptive Server ODBC 驱动程序不支持用于设置数值或小数列的标度和精度的 **SQLSetDescField()** 调用。

所有对此 API 的调用均已忽略，并且 Adaptive Server ODBC 驱动程序将根据接收的值来设置列的精度和标度。 由于 Adaptive Server 支持的精度大于 ODBC 数值结构，因此 Adaptive Server ODBC 驱动程序将根据需要进一步缩小从服务器接收的值，以适应 ODBC 数值结构。 在 15.7 及更高版本中，Adaptive Server ODBC 驱动程序将不再忽略用于设置数值或小数列的精度和标度的 **SQLSetDescField()** 调用。 因此，可以发现之前工作的 ODBC 应用程序现在将接收到新 Adaptive Server ODBC 驱动程序的数据流错误。 **AdjustLargePrecisionAndScale** 属性允许以前的行为继续，并启用 Adaptive Server ODBC 驱动程序来选择最佳精度和标度，以适应从服务器接收的值。

缺省情况下，**AdjustLargePrecisionAndScale** 设置为 0，这将导致 Adaptive Server ODBC 驱动程序接受用于设置精度或标度的 **SQLSetDescField()** API 调用。

将 **AdjustLargePrecisionAndScale** 连接属性设置为 1 时，Adaptive Server ODBC 驱动程序将忽略用于设置精度或标度的所有 **SQLSetDescField()** API 调用，并使用实际数据值的精度和标度。

有关 **SQLSetDescField()** 的详细信息，请参见 Microsoft Developers Networkhttp://msdn.microsoft.com/。

ESD #4 的新功能

ESD #4 为 Open Client 15.7 和 Open Server 15.7、SDK 15.7、用于 Python 15.7 的 Adaptive Server Enterprise 扩展模块、用于 PHP 15.7 的 Adaptive Server Enterprise 扩展模块以及用于 Perl 15.7 的 Adaptive Server Enterprise 数据提供程序引入了新功能。

ESD #4 中的 Open Client 15.7 和 Open Server 15.7 功能

Open Client 15.7 和 Open Server 15.7 经增强现可提供新功能，包括适用于 Open Client 和 Open Server 文件 (UNIX) 的更为严格的权限、成批参数以及新的安全字符串处理例程。

适用于 Open Client 和 Open Server 文件（仅限 UNIX）的更为严格的权限

从 ESD#4 开始，新生成的 Open Client 和 Open Server 文件具有更为严格的权限。

表 6. 文件及其权限设置

文件	权限
Interfaces 文件	rw- r-- r-- (644)
BCP 数据文件	rw- r-- --- (640)
BCP 格式文件	rw- r-- --- (640)
BCP 输出文件	rw- --- --- (600)
BCP 错误文件	rw- --- --- (600)
ISQL 输出文件（-o 选项）	rw- --- --- (600)
ISQL 命令历史记录文件	rw- --- --- (600)
ISQL 临时文件	rw- --- --- (600)
ISQL 输出重定向	rw- --- --- (600)
Open Server 日志文件	rw- --- --- (600)
LDAP 调试日志文件	rw- --- --- (600)
Kerberos 调试日志文件	rw- --- --- (600)
Netlib 跟踪输出文件	rw- --- --- (600)
DCL 跟踪输出文件	rw- --- --- (600)

注意： 这些权限仅应用于新生成的文件；现有文件保留其自身权限（通常为 rw- rw- rw-(666)）。 Microsoft Windows 上的文件权限保持不变。

用于设置 libtcl*.cfg 文件的替代路径的新 SYBOCS_TCL_CFG 环境变量

从 ESD#4 开始，可使用新的 SYBOCS_TCL_CFG 环境变量来设置 `libtcl.cfg` 和 `libtcl64.cfg` 文件的替代完整路径名。

例如：

Windows：

```
set SYBOCS_TCL_CFG = c:\joe\libctl.cfg
```

UNIX：

```
%setenv SYBOCS_TCL_CFG /usr/u/joe/libtcl.cfg
```

缺省情况下，Windows 系统与 UNIX 系统分别在 `%SYBASE%\%SYBASE_OCS%\ini` 目录和 `$SYBASE/$SYBASE_OCS/config` 目录中搜索 `libtcl.cfg` 和 `libtcl64.cfg` 文件。

也可以使用 CS_LIBTCL_CFG 属性设置 `libtcl.cfg` 和 `libtcl64.cfg` 文件的替代路径。

用于设置通用远程口令的新 isql 命令行选项 --URP

使用新的 `--URP` 命令行选项可为访问 Adaptive Server 的客户端设置通用远程口令。

```
isql  --URP  remotepassword
```

remotepassword 是通用远程口令。

示例：

```
%isql --URP “ASEremotePW”
```

用于 SYBPLATFORM 的新 linux64 和 nthread_linux64 设置

`linux64` 和 `nthread_linux64`（用于线程化应用程序）现在是 *SYBPLATFORM* 环境变量的有效设置，可用于在 Linux x86-64 64 位上编译 Open Client 和 Open Server 示例应用程序。 现有的 `linuxamd64` 和 `nthread_linuxamd64` 设置仍具有同样的作用。

用于 Microsoft Windows 64 位的 LAN Manager 驱动程序

Open Client 和 Open Server 包括 `libsybsmssp64.dll`，后者是用于 Microsoft Windows x86-64 64 位的 64 位 LAN Manager 驱动程序。 `libsybsmssp64.dll` 位于 `%SYBASE%\%SYBASE_OCS%\dll` 中；其行为类似于 32 位驱动程序 `libsybsmss.dll`。

支持成批参数

从 ESD #4 开始，Open Client 和 Open Server 允许在不结束命令本身的情况下发送多组命令参数。

在 Open Client 应用程序中，重复使用新的 **ct_send_params()** 例程传输参数，而无需处理上一命令的结果和重新发送命令本身。 在 Open Server 应用程序中，将 **SRV_S_PARAM_BATCHING** 属性设置为 CS_TRUE。

ct_send_params

以批量形式发送命令参数。

语法

```
CS_RETCODE ct_send_params(
  CS_COMMAND *cmd,
  CS_INT reserved)
```

参数

- *cmd*
  指向 CS_COMMAND 结构的指针。
- *reserved*
  设置为 CS_UNUSED。 其为占位符，保留以备后用。

返回值

ct_send_params 返回：

返回内容	表示
CS_SUCCEED	例程成功完成。
CS_FAIL	例程失败。

用法

对此函数的调用会发送早期使用 **ct_param()** 或 **ct_setparam()** 所指示的参数。 要停止发送参数，请在最后一个 **ct_send_params()** 调用之后使用 **ct_send()** 调用。 这将示意参数结束并完成当前命令。

- 第一个 **ct_send_params()** 调用将向服务器发送实际命令、所有参数的参数格式以及第一组参数。 后续调用仅发送其它参数而不发送格式。
- 每次调用 **ct_send_params()** 时都会刷新包含参数的网络缓冲区，以便服务器可以开始处理该命令。
- 与 **ct_send()** 不同，**ct_send_params()** 不会结束当前命令。 可重复调用 **ct_send_params()** 以发送多组参数。

- 只有在 **ct_send()** 调用完成命令后，才开始处理结果。 如果在 **ct_send()** 之前调用 **ct_results()**，则会导致错误。

使用 ct_setparam() 进行重新绑定

发送多组参数时，应用程序可能需要将 CT-Library 指向与上一组参数不同的其它内存位置。

要重新绑定参数，请使用 **ct_setparam()** 为数据提供不同的位置。 以下是现有的 **ct_setparam()** 声明：

```
ct_setparam(cmd, datafmt, data, datalenp, indp)

  CS_COMMAND *cmd;
  CS_DATAFMT *datafmt;
  CS_VOID *data;
  CS_INT *datalenp;
  CS_SMALLINT *indp;
```

在 **ct_setparam()** 调用中为 *data*、*datalenp* 和 *indp* 参数提供新值以绑定到不同的内存位置。

在 **ct_send_params()** 调用之后，无法更改参数的格式。 因此，在调用 **ct_send_params()** 之后进行的任何 **ct_setparam()** 调用都必须为 *datafmt* 传递一个 NULL 值。

只能重新绑定最初使用 **ct_setparam()** 进行绑定的参数。

针对 Server-Library 的成批参数支持

要在 Open Server Server-Library 中启用成批参数支持，请将 **SRV_S_PARAM_BATCHING** 服务器属性设置为 CS_TRUE。

例如，在 **srv_run()** 之前：

```
if (srv_props(ctos_ctx->cx_context, CS_SET,
SRV_S_PARAM_BATCHING, (CS_VOID *)&cs_true, sizeof(cs_true), NULL) !=
CS_SUCCEED)
{...}
```

然后，当命令包含多组命令参数时，**srv_xferdata()** 将具有两个新的返回代码。

- CS_PARAMS_MORE - 指示已成功复制参数，且批处理中含有多个参数。
- CS_PARAMS_END - 指示已成功复制参数。 它是批处理中的最后一组参数。

示例程序

有两个新的 CT-Library 示例程序可用。

- `batch_lang.c` - 演示如何将 **ct_send_params()** 与语言语句配合使用。 该示例重复使用 **ct_send_params()** 将从文件读取的行插入表中。 由于每次读取行时都对参数使用相同的位置，因此不必在 **ct_send_params()** 的两次调用之间调用 **ct_param()** 或 **ct_setparam()**。

- `batch_dynamic.c` - 使用动态 SQL 将参数发送到数据驻留在不同内存位置的服务器。 因此，此示例还将演示再次调用 **ct_send_params()** 之前，如何使用 **ct_setparam()** 重新绑定到其它变量。

`ctos` 示例程序已更新，现包括如下内容:

- 打开 SRV_S_PARAM_BATCHING 服务器属性。
- 使用 **ct_setparams()** 将 CT-Lib 绑定到数据位置。
- 处理来自 **srv_xferdata()** 的新返回值
- 为每组命令参数调用 **ct_send_params()**。

新的 CS-Library 字符串处理例程

cs_strlcpy、**cs_strlcat** 和 **cs_snprintf** 是三个新的 CS-Library 字符串处理例程。

cs_strlcpy
安全字符串复制函数。 最多将 *target_size*-1 个字符从 *source_str* 复制到 *target_str*，必要时需截断。 结果始终是以空值终止的字符串，*source_str* 或 *target_str* 为 NULL 或者 *target_size* 为 0 时除外。

语法

```
CS_RETCODE cs_strlcpy(target_str, source_str, target_size)

 CS_CHAR           *target_str;
 CS_CHAR           *source_str;
 CS_INT                *target_size;
```

参数

- *target_str*
  源字符串要复制到的目标字符串。
- *source_str*
  要复制的源字符串。
- *target_size*
  目标字符串的大小

返回值

- *source_str* 为 NULL、*target_str* 为 NULL 或 *target_size* 为 0 时，返回值为 0。
- 溢出时返回值为 *target_size*。
- 所有其它情况下，返回值均为 **strlen(*source_str*)**。

cs_strlcat
安全字符串并置函数。 最多将 *source_str* 的 *target_size* - **strlen**(*target_str*) - 1 个字符附加至 *target_str*。 结果始终为以空值终止的字符串，*source_str* 或 *target_str* 为 NULL、*target_size* 为 0 或 *target_str* 所指向的字符串长度大于 *target_size* 字节时除外。

语法

```
CS_RETCODE cs_strlcat(target_str, source_str, target_size)

 CS_CHAR           *target_str;
 CS_CHAR           *source_str;
 CS_INT                *target_size;
```

参数

- *target_str*
  源字符串要附加到的目标字符串。
- *source_str*
  要附加的源字符串。
- *target_size*
  目标字符串的大小

返回值

- *source_str* 为 NULL、*target_str* 为 NULL 或 *target_size* 为 0 时，返回值为 0
- 溢出时返回值为 *target_size*
- 其它所有情况下，返回值均为 **strlen(*target_str*) + strlen(*source_str*)**

cs_snprintf

用于所有平台的公用 **snprintf**（类似于函数），提供格式化的输出转换。 结果始终是以空值终止的字符串。

语法

```
void cs_snprintf(char *str, size_t size, const char *format, ...)
```

参数

- *str*
  输出所写入到的字符串。
- *size*
  写入的最大字节数。
- *format*
  由零个或多个转换指令组成的字符串。

返回值

无

ESD #4 中针对 jConnect、Adaptive Server 驱动程序和提供程序的 SDK 15.7 功能

ESD #4 为 jConnect for JDBC 7.07、Adaptive Server Enterprise ODBC 驱动程序 15.7、Adaptive Server Enterprise OLE DB 提供程序 15.7 和 Adaptive Server Enterprise ADO.NET 数据提供程序 15.7 引入了新功能。

细化权限和谓词权限

从 Adaptive Server 15.7 ESD #2 开始，角色特权管理模型有所增强。

- 已添加全新且细化的可授予系统特权，以强化“职责分离”(SOD) 和“最小特权”(LP) 的原则。 这些可授予的系统权限可以是服务器范围特权或数据库范围特权。
- 系统定义的角色 *sa_role*、*sso_role*、*oper_role*、*replication_role* 和 *keycustodian_role* 现已重新构建为含有一组显式授予的特权的*特权容器*。
- 现在可从即用的系统定义角色来创建自定义角色，具体做法是授予或撤消特权。
- **CREATE PROCEDURE** 语句现在支持新的 **EXECUTE AS OWNER | CALLER** 选项。 因此，ASE 可作为过程所有者或过程调用方来检查运行时权限、执行 DDL 以及解析对象名。
- 使用新的 **enable granular permissions** 配置选项可启用增强的角色特权管理模型。

请参见 Adaptive Server Enterprise 15.7 ESD #2 文档。

如果在启用新的角色特权管理模型的情况下连接到 Adaptive Server，jConnect for JDBC、Adaptive Server Enterprise ODBC 驱动程序、Adaptive Server Enterprise OLE DB 提供程序和 Adaptive Server Enterprise ADO.NET 数据提供程序将支持该新模型。

为支持返回用于授予谓词特权的谓词的相关信息，下列方法将返回一个名为 PREDICATE 的附加列：

- ODBC – **SQLColumnPrivileges()** 和 **SQLTablePrivileges()**
- JDBC – **ResultSet getColumnPrivileges()** 和 **ResultSet getTablePrivileges()**
- OLE DB – **IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMN_PRIVILEGES)** 和 **IDBSchemaRowset::GetRowset(DBSCHEMA_TABLE_PRIVILEGES)**

如果对数据库设置了细化权限，这些方法将返回传达细化权限的附加行。

ADO.NET 方法的行为无任何变化。

在不复制数据的情况下更改表删除列

Adaptive Server 15.7 ESD #2 版允许您在不执行数据复制的情况下从表中删除列。

这可减少运行 **alter table drop column** 所需的时间量。 请参见 Adaptive Server Enterprise 15.7 ESD #2 文档。

如果在启用此新功能的情况下连接到 Adaptive Server，jConnect for JDBC、Adaptive Server Enterprise ODBC 驱动程序、Adaptive Server Enterprise OLE DB 提供程序和

Adaptive Server Enterprise ADO.NET 数据提供程序将支持使用该功能进行常规 DML 操作（**insert**、**delete**、**update** 和 **merge**）。 无需进行任何特殊配置即可使用该功能；系统自动对其提供支持。

如果在启用该功能的情况下连接到 Adaptive Server，jConnect for JDBC 和 Adaptive Server Enterprise ODBC 驱动程序还支持使用该功能进行批量复制。

该功能不可用于非物化列或虚拟计算列、加密列和 XML 列。

快速记录批量插入

Adaptive Server 15.7 ESD #2 版可以在 **fast** 模式下完全记录 **bcp**，以便实现完全数据恢复。

以前版本的 **bcp** 在 **fast** 模式下只记录页分配。 请参见 Adaptive Server Enterprise 15.7 ESD #2 文档。

在 jConnect for JDBC 中，将 ENABLE_BULK_LOAD 连接属性设置为新值 LOG_BCP 以启用完全记录。

在 ODBC 驱动程序中，将 EnableBulkLoad 连接属性设置为新值 3 以启用完全记录。或者，在 ODBC 应用程序中将 SQL_ATTR_ENABLE_BULK_LOAD 连接属性设置为所需级别：

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_ENABLE_BULK_LOAD,
(SQLPOINTER)3, SQL_IS_INTEGER);
```

这允许单个连接使用不同类型的批量装载。

在 ADO.NET 提供程序中，将 EnableBulkLoad 连接属性设置为新值 3 以启用完全记录。

动态记录

从 ESD #4 开始，jConnect for JDBC 通过实施标准 Java Logger 机制来支持记录机制。

现在，应用程序可控制 jConnect 的记录器并根据需要打开或关闭记录。 请参见《jConnect for JDBC 程序员参考》。

动态客户端信息设置

从 ESD #4 开始，即使在建立连接后，您也可以使用 **setClientInfo()** 和 **getClientInfo()** 标准方法为 jConnect for JDBC 客户端信息属性（ApplicationName、ClientUser、ClientHostName）设置新值。

动态连接属性设置

从 ESD #4 开始，即使在建立连接后，您也可以使用 **setClientInfo()** 和 **getClientInfo()** 标准方法为 jConnect for JDBC 连接属性设置新值。

有关可动态设置的连接属性的列表，请参见《jConnect for JDBC 程序员参考》。

异常处理

jConnect for JDBC 中的异常处理有所增强。 当异常消息中包含使用 **getcause()** 的指令时，可使用 **getCause()** 方法获取导致异常的原因。

可提高性能的新 jConnect 连接属性

从 ESD #4 开始，jConnect for JDBC 具有一组可提高性能的新连接属性。

属性	说明	缺省值
OPTIMIZE_STRING_CONVERSIONS	指定是否启用字符串转换优化。 客户端在 SQL 预准备语句中使用字符数据类型时，此优化行为可提高 jConnect 的性能。 值: • 0 - 缺省值；不启用字符串转换优化。 • 1 - 在 jConnect 使用 utf8 或服务器缺省字符集时启用字符串转换优化。 • 2 - 在所有情况下均启用字符串转换优化。	0
SUPPRESS_PARAM_FORMAT	在执行动态 SQL 预准备语句时，jConnect 客户端可使用 SUPPRESS_PARAM_FORMAT 连接字符串属性来抑制参数数据 (TDS_PARAMS)。 在可能的情况下，客户端会减少发送的参数元数据以提高性能。 值: • false - 在 select、insert 和 update 操作中不抑制 TDS_PARAMFMT。 • true - 缺省值；在可能的情况下抑制 TDS_PARAMFMT。	true
SUPPRESS_ROW_FORMAT	在 jConnect 中，客户端可使用 SUPPRESS_ROW_FORMAT 连接字符串属性来强制 Adaptive Server 仅在动态 SQL 预准备语句的行格式更改时发送 TDS_ROWFMT 或 TDS_ROWFMT2 数据。 这样，Adaptive Server 可以尽量向客户端发送较少的数据，从而提高性能。 值: • false – 即便行格式未发生更改，也发送 TDS_ROWFMT 或 TDS_ROWFMT2 数据。 • true – 缺省值；强制服务器仅在行格式更改时才发送 TDS_ROWFMT 或 TDS_ROWFMT2 数据。	true

新的 jConnect 连接属性

从 ESD #4 开始，jConnect for JDBC 具有一组新连接属性。

属性	说明	缺省值
EARLY_BATCH_READ_THRESHOLD	指定行数阈值，超出该值后，读取器线程应启动，以清除批处理的服务器响应。 如果无需较早读取，则将该值设置为 -1。	-1
STRIP_BLANKS	强制服务器在将字符串值存储到表中之前，先删除其前导空白和尾随空白。 值： • false – 缺省值；客户端发送的字符串值“按原样”存储。 • true – 在将字符串值存储到表中之前，先删除其前导空白和尾随空白。	false
SUPPRESS_CONTROL_TOKEN	取消发送控制令牌。 值： • false – 缺省值；发送控制令牌。 • true – 取消发送控制令牌。	false

有关 JDBC 的 Hibernate 支持的注意事项

Hibernate 是相关项目的集合，通过它开发人员可在其扩展超出“对象”或“关系映射”的应用程序中利用 POJO 样式域模型。 在众多模块中，Hibernate 核心模块对“对象关系映射”进行处理。

方言可帮助 Hibernate 以其自己的语言与数据库进行通信。 Hibernate 已为各种版本的 Adaptive Server Enterprise 创建了方言文件：

Sybase 方言文件	ASE 版本
`Sybase11Dialect.java`	11.9.2
`Sybase15Dialect.java`	15.0
`Sybase157Dialect.java`	15.7

注意： Hibernate 和 Sybase 主动测试最新版本并在必要时创建新方言。 所有更新的方言均为预定的 Hibernate 版本的一部分。 该版本计划可能与 Adaptive Server 版本计划不相符。 如果在发布相应的 Hibernate 版本之前需要访问更新的方言，请从 Hibernate on Sybase ASE 中获取。

支持 SQL_ATTR_OUTPUT_NTS=SQL_FALSE

Adaptive Server Enterprise ODBC 驱动程序现在允许您将 **SQL_ATTR_OUTPUT_NTS** 属性设置为 SQL_FALSE，这样驱动程序就不会返回以空值终止的字符串数据。

请先设置该属性，然后再分配连接句柄：

```
SQLSetEnvAttr(hEnv, SQL_ATTR_OUTPUT_NTS, (SQLPOINTER)SQL_FALSE,
SQL_IS_INTEGER)
```

缺省情况下，**SQL_ATTR_OUTPUT_NTS** 属性设为 SQL_TRUE，而且所有输出字符串均为以空值终止。

支持长度为 8 字节的 SQLLEN 数据类型（仅限 Linux 64 位）

现在，适用于 Linux x86-64 64 位和 Linux on POWER 64 位的 Adaptive Server Enterprise ODBC 驱动程序支持 4 字节的 SQLLEN 数据类型和 8 字节的 SQLLEN 数据类型。

Red Hat 和 SUSE 提供 unixODBC 驱动程序管理器作为其驱动程序管理器。 2.2.13 之前版本的 unixODBC 驱动程序管理器应使用 4 字节的 SQLLEN 数据类型。 2.2.13 及更高版本的 unixODBC 驱动程序管理器的缺省配置（如由 Red Hat Enterprise Linux 6 及更高版本所提供）应使用 8 字节的 SQLLEN 数据类型。 因此，Adaptive Server Enterprise ODBC 驱动程序提供两种版本的驱动程序。 请检查您的 64 位 Linux 系统所使用的 unixODBC 驱动程序管理器版本。

从 ESD #4 开始，DataAccess64/ODBC/lib/ 目录中存在两个驱动程序共享库文件和一个软链接：

- libsybdrvodb-sqllen4.so - 等同于支持 4 字节 SQLLEN 数据类型的原始 libsybdrvodb.so 文件
- libsybdrvodb-sqllen8.so 文件 - 支持 8 字节 SQLLEN 数据类型的新版 libsybdrvodb.so 文件
- 指向原始驱动程序共享库文件的 libsybdrvodb.so 软链接现在命名为 libsybdrvodb-sqllen4.so

如果要继续使用 4 字节的 SQLLEN 数据类型，则无需进行更改。

要使用 8 字节的 SQLLEN 数据类型，请修改该软链接，使其指向 libsybdrvodb-sqllen8.so 文件：

```
> cd DataAccess64/ODBC/lib
> rm libsybdrvodb.so
> ln -s libsybdrvodb-sqllen8.so libsybdrvodb.so
```

ODBC 延迟数组绑定

Adaptive Server Enterprise ODBC 驱动程序现在提供扩展的 **SQLBindColumnDA()** 和 **SQLBindParameterDA()** API，它们允许应用程序通过单个 API 调用绑定所有列或参数。

如果您使用这些 API，则在每次调用 **SQLExecute()** 或 **SQLExecDirect()** 时，都会对指向列缓冲区或参数缓冲区的指针重新求值。 因此，应用程序能够在不另外调用 **SQLBindCol()** 或 **SQLBindParameter()** 的情况下更改缓冲区。 因为用于绑定新指针的调用会耗用很多资源，所以，在需要多次执行同一语句的情况下使用经扩展的新 API 可提高应用程序性能。 应用程序还可以在执行查询前更改缓冲区指针，藉此省去一些内存复制操作，这样，便可从可用位置读取数据或将数据复制到所需位置。

请参见《Sybase Adaptive Server Enterprise ODBC 驱动程序用户指南》。

对 ODBC 数据批处理的批量插入支持

15.7 版本中引入了在不绑定参数数组的情况下对 ODBC 数据进行批处理的功能，该功能经扩展现可支持使用批量插入协议进行批量插入。

要启用该功能，请将 EnableBulkLoad 连接属性设置为所需的批量插入级别（1、2 或 3），并将 HomogeneousBatch 连接属性设置为 2。 请参见《Sybase Adaptive Server Enterprise ODBC 驱动程序用户指南》。

例如，在连接字符串中添加 `;enablebulkload=3;homogeneousbatch=2`，在批处理中执行的简单插入语句即转换为快速记录批量插入语句。

或者，使用 SQL_ATTR_HOMOGENEOUS_BATCH 和 SQL_ATTR_ENABLE_BULK_LOAD 连接属性通过编程方式设置相关连接属性，也可以得到相同的结果：

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_HOMOGENEOUS_BATCH,
(SQLPOINTER)2, SQL_IS_INTEGER);
sr = SQLSetConnectAttr(hdbc,
SQL_ATTR_ENABLE_BULK_LOAD, (SQLPOINTER)3, SQL_IS_INTEGER);
```

支持在不使用 ODBC 驱动程序管理器跟踪的情况下进行动态记录

Adaptive Server Enterprise ODBC 驱动程序 15.7 引入了在不使用 ODBC 驱动程序管理器跟踪的情况下进行应用程序记录的功能。

可在应用程序执行的持续期间内启用（或禁用）应用程序记录。 请参见“在不使用 ODBC 驱动程序管理器跟踪的情况下进行记录”。

ESD #4 对该支持进行了扩展，即：您可以在应用程序执行期间，通过设置新的 SQL_OPT_TRACE 环境属性来动态启用或禁用应用程序记录。 有效值为 0（缺省值，表示禁用）或 1（表示启用）。

```
// enable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)1,
  SQLINTEGER);
```

```
// disable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)0,
  SQLINTEGER);
```

- 动态记录的启用和禁用是全局性的，因此将影响所有连接，而不管连接何时开启，也不管其是否是用于设置 SQL_OPT_TRACE 的环境句柄的一部分。
- 缺省情况下，日志写入到当前目录的 `sybodbc.log` 文件中。 使用 SQL_OPT_TRACEFILE 环境属性设置其它文件或文件路径。

  ```
  SQLSetEnvAttr(0, SQL_OPT_TRACEFILE, (SQLPOINTER) "logfilepath",
  SQL_NTS);
  ```
- 如果设置 LOGCONFIGFILE 环境变量或注册表值，则会在应用程序执行的整个持续期间进行记录并覆盖 SQL_OPT_TRACE。
- 如果正在使用 ODBC 驱动程序管理器，则设置 SQL_OPT_TRACE 会开启驱动程序管理器跟踪，但对驱动程序跟踪无任何影响。
- 客户端应用程序可以在直接链接到驱动程序时使用空值句柄，或在使用驱动程序管理器跟踪时使用分配的句柄。
- `log4cplus` 配置文件不能与 SQL_OPT_TRACE 一起使用。

TDS 协议捕获的动态控制

Adaptive Server Enterprise ODBC 驱动程序的新 SQL_ATTR_TDS_CAPTURE 连接属性可以实现 TDS 协议捕获的暂停 (SQL_CAPTURE_PAUSE) 和恢复 (SQL_CAPTURE_RESUME)。

```
// pause protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_PAUSE, SQLINTEGER);

// resume protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_RESUME, SQLINTEGER);
```

缺省情况下，在为连接设置 ProtocolCapture 连接属性后，会在连接的持续期间执行 TDS 协议捕获操作。 使用 SQL_ATTR_TDS_CAPTURE（已设置 ProtocolCapture 连接属性的情况下），应用程序可针对所需的程序执行段有选择性地暂停和恢复 TDS 协议捕获。

可在分配连接句柄之后设置 SQL_ATTR_TDS_CAPTURE。 可以在建立连接之前或针对未使用 TDS 协议捕获的连接暂停或恢复 TDS 协议捕获。 驱动程序可能会延迟 TDS 协议捕获的暂停或恢复操作，以保证捕获流的完整性。 这可确保写入完整 PDU 包以精确捕获 Ribo 和其它协议转换器实用程序所占用的资源。

对于需要捕获连接的所有 TDS 包的应用程序，请勿设置 SQL_ATTR_TDS_CAPTURE。

Replication Server 连接支持

Adaptive Server Enterprise ODBC 驱动程序可连接到 Replication Server® 以监控和管理服务器。

Replication Server 仅支持由 ODBC 驱动程序发送的有效 Replication Server 管理命令。将 **BackEndType** 连接属性设置为 `Replication Server`，以进行 Replication Server 连接。

综合的 ADO.NET 提供程序程序集文件

从 ESD #4 开始，Adaptive Server Enterprise ADO.NET 数据提供程序只有两个提供程序程序集文件，每个均包含所有功能。

- `Sybase.AdoNet2.AseClient.dll` - 支持 .NET 2.0、.NET 3.0 和 .NET 3.5 的功能。
- `Sybase.AdoNet4.AseClient.dll` - 支持 .NET 4.1 及更高版本的功能。

这些文件的 32 位版本安装在 `C:\Sybase\DataAccess\ADONET\dll` 目录中，64 位版本安装在 `C:\Sybase\DataAccess64\ADONET\dll` 目录中。

更新引用任一过时 DLL 的所有构建脚本或部署脚本。

ADO.NET 对更大小数精度/标度的支持

Adaptive Server Enterprise ADO.NET 数据提供程序现在支持 AseDecimal - 一种可支持精度/标度 78 的结构。

Adaptive Server 数值和小数数据类型支持的最大精度/标度为 38，算术运算结果可支持的最大精度/标度为 78，而 .NET Framework 小数数据类型可支持的最大精度/标度为 28。 这将导致的情况是：在将 AdaptiveServer 数值和小数类型的数据或算术运算的结果读入到 .NET Framework 小数类型时，数据会溢出。

Adaptive Server Enterprise ADO.NET 数据提供程序现在支持 AseDecimal - 一种可支持精度/标度 78 的结构。 要使用 AseDecimal 结构检索数值或小数值，请将新 UseAseDecimal 连接属性设置为 1。 缺省情况下，UseAseDecimal 将设置为 0，表示不使用 AseDecimal 结构。

为额外连接属性进行的 Visual Studio DDEX“连接”(Connection) 对话框增强

Adaptive Server Enterprise ADO.NET 数据提供程序现在允许您在 Visual Studio DDEX 的“添加连接”(Add Connection) 对话框中添加额外的连接属性。

- 可将连接属性指定为以分号 (;) 分隔的列表。
- 最后一个连接属性无需以分号 (;) 终止。
- 不具有值的属性将被忽略。

当前，不存在标记错误连接规范的警告或错误消息。

OLE DB 应用程序的新连接字符串

将介绍 OLE DB 应用程序的一组新连接字符串。

属性名称	说明	必需	缺省值
ProtocolCapture	启用该属性可捕获 OLE DB 应用程序和服务器之间的通信。 请参见《Adaptive Server Enterprise OLE DB 提供程序用户指南》。	否	Empty
RetryCount、RetryDelay	控制连接的重试行为。 **RetryCount** 是在报告连接失败之前尝试连接到服务器的次数。在两次重试之间，驱动程序延迟的秒数为 **RetryDelay**。 缺省情况下，OLE DB 应用程序不会重试连接。 也可以在 SQL.INI 和 LDAP 接口中指定这些值： • 可将 **RetryCount** 在 SQL.INI 中指定为 **RetryCount**、在 LDAP 中指定为 **sybaseRetryCount**。 • 可将 **RetryDelay** 在 SQL.INI 中指定为 **LoopDelay**、在 LDAP 中指定为 **sybaseRetryDelay**。	否	0
SuppressControlTokens	指定 SAP Adaptive Server 不应发送 TDS_CONTROL 令牌。 值： • 0 - 在可能的情况下，强制 Adaptive Server 发送 TDS_CONTROL 令牌。 • 1 - 缺省值；强制 Adaptive Server 取消发送 TDS_CONTROL 令牌。	否	1
SuppressParamFormat	指定 OLE DB 应用程序仅应在格式更改时发送参数格式令牌。 值： • 0 - 强制 OLE DB 应用程序始终在每次执行时发送参数格式令牌。 • 1 - 缺省值；请求 OLE DB 应用程序在已设置格式后，取消发送参数格式令牌。	否	1
SuppressRowFormat	指定 Adaptive Server 仅应在首次执行或格式更改时发送行格式令牌。 值： • 0 - 强制 Adaptive Server 在每次执行时都发送格式信息。 • 1 - 缺省值；请求 Adaptive Server 在可能的情况下取消发送行格式令牌。	否	1

属性名称	说明	必需	缺省值
SuppressRow-Format2	指定 Adaptive Server 应在可能的情况下使用 TDS_ROWFMT 字节序列而非 TDS_ROWFMT2 字节序列来发送数据。 值: • 0 - 缺省值；强制 Adaptive Server 在可能的情况下以 TDS_ROWFMT2 格式发送数据。 • 1 - 强制 Adaptive Server 在可能的情况下以 TDS_ROWFMT 格式发送数据。 请参见《Adaptive Server Enterprise OLE DB 提供程序用户指南》。	否	0

ESD #4 中用于 Python 的 Adaptive Server Enterprise 扩展模块

用于 Python 的 Adaptive Server 扩展模块已得到增强，现可支持用于动态语句和存储过程的新参数数据类型。

动态语句和存储过程的新参数数据类型支持

从 ESD #4 开始，用于 Python 的 Adaptive Server Enterprise 扩展模块支持小数数据类型、货币数据类型以及 LOB 作为动态语句和存储过程的参数。

用于 Python 的 Adaptive Server Enterprise 扩展模块还支持对存储过程使用 date、time、datetime 和 float 参数。

请参见《用于 Python 的 Adaptive Server Enterprise 扩展模块程序员指南》。

ESD #4 中用于 PHP 的 Adaptive Server Enterprise 扩展模块

从 ESD #4 开始，用于 PHP 的 Adaptive Server Enterprise 扩展模块具有一整套用于开发应用程序的 API。

API 类型	API	说明
连接:	**sybase_close()**	关闭与 ASE 的指定连接。
	sybase_connect()	打开与 ASE 的连接。
	sybase_pconnect()	（**新增**）打开与 ASE 的持久连接。
查询:	**sybase_affected_rows()**	（**新增**）返回上次在指定连接上插入、删除或更新查询所影响的行数。

API 类型	API	说明
	sybase_query()	向指定连接发送查询。 将自动获取和缓冲完整结果集。
	sybase_unbuf-fered_query()	（**新增**）向指定连接发送查询。 不会像使用 **sybase_query()** 一样自动获取和缓冲完整结果集。
远程过程调用：	**sybase_rpc_bind_param_ex**	（**新增**）将 PHP 变量绑定到远程过程参数。
	sybase_rpc_exe-cute	（**新增**）执行通过 **sybase_rpc_init()** 初始化的远程过程调用。
	sybase_rpc_init	（**新增**）返回语句标识符，该标识符指向针对连接上的远程过程初始化的语句。
结果集：	**sybase_data_seek()**	（**新增**）移动与结果标识符相关联的结果集的内部行指针，使其指向指定行号。
	sybase_fetch_ar-ray()	（**新增**）采用关联数组和/或数值数组的形式获取结果行。
	sybase_fetch_as-soc()	从与关联数组中的指定结果标识符相关联的结果集中获取一行数据。
	sybase_fetch_field()	（**新增**）返回包含字段信息的对象。
	sybase_fetch_ob-ject()	（**新增**）从与指定结果标识符相关联的结果集中获取一行数据作为对象。
	sybase_fetch_row()	（**新增**）从与数值数组中的指定结果标识符相关联的结果集中获取一行数据。
	sybase_field_seek()	（**新增**）设置指向请求的字段偏移的内部指针。
	sybase_free_re-sult()	释放与结果集相关联的全部内存。
	sybase_next_re-sult()	（**新增**）返回指向连接上下一结果集的结果集标识符。
	sybase_num_fields()	（**新增**）返回结果集中字段的数量。
	sybase_num_rows()	（**新增**）返回 **select** 语句的结果集中行的数量。
	sybase_use_result	（**新增**）存储连接上最后一次未缓冲查询的结果集，并返回指向该存储结果集的结果集标识符。

API 类型	API	说明
其它:	**sybase_get_last_message()**	（**新增**）返回服务器返回的最后一条消息。
	sybase_get_last_status	（**新增**）返回在连接上发送的最后一个状态结果。
	sybase_select_db()	（**新增**）设置连接资源所引用的服务器上的当前活动数据库。
	sybase_set_message_handler()	（**新增**）设置将在收到客户端消息或服务器消息时调用的用户定义的回调函数。

请参见《用于 PHP 的 Adaptive Server Enterprise 扩展模块程序员指南》。

ESD #4 中用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序

在 ESD #4 中，用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序具有以下改进功能。

请参见《用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序程序员指南》。

- 新增数据库处理属性
- 新增缺省日期转换和显示格式支持使用新的 **_data_fmt** 私有方法
- 新增 LONG/BLOB 数据处理支持

  现在，用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序支持 image 和 text 类型的 LONG/BLOB 数据。 每种类型均最多可容纳 2GB 的二进制数据。
- 新增自动生成密钥支持

  现在，用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序支持用于自动生成密钥的 IDENTITY 功能。 声明带有 IDENTITY 列的表会针对每项插入生成一个新值。这些值单调递增，但不能保证连续递增。 要获取上次插入生成及使用的值:

  ```
  SELECT @@IDENTITY
  ```
- 新增参数绑定支持

  现在，用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序直接支持参数绑定。仅支持“?”样式的参数；不支持“:1”占位符类型的参数。 不支持绑定 text 或 image 数据类型的参数。
- 新增对带输入和输出参数的存储过程的支持

ESD #3 的新功能

ESD #3 针对 Open Client 15.7 和 Open Server 15.7 以及用于 Python 15.7 的 Adaptive Server Enterprise 扩展模块引入了新功能。

跳过示例文件、文档文件和调试文件的安装

从 ESD#3 开始，您可选择跳过示例文件、文档文件和调试文件的安装。

缺省情况下，在安装 Open Server 和 SDK 时安装这些文件。 为了跳过这些文件的安装：

- 在 GUI、主控台和无提示模式下安装时，使用新的 `-DPRODUCTION_INSTALL=TRUE` 安装程序命令行参数。
- 在无提示模式下安装时，使用响应文件中的新 **PRODUCTION_INSTALL=TRUE** 属性。

ESD #3 中的 Open Client 15.7 和 Open Server 15.7 功能

ESD #3 中的新功能包括 64 位 Microsoft Windows 中的 CyberSafe Kerberos 驱动程序以及脚本化语言增强、UNIX 命名的套接字和拒绝记录行。

64 位 Microsoft Windows 上的 CyberSafe Kerberos 驱动程序

Open Client 和 Open Server 包含 `libsybskrb64.dll`，后者是 Microsoft Windows x86-64 64 位的 64 位 CyberSafe Trustbroker Kerberos 驱动程序库。

`libsybskrb64.dll` 位于 `%SYBASE%\%SYBASE_OCS%\dll` 中；其行为类似于 32 位 CyberSafe TrustBroker Kerberos 驱动程序库 `libsybskrb.dll`。

UNIX 命名的套接字

该功能为 Open Client 和 Open Server 中的 UNIX 命名的套接字提供支持。 此类套接字也被称为 UNIX 域套接字。

该功能允许使用 UNIX 命名的套接字在主机内部实现更快通信，因为在进程间通信时无需遍历 TCP 堆栈。要启用该功能，请向目录服务层中添加条目以指定 `afunix`（而非 `tcp`）作为传输类型。

例如，传统 interfaces 文件条目可能如下所示：

```
MYSERVER

master tcp unused myhost 8600
query tcp unused myhost 8600
```

在仍将 TCP 用于远程客户端时，要将 UNIX 命名的套接字而非 TCP 用于本地客户端，以上条目将变成：

MYSERVER

```
master afunix unused //myhost/tmp/MYSERVER.socket
query afunix unused //myhost/tmp/MYSERVER.socket
master tcp unused myhost 8600
query tcp unused myhost 8600
```

记录客户端拒绝的行

已经添加名为 **--clienterr errorfile** 的新 `bcp` 选项，它用于因客户端检测的错误（如转换或格式错误）而拒绝行时在错误文件中记录已拒绝行及其相关联的错误消息。

如果使用 `--clienterr` 选项而未使用 `-e` 选项，则将客户端错误消息写入错误文件中。 但是，不会将服务器错误消息写入错误文件中。

如果同时使用 `--clienterr` 选项和 `-e` 选项，**bcp** 不会继续进行拷入和拷出操作。

增加的 bcp 最大行数处理量

bcp 可处理的最大行数已从 *INT32_MAX* 增加至 *UINT64_MAX*（即 18446744073709551615）。

参数格式抑制

现在，Open Client 支持对 Adaptive Server Enterprise 中的动态语句的参数格式抑制。

ESD #3 中用于 Python 的 Adaptive Server Enterprise 扩展模块

用于 Python 的 Adaptive Server Enterprise 扩展模块已经得到增强，其支持带输入和输出参数、计算行和本地化错误消息的存储过程。

使用 Python 访问存储过程

用于 Python 的 Adaptive Server Enterprise 扩展模块增加了向存储过程传递输入和输出参数的支持。

使用游标对象的 **callproc()** 方法调用存储过程。 如果执行存储过程时出现错误，则 **callproc()** 会抛出异常，且您可使用 `proc_status` 属性检索状态值。 该支持是对 Python DBAPI 规范的扩展。

以下是包含多行结果的 Python 应用程序示例：

```
import sybpydb
#Create a connection.
conn = sybpydb.connect(user='sa')
# Create a cursor object.
```

```
cur = conn.cursor()
# Call the stored procedure
try:
    cur.callproc('myproc')
    continue = True
    while(continue == True):
        row = cur.fetchall()
        continue = cur.nextset()
except sybpydb.Error:
    print("Status=%d" % cur.proc_status)
```

为了指定输出参数，扩展模块提供了 **OutParam** 构造方法。 该支持是对 Python DBAPI 规范的扩展。 **callproc()** 方法返回向该方法传递的所有参数的列表。 如果存在输出参数但存储过程未生成结果集，在 **callproc()** 完成后，列表则包含修改的输出值。 但是，如果存在结果集，在使用 **fetch*()** 方法检索存储过程的所有结果集并调用 **nextset()** 检查是否存在任何其它结果集之前，列表不含修改的输出值。 即使预计只有一个结果集，也必须调用 **nextset()** 方法。

以下是带有输出参数的示例 Python 应用程序:

```
import sybpydb
#Create a connection.
conn = sybpydb.connect(user='sa')
# Create a cursor object.
cur = conn.cursor()
cur.execute("""
    create procedure myproc
    @int1 int,
    @int2 int output
    as
    begin
        select @int2 = @int1 * @int1
    end
    """)
int_in = 300
int_out = sybpydb.OutParam(int())
vals = cur.callproc('pyproc', (int_in, int_out))
print ("Status = %d" % cur.proc_status)
print ("int = %d" % vals[1])
cur.connection.commit()
# Remove the stored procedure
cur.execute("drop procedure myproc")
cur.close()
conn.close()
```

示例程序 `callproc.py` 中提供了不同输出参数类型的更多示例。

使用 Python 的计算行

用于 Python 的 Adaptive Server Enterprise 扩展模块添加了对计算行的支持。

示例程序 `compute.py` 中提供了计算行的处理示例。

本地化错误消息

现在，用于 Python 的 Adaptive Server Enterprise 扩展模块支持本地化错误消息。

ESD #1 的新功能

ESD #1 针对 Open Client 15.7、Open Server 15.7、SDK 15.7 和用于 Python 15.7 的 Adaptive Server Enterprise 扩展模块引入了新功能。

ESD #1 中的 Open Client 15.7 和 Open Server 15.7 功能

ESD #1 的新功能包括 FIPS 认证的 SSL 过滤器和对 64 位 Windows 上用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序和用于 PHP 的 Adaptive Server Enterpris 扩展模块的支持。

FIPS 认证的 SSL 过滤器

现在，Sybase SSL 过滤器为美国联邦信息处理标准 (FIPS) 140-2，它可与支持 Certicom SSL 的平台兼容。

- HP-UX Itanium 32 位
- HP-UX Itanium 64 位
- IBM AIX 32 位
- IBM AIX 64 位
- Linux x86 32 位
- Linux x86-64 64 位
- Linux on POWER 32 位
- Linux on POWER 64 位
- Microsoft Windows x86 32 位
- Microsoft Windows x86-64 64 位
- Solaris SPARC 32 位
- Solaris SPARC 64 位
- Solaris x86 32 位
- Solaris x86 64 位

Linux on POWER 32 位和 64 位的共享对象 SSL 过滤器文件的名称已经分别从 `libsybfcsissl.so` 和 `libsybfcsissl64.so` 改为 `libsybfssl.so` 和 `libsybfssl64.so`。 `libtcl.cfg` 示例文件也已经更新：

```
[FILTERS]
;ssl=libsybfssl.so
```

Microsoft Windows x86-64 64 位的 SSL 过滤器 DLL 的名称已经从 `libsybfcsissl64.dll` 改为 `libsybfssl64.dll`。 `libtcl64.cfg` 示例文件也已经更新：

```
[FILTERS]
;ssl=libsybfssl64
```

64 位 Windows 支持用于 Perl 的 ASE 数据库驱动程序和用于 PHP 的 ASE 扩展模块

Microsoft Windows 64 位平台现已支持将用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序与 ActivePerl 5.14.1 和 DBI 1.616 配合使用。

Microsoft Windows 64 位平台现已支持将用于 PHP 的 Adaptive Server Enterprise 扩展模块与 PHP 5.3.6 版配合使用。

ESD #1 中针对 jConnect、Adaptive Server 驱动程序和提供程序的 SDK 15.7 功能

ESD #1 引入了对抑制参数格式元数据和行格式元数据的支持，从而提高性能。

抑制参数格式元数据以提高预准备语句性能

可以在重新执行预准备语句时通过抑制参数格式元数据提高采用 ODBC 驱动程序时预准备语句的性能。

Adaptive Server 15.7 ESD#1 及更高版本支持参数格式元数据抑制。
请将 DynamicPrepare 连接属性设置为 1，然后使用 SuppressParamFormat 连接字符串属性。

SuppressParamFormat 连接字符串属性的有效值为：
- 0 - 不在预准备语句中抑制参数格式元数据。
- 1 - 缺省值；在可能情况下均抑制参数格式元数据。

注意： 仅当连接的 Adaptive Server 支持该功能时，才可在预准备语句中抑制参数格式元数据。 如果将 DynamicPrepare 和 SuppressParamFormat 参数均设置为 1，但连接的 Adaptive Server 不支持参数格式元数据抑制，Adaptive Server 将忽略参数设置。

示例

该 ODBC 连接字符串在预准备语句中抑制参数格式元数据：

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;SuppressParam
Format=1;
```

抑制行格式元数据以提高查询性能

当会话中需要重新执行查询时，可通过抑制行格式元数据 (TDS_ROWFMT 或 TDS_ROWFMT2) 来提高使用 ODBC 驱动程序和 ADO.NET 数据提供程序重复执行查询的性能。

Adaptive Server 15.7 ESD#1 及更高版本支持行格式元数据抑制。

使用 SuppressRowFormat 连接字符串属性。

SuppressRowFormat 连接字符串属性的有效值为：
- 0 - 不抑制行格式元数据。
- 1 - 缺省值；Adaptive Server 尽可能不发送行格式元数据。

注意： 仅当连接的 Adaptive Server 支持行格式元数据抑制时，才可以使用该功能。如果将 SuppressRowFormat 参数设置为 1，但连接的 Adaptive Server 不支持行格式元数据抑制，Adaptive Server 将忽略该参数设置。

示例

该 ODBC 连接字符串抑制行格式元数据：

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressRowFormat=1;
```

SuppressRowFormat2 和 SQLBulkOperations

请勿将 SuppressRowFormat2 连接字符串属性与使用 **SQLBulkOperations** API 的 ODBC 程序配合使用。

启用 SuppressRowFormat2 会抑制 **SQLBulkOperations** 需要的信息，并导致错误。

ESD #1 中用于 Python 的 Adaptive Server Enterprise 扩展模块

自 ESD #1 起，用于 Python 的 Adaptive Server Enterprise 扩展模块支持 Python 2.6、2.7 和 3.1 版本。

可以从 SDK 安装程序中安装用于 Python 的 Adaptive Server Enterprise 扩展模块。 有关安装说明，请参见《软件开发工具包和 Open Server 安装指南》和《软件开发工具包和 Open Server 发行公告》。 有关使用用于 Python 的 Adaptive Server Enterprise 扩展模块的信息，请参见《用于 Python 的 Adaptive Server Enterprise 扩展模块程序员指南》。

配置用于 Python 的 Adaptive Server Enterprise 扩展模块

在缺省安装目录路径中设置 PYTHONPATH 或 Python 变量 *sys.path* 以在应用程序中使用用于 Python 的 Adaptive Server Enterprise 扩展模块。

Python 模块搜索路径

Python 在 Python 变量 *sys.path* 指定的目录列表中搜索导入的模块。

sys.path

sys.path 变量在应用程序所在的目录中初始化，并且位于环境变量 PYTHONPATH（使用的语法与 shell 变量 PATH 相同，即目录名列表）指定的目录列表中。

如果未设置 PYTHONPATH 或者如果找不到模块文件，将在与安装有关的缺省路径中继续搜索。 要在应用程序中使用用于 Python 的 Adaptive Server Enterprise 扩展模块，

必须将 PYTHONPATH 或 Python 变量 *sys.path* 设置为以下目录路径（不同版本的 Adaptive Server Python 扩展模块的缺省安装目录）之一：

平台	Python 版本	缺省安装路径
Windows	2.6	`$SYBASE\$SYBASE_OCS\python\python26_64\dll`
	2.7	`$SYBASE\$SYBASE_OCS\python\python27_64\dll`
	3.1	`$SYBASE\$SYBASE_OCS\python\python31_64\dll`
所有其它平台	2.6、2.7	`$SYBASE/$SYBASE_OCS/python/python26_64r/lib`
	3.1	`$SYBASE/$SYBASE_OCS/python/python31_64r/lib`

Open Client 15.7 和 Open Server 15.7 功能

Open Client 和 Open Server 15.7 版引入了一些新功能，例如支持大对象 (LOB) 定位符、行内和行外 LOB 以及其它许多内容。

大对象定位符支持

LOB 定位符包含指向 Adaptive Server 中的 LOB 数据的逻辑指针，而不是 LOB 数据；因而，减少了通过网络在 Adaptive Server 及其客户端之间传输的数据量。

Adaptive Server 15.7 包括使用 LOB 定位符对 LOB 数据进行运算的 Transact-SQL 命令和函数。 可以从 Client-Library 中将这些命令和函数当作语言命令调用。 请参见《Adaptive Server Enterprise Transact-SQL 用户指南》中的第 21 章“行内、行外 LOB”。

Client-Library 更改

CS_LOCATOR 数据类型支持 LOB 定位符。 **cs_locator_alloc()** 和 **cs_locator_drop()** API 为 CS_LOCATOR 变量分配和释放内存。 已经添加 **cs_locator()**，以便检索 CS_LOCATOR 变量的信息。

Client-Library 例程 **cs_convert()** 和 **ct_bind()** 已经得到增强，可以处理 CS_LOCATOR 变量。

CS_LOCATOR

CS_LOCATOR 是 opaque 数据类型，用于存储定位符值和可选预取数据。

将传入定位符与 CS_LOCATOR 变量绑定之前使用 **cs_locator_alloc()** 为该变量分配内存，否则，会发生错误。 当不再需要该变量时，可使用 **cs_locator_drop() 释放其内存。**

CS_LOCATOR 变量可以重复使用，但 Adaptive Server 中的当前定位符值仅在事务结束之前有效。

CS_LOCATOR 的类型常量包括：

- CS_TEXTLOCATOR_TYPE - 用于 `text` LOB。
- CS_IMAGELOCATOR_TYPE - 用于 `image` LOB。
- CS_UNITEXTLOCATOR_TYPE - 用于 `unitext` LOB。

使用 **cs_convert()** 从 CS_LOCATOR 变量中检索定位符的预取数据和定位符值的字符表示。 将 CS_LOCATOR 转换为 CS_CHAR 可以字符串的形式返回定位符的十六进制值。 将定位符转换为 CS_TEXT_TYPE、CS_IMAGE_TYPE 或 CS_UNITEXT_TYPE 可返回定位符的预取数据。

支持的 LOB 定位符转换
该表列出了 LOB 定位符转换。

	CS_TEXT_ LOCATOR	CS_IMAGE_ LOCATOR	CS_UNITEXT_ LOCATOR
CS_CHAR_TYPE	X	X	X
CS_TEXT_TYPE	X		
CS_IMAGE_TYPE		X	
CS_UNITEXT_TYPE			X
CS_TEXT_LOCATOR	X		
CS_IMAGE_LOCATOR		X	
CS_UNITEXT_LOCATOR			X
图例： X = 支持的转换。			

处理定位符数据类型时：

- **ct_bind()** 忽略 CS_DATAFMT 的 *maxlength* 值，因为 Client-Library 认为定位符数据类型的长度是固定的。 任何通过定位符发送的可选预取数据所需的内存都是在内部针对其整个长度分配的。 *maxlength* 值不影响预取数据的长度。
- 可将传入的 LOB 定位符绑定到 CS_CHAR_TYPE。 但不能直接将定位符绑定到 CS_TEXT_TYPE、CS_IMAGE_TYPE 或 CS_UNITEXT_TYPE。

cs_locator()

从 CS_LOCATOR 变量中检索信息，如预取数据、服务器中的 LOB 的总长度或定位符指针的字符表示。

语法

```
CS_RETCODE cs_locator(ctx, action, locator, type, buffer, buflen,
   outlen)

CS_CONTEXT        *ctx;
CS_INT                   action;
CS_LOCATOR        *locator;
CS_INT                   type;
CS_VOID                *buffer;
CS_INT                   buflen;
CS_INT                     *outlen;
```

参数

- *ctx* - 指向 CS_CONTEXT 结构的指针。

- *action* - 指定是设置还是检索信息。 当前，仅允许 CS_GET 操作。
- *locator* - 指向定位符变量的指针。
- *type* - 要检索或设置的信息的类型。 符号值：

值	操作	***buffer* 指向**	说明
CS_LCTR_LOBLEN	CS_GET	CS_BIGINT	检索服务器中的 LOB 数据的总长度。
CS_LCTR_LOCATOR	CS_GET	CS_CHAR	以字符串形式检索定位符值。
CS_LCTR_PREFETCHLEN	CS_GET	CS_INT	检索定位符变量中包含的预取 LOB 数据的长度。
CS_LCTR_PREFETCHDATA	CS_GET	CS_CHAR	检索定位符变量中包含的预取 LOB 数据。
CS_LCTR_DATATYPE	CS_GET	CS_INT	检索定位符类型。 有效的返回类型为 CS_TEXTLOCATOR_TYPE、CS_IMAGELOCATOR_TYPE 和 CS_UNITEXTLOCATOR_TYPE。

- *buffer* - 指向将数据存储到的变量的指针。 字符数据以 NULL 结尾。
- *buflen* - **buffer* 长度（以字节为单位）。
- *outlen* - 指向 CS_INT 变量的指针。 如果 *outlen* 不是 NULL，**cs_locator()** 会将 **outlen* 设置为在 **buffer* 中放置的数据的长度（以字节为单位）。 如果返回的数据是字符数据（例如，预取数据或定位符字符串），则 **outlen* 中返回的长度会将 NULL 终止符计算在内。 如果 **cs_locator()** 返回 CS_TRUNCATED 并且 *outlen* 不是 NULL，则 **cs_locator()** 在 **outlen* 中返回必需的缓冲区大小。

返回内容

返回值	含义
CS_SUCCEED	例程成功完成。
CS_TRUNCATED	因为缓冲区太小，结果被截断。
CS_FAIL	例程失败。

cs_locator_alloc()

分配 CS_LOCATOR 数据类型结构。

语法

```
CS_RETCODE cs_locator_alloc(ctx, locator)

CS_CONTEXT *ctx;
CS_LOCATOR **locator;
```

参数

- *ctx* - 指向 CS_CONTEXT 结构的指针。
- *locator* - 要分配的定位符变量的地址。 将 **locator* 设置为新分配的 CS_LOCATOR 结构的地址。

返回内容

返回值	含义
CS_SUCCEED	例程成功完成。
CS_FAIL	例程失败。

cs_locator_drop()
释放 CS_LOCATOR 数据类型结构。

语法

```
CS_RETCODE cs_locator_drop(ctx, locator)

CS_CONTEXT *ctx;
CS_LOCATOR *locator;
```

参数

- *ctx* - 指向 CS_CONTEXT 结构的指针。
- *locator* - 指向要释放的定位符变量的指针。

返回内容

返回值	含义
CS_SUCCEED	例程成功完成。
CS_FAIL	例程失败。

isql 增强
isql 以十六进制字符形式显示 LOB 定位符值。 在 CS_LOCATOR 中存储的预取数据不会显示。

示例
将 LOB 数据转换为定位符，并显示定位符值：

```
1> set send_locator on
2> go

1> select * from testable
2> go
```

```
charcol         textcol
--------------- ------------------------------------------------
Hello           0x48656c6c6f20576f726c642e204865726520492061 6d2e2e
```

Open Server 支持大对象定位符

Server-Library 已经新增了 LOB 定位符功能，供 Open Server 应用程序将 LOB 定位符语言命令从客户端传递到后端服务器。

为将 LOB 定位符从服务器传递到客户端应用程序，Open Server 应用程序为 *CS_LOCATOR* 变量分配内存，而且从服务器绑定并接收 LOB 信息。

srv_bind() 和 **srv_descfmt()** 已经得到增强，可以处理 CS_TEXT_LOCATOR_TYPE、CS_IMAGE_LOCATOR_TYPE 和 CS_UNITEXT_LOCATOR_TYPE。

大对象定位符支持

以下连接功能表示支持发送和接收 LOB 定位符。

- CS_DATA_LOBLOCATOR - 在通过 CS_VERSION_157 初始化客户端应用程序时隐式设置的只读请求功能，表示 Client-Library 可以向服务器发送 LOB 定位符。
- CS_DATA_NOLOBLOCATOR - 一种响应功能，客户端应用程序设置为通知服务器不要发送 LOB 定位符，即使基础 Client-Library 支持它们也是如此。

从服务器中请求 LOB 定位符

缺省情况下，当选择 LOB 列或值时，Adaptive Server 发送 LOB 数据而非 LOB 定位符，无论是否支持协商的 LOB 定位符都是如此。

要显式请求 LOB 定位符或请求预取数据，请使用 **ct_options()** 设置以下查询处理选项：

- CS_OPT_LOBLOCATOR - 布尔值，如果设置为 CS_TRUE，则请求服务器返回定位符而非 LOB 值。 在向服务器发送查询之前，先设置此选项。 缺省值为 **CS_FALSE**。
- CS_OPT_LOBPREFETCHSIZE - 整数，指定服务器必须发送的预取数据的大小。对于 image 定位符，此大小表示预取数据的字节数；对于 text 和 unitext 定位符，则表示字符数。

  CS_OPT_LOBPREFETCHSIZE 的缺省值为 0，表示通知服务器不要发送预取数据。值为 -1 时，会在整个 LOB 数据中检索请求的 LOB 及其定位符。

定位符值和可选预取数据均存储在 CS_LOCATOR 数据类型中。 客户端必须在请求获得定位符数据之前为 CS_LOCATOR 变量分配内存。

示例

检索需要截断的文本值的 LOB 定位符。 有关更多代码示例，请参见《Open Client Client-Library/C 参考手册》。

```
CS_LOCATOR *lobloc;
CS_INT     prefetchsize;
CS_BOOL    boolval;
```

```
CS_INT     start, length;
CS_INT     outlen;
CS_CHAR    charbuf[1024];
CS_BIGINT  totallen;
...

/*
** Turn on option CS_LOBLOCATOR first and set the prefetchsize to
100.
*/boolval = CS_TRUE;
ct_options(conn, CS_SET, CS_OPT_LOBLOCATOR, &boolval, CS_UNUSED,
NULL);
prefetchsize = 100;
ct_options(conn, CS_SET, CS_OPT_LOBPREFETCHSIZE, &prefetchsize,
CS_UNUSED,
   NULL);

/*
** Allocate memory for the CS_LOCATOR.
*/
cs_locator_alloc(ctx, &lobloc);

/*
** Open a transaction and get the locator. The locator is only valid
within a
** transaction.
*/
sprintf(cmdbuf, “begin transaction \
   select au_id, copy from pubs2..blurbs where au_id \
   like ‘486-29-%’ ”);
ct_command(cmd, CS_LANG_CMD, cmdbuf , CS_NULLTERM, CS_UNUSED);
ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
{
   ...
}
   /*
   ** Bind the locator and fetch it.
   */
   strcpy(prmfmt.name, "@locatorparam");
   prmfmt.namelen = CS_NULLTERM;
   prmfmt.datatype = CS_TEXTLOCATOR_TYPE;
   prmfmt.maxlength = CS_UNUSED;
   ...

   ct_bind(cmd, 1, &fmt, lobloc, NULL, &indicator);
   ct_fetch(cmd, CS_UNUSED, CS_UNUSED, CS_UNUSED, &count);
}

/*
** Use the cs_locator() routine to retrieve data from the fetched
```

```
locator.
** Get the prefetch length and the prefetch data.
*/
cs_locator(ctx, CS_GET, lobloc, CS_LCTR_PREFETCHLEN, (CS_VOID
*)&prefetchsize,
   sizeof(CS_INT), &outlen);

cs_locator(ctx, CS_GET, lobloc, CS_LCTR_PREFETCHDATA, (CS_VOID
*)charbuf,
   sizeof(charbuf), &outlen);

/*
** Retrieve the total length of the LOB data in the server for this
** locator.
*/
cs_locator(ctx, CS_GET, lobloc, CS_LCTR_LOBLEN,(CS_VOID *)&totallen,
   sizeof(totallen), &outlen);

/*
** Use the retrieved locator to perform an action to the LOB, pointed
to by
** this locator in the server.
**
** Get a substring from the text in the server, using a parameterized
language
** command.
*/
start = 10;
length = 20;
sprintf(cmdbuf, “select return_lob(text, substring(@locatorparam, \
   start, length))” );
ct_command(cmd, CS_LANG_CMD, cmdbuf, CS_NULLTERM, CS_UNUSED);

/*
** Set the format structure and call ct_param()
*/
strcpy(prmfmt.name, "@locatorparam");
prmfmt.namelen = CS_NULLTERM;
prmfmt.datatype = CS_TEXTLOCATOR_TYPE;
prmfmt.format = CS_FMT_UNUSED;
prmfmt.maxlength = CS_UNUSED;
prmfmt.status = CS_INPUTVALUE;

indicator = 0;
ct_param(cmd, &prmfmt, (CS_VOID *)lobloc, CS_UNUSED, indicator);

/*
** Send the locator commands to the server.
*/
ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
```

```
{
   ...
}

/*
** Truncate the text to 20 bytes and commit the transaction.
*/
sprintf(cmdbuf, “truncate lob @locatorparam (length) \
   commit transaction”);
ct_command(cmd, CS_LANG_CMD, cmdbuf, CS_NULLTERM, CS_UNUSED);
ct_param(cmd, &prmfmt, (CS_VOID *)lobloc, CS_UNUSED, indicator);

ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
{
   ...
}

/*
** The transaction is closed, deallocate the locator.
*/
cs_locator_drop(ctx, lobloc);
```

行内和行外 **LOB** 支持

Bulk-Library 15.7 版支持在 Adaptive Server 中对 `text`、`image` 和 `unitext` 大对象 (LOB) 列进行行内存储。

在 Adaptive Server 15.7 中，当行中的可用空间足够时，标记为行内存储的 LOB 列存储在行内。 只有绑定的 LOB 数据可以写入行内。 **bcp** 实用程序绑定 LOB 数据，因此发送行内 LOB 数据（如果适用）。 请参见《Adaptive Server Enterprise Transact-SQL 用户指南》中的第 21 章“行内、行外 LOB”。

Bulk-Library Select into 记录

为处理向代理表中插入行的 **select into existing table** 语句，Adaptive Server 使用 Bulk-Library 生成批量复制操作。

但是，无法完全记录常规批量复制操作。 BLK_CUSTOM_CLAUSE 属性使 Adaptive Server 能够区分一般批量复制操作和影响代理表的 **insert into** 语句导致的批量复制操作。 然后，可以通过由 BLK_CUSTOM_CLAUSE 属性指定的自定义子句来附加此类 **insert into** 语句导致的批量复制操作。 Adaptive Server 可以检测该语句并进行完全记录。

BLK_CUSTOM_CLAUSE

应用程序可以使用 **blk_props** Bulk-Library 例程来设置或检索 BLK_CUSTOM_CLAUSE。

表 7. Client/Server BLK_CUSTOM_CLAUSE 属性

属性名称	说明	***buffer* 是**	适用于	注释
BLK_CUS-TOM_ CLAUSE	在 **insert bulk** 命令的现有 **with** 子句之后添加的特定于应用程序的自定义的 SQL 子句。	一个包含自定义子句的字符串。	仅限 IN 复制	只有支持自定义 SQL 子句的服务器版本支持此属性。 目前仅供内部产品使用。

- 只有将 Adaptive Server **select into/bulkcopy/pllsort** 数据库选项设置为 on 时，才可使用 **select into** 操作。
- 为了完全记录 **select into** 操作，必须将 Adaptive Server **full logging for select into** 数据库选项设置为 on。

示例

采用 **blk_props** 设置 BLK_CUSTOM_CLAUSE：

```
blk_props(blkdesc, CS_SET, BLK_CUSTOM_CLAUSE,
(CS_VOID *)"from select_into", CS_NULLTERM, NULL);
```

Adaptive Server 产生附加了指定自定义子句的批量复制操作：

```
insert bulk mydb.mytable with nodescribe from select_into
```

其中，`mydb` 和 `mytable` 是受影响的数据库和表。

Bulk-Library 和非物化列的 bcp 处理

Bulk-Library 已经得到增强，可以在 Adaptive Server 15.7 中处理非物化列。

通过此项增强，您可以使用 Bulk-Library 和 **bcp** 15.7 及更高版本将数据批量复制到包含非物化列的已更改 Adaptive Server 表中。 如果使用较低版本的 **bcp** 将数据批量复制到非物化列中，Adaptive Server 将发出错误。

支持保留尾随零

Open Client 和 Open Server 15.7 版支持 Adaptive Server 15.7 中引入的 **disable varbinary truncation** 配置参数。 该参数指定 Adaptive Server 是保留还是截断 `varbinary` 和 `binary` 空值数据的尾随零。

低于 15.7 的 Adaptive Server 版本以及低于 15.7 的 **bcp** 和 **bulklib** 版本会截断 `varbinary` 数据类型的尾随零。 Adaptive Server 15.7 或更高版本以及 **bcp** 和 **bulklib** 15.7 或更高版本可截断或保留 `varbinary` 数据类型的尾随零。

缺省情况下，服务器的 **disable varbinary truncation** 为 0（禁用）。 将其设置为 1（启用）可启用该功能。

新增 DB-Library 溢出错误

发生的错误与 DB-Library 溢出有关。

使用导致整数溢出的 DB-Library 例程会导致以下错误：

```
302 = SYBEINTOVFL, "DB-LIBRARY internal error: The arithmetic
operation results in integer overflow."
```

导致溢出的 dbcursoropen DB-Library 例程的 scrollopt 和 nrows 参数相乘会导致以下错误：

```
301 = SYBCOPNOV, "dbcursoropen(): The multiplication of scrollopt and
nrows results in overflow."
```

新增对无名应用程序配置设置的处理

现在，您可以设置是否针对无名应用程序（应用程序未显式设置 CS_APPNAME）使用 `ocs.cfg` 运行时配置文件解析特定的应用程序设置，以及是否将找到的所有设置应用于该应用程序。

从操作系统中获取的可执行程序名会针对应用程序被设置为 CS_APPNAME，并用于解析运行时配置文件。

在 `ocs.cfg` 运行时配置文件的 DEFAULT 部分将 CS_USE_DISCOVERED_APPNAME 设置为 CS_TRUE 以启用该功能。

当将 CS_USE_DISCOVERED_APPNAME 设置为 CS_FALSE（缺省值）时，不会针对无名应用程序解析运行时配置文件。

使用 CS_SANITIZE_DISC_APPNAME 指定发现的无名应用程序（应用程序未显式设置 CS_APPNAME）的应用程序名称（从操作系统获取的可执行程序名）是按原样用于解析运行时配置文件，还是先转换为大写或小写后再用于解析运行时配置文件。

可在 `ocs.cfg` 运行时配置文件的 DEFAULT 部分中将 CS_SANITIZE_DISC_APPNAME 设置为以下任意值：

- CS_CNVRT_UPPERCASE - 在使用前将发现的应用程序名称转换为大写。
- CS_CNVRT_LOWERCASE - 在使用前将发现的应用程序名称转换为小写。
- CS_CNVRT_NOTHING（缺省值）- 按原样使用发现的应用程序名称。

TCP 套接字缓冲区大小配置

可以使用 Open Client 和 Open Server 上下文/连接和服务器属性来设置 TCP 输入和输出缓冲区的大小。

Open Client 和 Open Server 应用程序使用这些属性设置，通过操作系统 **setsockopt** 命令来设置缓冲区大小。 因为在 TCP 连接并接受命令之前必须调用 **setsockopt**，所以，您必须在尝试创建连接之前设置这些 Open Client 和 Open Server 属性。

属性

用于设置 TCP 输入和输出缓冲区大小的上下文/连接属性是 CS_TCP_RCVBUF 和 CS_TCP_SNDBUF。

表 8. 用于缓冲区大小配置的 Client-Library 属性

属性	含义	*buffer* 值	级别
CS_TCP_RCVBUF	客户端应用程序的输入缓冲区的大小	正整数	上下文与连接
CS_TCP_SNDBUF	客户端应用程序的输出缓冲区的大小	正整数	上下文与连接

上下文示例

```
ct_config(*context, CS_SET, CS_TCP_RCVBUF, &bufsize, CS_UNUSED,
NULL);
```

连接示例

```
ct_con_props(*connection, CS_SET, CS_TCP_RCVBUF, &bufsize,
CS_UNUSED, NULL);
```

用于设置 TCP 输入和输出缓冲区大小的服务器属性是 SRV_S_TCP_RCVBUF 和 SRV_S_TCP_SNDBUF。

表 9. 用于缓冲区大小配置的服务器属性

属性	SET/CLEAR	GET	当 *cmd* 为 CS_SET 时的 *bufp*	当 *cmd* 为 CS_GET 时的 *bufp*
SRV_S_TCP_RCVBUF	是	是	一个 CS_INT	一个 CS_INT
SRV_S_TCP_SNDBUF	是	是	一个 CS_INT	一个 CS_INT

服务器示例

```
srv_props(cp, CS_SET, SRV_S_TCP_SNDBUF, bufp, CS_SIZEOF(CS_INT),
(CS_INT *)NULL);
```

- 为您的应用程序设置这些参数（如果适当）。 例如，如果预计客户端将向服务器发送大量数据，请将 CS_TCP_SNDBUF 和 SRV_S_TCP_RCVBUF 设置为大值以增加相应的缓冲区大小。
- 缺省情况下，将套接字缓冲区大小设置为操作系统允许的最大大小。

用于所有 64 位产品的 isql64 和 bcp64

64 位版本的 **isql** 和 **bcp**（**isql64** 和 **bcp64**）现在适用于 Open Client 和 Open Server 支持的所有 UNIX 和 Windows 平台。

在低于 Open Server 和 SDK 15.5 ESD #9 的版本中，只有 64 位 **isql.exe** 和 **bcp.exe** 适用于 64 位 Windows。 如果您的脚本引用 **isql.exe** 或 **bcp.exe** 而且您打算使用 64 位版本，则必须将脚本中的引用更改为 **isql64.exe** 或 **bcp64.exe**。

对扩展可变长度行的支持

在 Adaptive Server 15.7 中，仅数据锁定 (DOL) 行的可变长度列的最大偏移已扩展到 32767 字节，这能让配置了大于 8K 的逻辑页大小的 Adaptive Server 支持宽可变长度 DOL 行。

用于填充 Adaptive Server 逻辑页的 Open Client 和 Open Server Bulk-Library 15.7 例程支持扩展 DOL 行。 此功能在 Bulk-Library 15.7 和更高版本中自动激活，但在 Adaptive Server 中必须启用才能激活。

配置了宽 DOL 行的数据库可以接受从使用 Bulk-Library 15.5 或更低版本的应用程序中发送的 DOL 行。 但是，使用 Bulk-Library 15.7 的应用程序不得向 Adaptive Server 15.5 或更低版本或者需要旧版本格式的 DOL 行的数据库发送宽 DOL 行。 否则，会发生以下错误之一：

- `BCP 在目标表中创建行失败。 列 %1! 将从超过 8191 字节的偏移量处开始，该起始位置无法在表（行）格式中准确表示。`
- `BCP 在目标表中创建行失败。 列 %1! 起始位置的偏移量大于 %2! 字节，当前数据库配置不允许此起始位置。`

要纠正此错误，请执行以下操作：

- 将表的锁定方案从仅数据锁定更改为所有页锁定。
- 当连接到 Adaptive Server 15.7 或更高版本时，请在目标数据库中启用 **allow wide dol rows** 选项。 请参见 Adaptive Server Enterprise《性能和调优系列： 物理数据库调优》中的第 2 章“数据存储”。

行格式高速缓存

Open Client 15.7 支持高速缓存行格式信息，以便在每次调用动态 SQL 语句时客户端应用程序能够请求数据服务器不要发送行格式信息。 行格式高速缓存可减少数据服务器和客户端应用程序之间的网络通信量，从而提高系统性能。

缺省情况下，行格式高速缓存在 Open Client 15.7 中处于启用状态。 若要禁用它，可将 CS_CMD_SUPPRESS_FMT 响应功能设置为 CS_FALSE。 使用 **ct_cmd_props()** 检查并设置 CS_CMD_SUPPRESS_FMT 的值。

要确定服务器是否支持行格式抑制，请使用 **ct_capability()** 检查 CS_RES_SUPPRESS_FMT 的值。

注意： 只有客户端应用程序连接到支持行格式高速缓存的服务器时，此功能才可用。

支持在游标关闭时释放锁

Open Client 15.7、Open Server 15.7 以及 Embedded SQL C 和 COBOL 15.7 处理器支持 Adaptive Server 15.7 中引入的 **release_locks_on_close** 游标选项。

此功能用于在游标关闭后释放读取锁。 请参见 Adaptive Server Enterprise《参考手册：命令》。

Client-Library 的使用

ct_cursor 语法中的 *option* 参数已经得到扩展，目前包括 CS_CUR_RELLOCKS_ONCLOSE。

使用此选项可在游标关闭时指令 Adaptive Server 释放共享锁。 若要和只读游标或可滚动游标一起使用，请使用逐位 OR 运算符“|”（竖线）：

- CS_CUR_RELLOCKS_ONCLOSE
- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_CURSOR
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_SEMISENSITIVE
- CS_CUR_RELLOCKS_ONCLOSE | CS_NOSCROLL_INSENSITIVE

示例

- 声明一个在关闭时释放共享锁的游标：

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE);
```

- 声明一个在关闭时释放共享锁的不敏感可滚动游标：

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE);
```

有关演示此功能的 Open Client 程序示例，请参见 `csr_disp_scrollcurs3.c`。

Open Server 的使用

当客户端应用程序在已经指定 CS_CUR_RELLOCKS_ONCLOSE 选项的情况下声明游标时，Open Server 将 SRV_CURDESC 结构的 **curstatus**（游标状态）字段设置为 SRV_CUR_RELLOCKS_ONCLOSE。

有关图示说明，请参见 `ctos` 示例代码中的 `cursor.c`。

ESQL/C 和 ESQL/COBOL 的使用

ESQL/C 和 ESQL/COBOL 中的 **SQL DECLARE** 语句已经得到扩展，目前均包括 **RELEASE_LOCKS_ON_CLOSE** 关键字。

```
EXEC SQL DECLARE cursor_name
    [SEMI_SENSITIVE | INSENSITIVE]
    [SCROLL | NOSCROLL]
    [RELEASE_LOCKS_ON_CLOSE]
    CURSOR FOR “select stmt”
    [for {read only | update [ of column_name_list]}]
```

不能将 **RELEASE_LOCKS_ON_CLOSE** 和 **UPDATE** 子句一起使用，但以下形式除外：

```
EXEC SQL declare cursor c1 release_locks_on_close
   cursor for select * from T for update of col_a
```

在此情况下，将忽略 **RELEASE_LOCKS_ON_CLOSE**。

cpre 和 **cobpre** 无法生成以下 **ct_cursor()** 选项：

- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE

`example8.cp` 中提供了 ESQL/C 示例代码；`example7.pco` 中提供了 ESQL/COBOL 示例代码。

作为存储过程参数的大对象

Open Client 和 Open Server 15.7 支持将 `text`、`unitext` 和 `image` 作为存储过程的输入参数以及动态 SQL 语句的参数。

为了便于针对此功能的使用开展登录协商，已经新增了两种连接功能：

- CS_RPCPARAM_LOB - 客户端应用程序向服务器发送此请求功能，从而确定大对象 (LOB) 数据类型能否用作存储过程的输入参数。 如果服务器无法支持此功能，则

会在初次登录协商中清除此功能位。如果您尝试向这种服务器发送 LOB 参数，则会发生错误。

- CS_RPCPARAM_NOLOB - 客户端应用程序发送此响应功能，以请求服务器拒绝将 LOB 数据作为参数发送。 缺省情况下，此功能处于打开状态。

将少量 LOB 数据作为参数发送

将少量 LOB 数据作为存储过程的输入参数或预准备 SQL 语句的参数发送与发送非 LOB 参数相同。

要发送少量 LOB 数据，请使用 **ct_param()** 或 **ct_setparam()** 为命令和数据分配内存并将它们直接发送到服务器。

当使用 `text`、`unitext` 或 `image` 参数时，必须设置 CS_DATAFMT 结构的 *maxlength* 字段。 *maxlength* 值指示是一次将所有 LOB 数据发送到服务器还是通过数据流将其传输到服务器。 如果 *maxlength* 大于零，将会在一个块中发送所有 LOB 数据。 如果 *maxlength* 设置为 CS_UNUSED，则通过数据流发送 LOB 数据，即调用一组 **ct_send_data()** 以成块发送数据。 块大小为零表示数据流结束。

示例 1
将少量 LOB 数据作为输入参数发送到存储过程：

```
CS_TEXT textvar[50];
CS_DATAFMT paramfmt;
CS_INT datalen;
CS_SMALLINT ind;

...
ct_command(cmd, CS_RPC_CMD, ...)

/*
** Clear and setup the CS_DATAFMT structure, then pass
** each of the parameters for the RPC.
*/
memset(&paramfmt, 0, sizeof(paramfmt));

/*
** First parameter, an integer.
*/
strcpy(paramfmt.name, "@intparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_INT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
ct_param(cmd, &paramfmt, (CS_VOID *)&intvar,
   sizeof(CS_INT), ind))

/*
** Second parameter, a (small) text parameter.
*/
```

```
strcpy((CS_CHAR *)textvar, “The Open Client and Open
   Server products both include Bulk-Library and
   CS-Library. ” );
datalen = sizeof(textvar);
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_TEXT_TYPE;
paramfmt.maxlength = EX_MYMAXTEXTLEN;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
ct_setparam(cmd, &paramfmt, (CS_VOID *)&textvar,
   &datalen, &ind);

ct_send(cmd);
ct_results(cmd, &res_type);

...
```

示例 2

使用预准备语句发送少量 LOB 数据:

```
/*
** Prepare the sql statement.
*/
sprintf(statement, "select title_id from mybooks where
   title like (?) ");

/*
** Send the prepared statement to the server
*/
ct_dynamic(cmd, CS_PREPARE, "my_dyn_stmt", CS_NULLTERM,
   statement, CS_NULLTERM);

ct_send(cmd);
handle_results(cmd);

/*
** Prompt user to provide a value for title
*/
printf("Enter title id value - enter an X if you wish
   to stop: \n");

while (toupper(title[0]) != 'X')
{
   printf("Retrieve detail record for title: ?");
   fgets(mytexttitle, 50,  stdin);

   /*
   ** Execute the dynamic statement.
   */

   ct_dynamic(cmd, CS_EXECUTE, "my_dyn_stmt",
   CS_NULLTERM, NULL, CS_UNUSED);

    /*
```

```
   ** Define the input parameter
   */

   memset(&data_format, 0, sizeof(data_format));
   data_format.status = CS_INPUTVALUE;
   data_format.namelen = CS_NULLTERM ;
   data_format.datatype = CS_TEXT_TYPE;
   data_format.format = CS_FMT_NULLTERM;
   data_format.maxlength = EX_MYMAXTEXTLEN;
   ct_setparam(cmd, &data_format,
      (CS_VOID *)mytexttitle, &datalen, &ind);

ct_send(cmd);
handle_results(cmd);
...
}
```

将大量 LOB 数据作为参数发送

大量 LOB 数据通过数据流发送到服务器，以便更好地管理资源。 在循环中使用 **ct_send_data()** 可将数据成块发送到服务器。

若要成块发送 LOB 数据参数，请使用以下设置定义该参数：

- 将 CS_DATAFMT 结构的 *datatype* 字段设置为 CS_TEXT_TYPE、CS_UNITEXT_TYPE 或 CS_IMAGE_TYPE。
- 将 CS_DATAFMT 结构的 *maxlength* 字段设置为 CS_UNUSED。
- 将 **ct_param()** 函数的 **data* 指针参数设置为 NULL。
- 将 **ct_param()** 函数的 `datalen` 参数设置为 0。

示例 1
成块发送大 LOB 数据参数：

```
#define BUFSIZE 2048

int fp;
char sendbuf[BUFSIZE]

/*
** Clear and setup the CS_DATAFMT structure, then pass
** each of the parameters for the RPC.
*/
memset(&paramfmt, 0, sizeof(paramfmt));
strcpy(paramfmt.name, "@intparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_INT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

ct_param(cmd, &paramfmt, (CS_VOID *)&intvar,
   sizeof(CS_INT), 0))

/*
```

```
** Text parameter, sent as a BLOB.
*/
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_TEXT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

/*
** Although the actual data will not be sent here, we
** must invoke ct_setparam() for this parameter to send
** the parameter format (paramfmt) information to the
** server, prior to sending all parameter data.
** Set *data to NULL and datalen = 0, to indicate that
** the length of text data is unknown and we want to
** send it in chunks to the server with ct_send_data().
*/
ct_setparam(cmd, &paramfmt, NULL, 0, 0);

/*
** Another LOB parameter (image), sent in chunks with
** ct_send_data()
*/
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_IMAGE_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

/*
** Just like the previous parameter, invoke
** ct_setparam() for this parameter to send the
** parameter format.
*/
ct_setparam(cmd, &paramfmt, NULL, 0, 0);

/*
** Repeat this sequence of filling paramfmt and calling
** ct_param() for any subsequent parameter that needs
** to be sent before finally sending the data chunks for
** the LOB type parameters.
*/
strcpy(paramfmt.name, "@any_otherparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_MONEY_TYPE;
...

/*
** Send the first LOB (text) parameter in chunks of
**  'BUFSIZE'  to the server. We must end with a 0 bytes
** write to indicate the end of the current parameter.
*/
fp = open( "huge_text_file" , O_RDWR, 0666);
```

```
do
{
   num_read = read(fp, sendbuf, BUFSIZE);
   ct_send_data(cmd, (CS_VOID *)sendbuf, num_read);
}  while (num_read != 0);

/*
** Repeat the ct_send_data() loop for the next LOB
** parameter.
** Send the image parameter in chunks of ‘BUFSIZE’
** to the server as well and end with a 0 bytes write
** to indicate the end of the current parameter.
*/
fp = open(“large_image_file”, O_RDWR, 0666);
do
{
   num_read = read(fp, sendbuf, BUFSIZE);
   ct_send_data(cmd, (CS_VOID *)sendbuf, num_read);
} while (num_read != 0);

/*
** Ensure that all the data is flushed to the server
*/
ct_send(cmd);
```

示例 2
使用预准备 SQL 语句以数据流形式发送 LOB 数据：

```
/*
** Prepare the sql statement.
*/
sprintf(statement, "select title_id from mybooks
   where title like (?) ");

/*
** Send the prepared statement to the server
*/
ct_dynamic(cmd, CS_PREPARE, "mydyn_stmt", CS_NULLTERM,
   statement, CS_NULLTERM);

ct_send(cmd);
handle_results();

/*
** Promt user to provide a value for title
*/
printf("Enter title id value - enter an X if you wish
   to stop: \n");

while (toupper(myblobtitle[0]) != 'X')
{
   printf("Retrieve detail record for title: ?");
   fgets(myblobtitle, 50,  stdin);
```

```
   /*
   ** Execute the dynamic statement.
   */
   ct_dynamic(cmd, CS_PREPARE, "my_dyn_stmt",
   CS_NULLTERM, statement, CS_NULLTERM);

   /*
   ** Define the input parameter, a TEXT type that we
   want to send in chunks to the server.
   */
   memset(&data_format, 0, sizeof(data_format)) ;
   data_format.namelen = CS_NULLTERM ;
   data_format.datatype = CS_TEXT_TYPE;
   data_format.maxlength = CS_UNUSED;
   data_format.status = CS_INPUTVALUE;
   ct_setparam(cmd, &data_format, NULL, 0, 0);

   /*
   ** Send the 'myblobtitle' data in chunks of
   ** 'CHUNKSIZE' to the server with ct_send_data() and
   ** end with 0 bytes to indicate the end of data for
   ** this parameter. This is just an example to show
   ** how chunks can be sent. (myblobtitle[] is used as
   ** a simple example. This could also be replaced by
   ** large file which would be read in chunks from disk
   ** for example).
   */
   bytesleft = strlen(myblobtitle);
   bufp = myblobtitle;

   do
   {
      sendbytes = min(bytesleft, CHUNKSIZE);
      ct_send_data(cmd, (CS_VOID *)bufp, sendbytes);
      bufp += bufp + sendbytes;
      bytesleft -= sendbytes;
   }  while (bytesleft > 0)

   /*
   ** End with 0 bytes to indicate the end of current
   data.
   */
   ct_send_data(cmd, (CS_VOID *)bufp, 0);

   /*
   ** Insure that all the data is sent to the server.
   */
   ct_send(cmd);
   handle_results(cmd)
   ...
}

/*
** Deallocate the prepared statement and finish up.
*/
```

```
ct_dynamic(cmd, CS_DEALLOC, "my_dyn_stmt", CS_NULLTERM,
   NULL, CS_UNUSED);

ct_send(cmd);
handle_results(cmd);
```

在 Open Server 中检索 LOB 参数

使用 **srv_xferdata** 一次检索全部 LOB 参数数据，也可以使用新的 **srv_get_data** 例程成块检索 LOB 参数数据。

当参数长度设置为 CS_UNUSED 时，Open Server 成块检索 LOB 参数。 请参见 srv_get_data。

示例
检索 LOB 参数的说明：

```
/*
** Retrieve the description of the parameters coming
** from client
*/

for (paramnum = 1; paramnum <= numparams; paramnum++)
{
   /*
   ** Get a description of the parameter.
   */
   ret = srv_descfmt(spp, CS_GET, SRV_RPCDATA,
      paramnum, &(paramfmtp[paramnum - 1]));

   /*
   ** Allocate space for the parameters and bind the
   ** data.
   */
   if (paramfmtp[paramnum-1].maxlength >= 0)
   {
      if (paramfmtp[paramnum-1].maxlength > 0)
      {
         data[paramnum-1] = calloc(1,
            paramfmtp[paramnum-1].maxlength);
      }
      else
      {
         ind[paramnum-1] = CS_NULLDATA;
      }
   }
   else
   {
      /*
      ** Allocate a large size buffer for BLOB data
      ** (which length is unknown yet)
      */
      blobbuf[blobnum] = malloc(BUFSIZE);
      blobnum++;
```

```
   }

   srv_bind(spp, CS_GET, SRV_RPCDATA, paramnum,
      &(paramfmtp[paramnum-1]), data[paramnum-1],
      &(len[paramnum-1]), &(ind[paramnum-1]))

   /*
   ** For every LOB parameter, call srv_get_data() in
   ** a loop as long as it succeeds
   /*
   for (i = 0; i < blobnum ; i++)
   {
      bufp = blobbuf[i];
      bloblen[i] = 0;
      do
      {
         ret = srv_get_data(spp, bufp, BUFSIZE,
            &outlen);
         bufp += outlen;
         bloblen[i] += outlen;
      } while (ret == CS_SUCCEED);

      /*
      ** Check for the correct return code
      */
      if (ret != CS_END_DATA)
      {
         return CS_FAIL;
      }

   }

   /*
   ** And receive remaining client data srv_xferdata()
   */
   ret = srv_xferdata(spp, CS_GET, SRV_RPCDATA);
}
```

srv_get_data

成块从客户端读取 text、unitext 或 image 参数流。

语法

```
CS_RETCODE srv_get_data(spp, bp, buflen, outlenp)

SRV_PROC *spp;
CS_BYTE *bp;
CS_INT buflen;
CS_INT *outlenp;
```

参数

- *spp* – 指向内部线程控制结构的指针。

- *bp* – 指向存储客户端数据的缓冲区的指针。
- *buflen* – **bp* 指针的大小。 它表示每块中传输的字节数。
- *outlenp* – 输出参数 *outlenp* 中包含读入 **bp* 缓冲区的字节数。

返回值

- CS_SUCCEED – **srv_get_data()** 已成功运行，更多数据待处理。
- CS_FAIL – 例程失败。
- CS_END_DATA – **srv_get_data()** 已读取完整个 `text`、`unitext` 或 `image` 参数。

SDK 15.7 中针对 jConnect、Adaptive Server Enterprise 驱动程序和提供程序的功能

本节介绍 SDK 15.7 中针对 jConnect、Adaptive Server Enterprise ODBC 驱动程序、Adaptive Server Enterprise OLE DB 提供程序和 Adaptive Server Enterprise ADO.NET 数据提供程序引入的新增功能。

ODBC 驱动程序版本信息实用程序

odbcversion 实用程序显示 ODBC 驱动程序的信息。

语法

odbcversion`-version|-fullversion|-connect`*dsnuseridpassword*

参数

`-version`

显示 ODBC 驱动程序的简单数字版本字符串。

`-fullversion`

显示 ODBC 驱动程序的详细版本字符串。

`-connect` *dsnuseridpassword*

显示 Adaptive Server 版本以及该 Adaptive Server 上安装的 ODBC 和 OLEDB MDA 脚本的版本。 该参数需要三个变量： 分别是 Adaptive Server 的数据源名称 *dsn*、用户 ID 和口令（用于连接到 Adaptive Server）。

示例

获取用于连接到 Adaptive Server 的 ODBC 驱动程序的简单数字版本字符串：

```
odbcversion -version
```

该实用程序返回数字版本字符串：

```
15.05.00.1015
```

用法

如果不指定参数，**odbcversion** 实用程序会显示有效参数列表。

SupressRowFormat2 连接字符串属性

使用 Adaptive Server Enterprise ODBC 驱动程序 15.7、Adaptive Server Enterprise OLE DB 提供程序 15.7 和 Adaptive Server Enterprise ADO.NET 数据提供程序 15.7，您可以通过 SupressRowFormat2 连接字符串属性强制 Adaptive Server 在可能的情况下使用 TDS_ROWFMT 字节序列发送数据，而不使用 TDS_ROWFMT2 字节序列。

TDS_ROWFMT 中包含的数据比 TDS_ROWFMT2 少（包括目录、方案、表和列信息），可以使很多小的 **select** 操作提高性能。 因为在 SupressRowFormat2 设置为 1 时，服务器发送更少的结果集元数据，所以，某些信息不可用于客户端程序。 如果您的应用程序依赖于缺失的元数据，则不应启用此属性。

值:

- 0 – 缺省值；不禁止 TDS_ROWFMT2。
- 1 – 强制服务器尽可能以 TDS_ROWFMT 格式发送数据。

示例
此连接字符串强制服务器在通过 ADO.NET 数据提供程序建立的连接上尽可能以 TDS_ROWFMT 格式发送数据。

```
Data Source='myASE';Port=5000;Database=myDB;
Uid=myUID;Pwd=myPWD;SupressRowFormat2=1
```

对 UseCursor 属性进行的增强

可以使用 Adaptive Server Enterprise ODBC 驱动程序的 UseCursor 连接字符串属性来确定服务器端游标如何用于生成结果集的 SQL 语句。

已对此属性进行更新，以便客户端应用程序能够控制哪些语句创建服务器端游标（值 2）。

值:

- 0 - 缺省值。 不使用服务器端游标。
- 1 – 服务器端游标用于所有生成结果集的语句。
- 2 – 只有在调用 **SQLSetCursorName** ODBC 函数时，服务器端游标才用于生成结果集的语句。 因为游标使用更多资源，所以此设置能让您将服务器端游标的使用限制为可从这些游标中获益的语句。

在不使用 ODBC 驱动程序管理器跟踪的情况下记录

在 Adaptive Server Enterprise ODBC 驱动程序 15.7 中，可以在不使用 ODBC 驱动程序管理器跟踪的情况下记录对 ODBC API 的调用。 当不使用驱动程序管理器或在不支持跟踪的平台上运行它时，这会很有用。

若要在 Microsoft Windows 上启用此功能，请使用 LOGCONFIGFILE 环境变量或 Microsoft Windows 注册表。 若要在 Linux 上启用此功能，请使用 LOGCONFIGFILE。

当使用 LOGCONFIGFILE 时，请将该环境变量设置为 ODBC 日志配置文件的完整路径。 LOGCONFIGFILE 会覆盖任何现有的注册表条目。

当使用 Microsoft Windows 注册表时，请在 `HKEY_CURRENT_USER\Software\Sybase\ODBC` 或 `HKEY_LOCAL_MACHINE\Software\Sybase\ODBC` 中创建一个名为 LogConfigFile 的条目，并将其值设置为 ODBC 日志配置文件的完整路径。 例如：

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Sybase\ODBC]
"LogConfigFile"="c:\\temp\\odbclog.properties"
```

若要禁用日志记录，请删除或重命名 *LogConfigFile* 值。

注意： 在 `HKEY_CURRENT_USER` 中指定的值会覆盖所有在 `HKEY_LOCAL_MACHINE` 中设置的值。

日志配置文件

此配置文件用于控制 ODBC 日志文件的格式和位置。

在以下示例中，粗体行指定日志文件的保存位置：

```
log4cplus.rootLogger=OFF, NULL

log4cplus.logger.com.sybase.dataaccess.odbc.api=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.parameter=TRACE,
ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.parameter=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.returncode=TRACE,
ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.returncode=fals
e

log4cplus.appender.NULL=log4cplus::NullAppender

log4cplus.appender.ODBCTRACE=log4cplus::FileAppender
log4cplus.appender.ODBCTRACE.File=c:\temp\odbc.log
log4cplus.appender.ODBCTRACE.layout=log4cplus::PatternLayout
log4cplus.appender.ODBCTRACE.ImmediateFlush=true
log4cplus.appender.ODBCTRACE.layout.ConversionPattern=%d{%H:%M:%S.
%q} %t %p
   %-25.25c{2} %m%n
```

jConnect setMaxRows 增强

JDBC 程序使用 **Statement.setMaxRows(int max)** 限制结果集返回的行数。 在 jConnect 7.0 和更低版本中，**select**、**insert**、**update** 和 **delete** 语句的结果会被此限制值限定行数。

为与 JDBC 规范一致，jConnect 7.07 引入了 SETMAXROWS_AFFECTS_SELECT_ONLY 连接属性，当设置为 true（缺省值）时，仅限制 **select** 语句返回的行。

当连接到 Adaptive Server 15.5 或更低版本时，会忽略 SETMAXROWS_AFFECTS_SELECT_ONLY。

TDS ProtocolCapture

Adaptive Server Enterprise ODBC 驱动程序 15.7 引入了 ProtocolCapture 连接字符串属性，该字符串属性可指定用于接收在 ODBC 应用程序和 Adaptive Server 之间交换的 Tabular Data Stream™ (TDS) 包的文件。

ProtocolCapture 会立即生效，这样在建立连接过程中交换的 TDS 包就会写入使用文件前缀生成的唯一文件名中。 TDS 包在连接持续过程中一直写入该文件。 可以使用 Ribo 和其它协议转换工具来解释 TDS 捕获文件。

例如，若要将 `tds_capture` 指定为 TDS 跟踪日志文件前缀，请键入：

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;ProtocolCapture=tds_capture;
```

第一个连接生成 `tds_capture0.tds`，第二个连接生成 `tds_capture1.tds`，依此类推。

没有绑定参数数组的 ODBC 数据批处理

当对不同参数值执行同样的 SQL 语句时，客户端应用程序通常使用 **SQLExecute**、**SQLExecuteDirect** 和 **SQLBulkOperations** 绑定参数数组并执行每组参数。

在将数组绑定到 SQL 参数时，会为数组分配内存，而且在执行 SQL 语句之前所有数据都复制到该数组。 在处理大量事务时，这可能会导致内存和资源的使用效率降低。

在 Adaptive Server Enterprise ODBC 驱动程序 15.7 中，客户端应用程序可以使用 **SQLExecute** 向 Adaptive Server 成批发送参数，而不将参数作为数组绑定。 **SQLExecute** 会一直返回 SQL_BATCH_EXECUTING，直到最后一批参数发送并处理完毕。 在最后一批参数处理完毕后会返回执行状态。

只有最后一个 **SQLExecute** 语句完成后，对 **SQLRowCount** 的调用才有效。

管理数据批

SQL_ATTR_BATCH_PARAMS 是一个特定于 Adaptive Server 的连接属性，引入它的目的是管理发送到 Adaptive Server 的参数批。 使用 **SQLSetConnectAttr** 设置 SQL_ATTR_BATCH_PARAMS。

值：

- SQL_BATCH_ENABLED – 通知 Adaptive Server Enterprise ODBC 驱动程序对参数进行批处理。 当处于此状态时，如果在连接上执行一个由 **SQLExecute** 处理的语句（在将 SQL_ATTR_BATCH_PARAMS 设置为 SQL_BATCH_ENABLED 之后执行的第一个语句）以外的语句，驱动程序便会发送错误。
- SQL_BATCH_LAST_DATA – 指定下一批参数是最后一批，而且参数中包含数据。
- SQL_BATCH_LAST_NO_DATA – 指定下一批参数是最后一批，而且忽略这些参数。
- SQL_BATCH_CANCEL – 通知 Adaptive Server Enterprise ODBC 驱动程序取消该批并回退事务。
  只有未提交的事务才可以回退。
- SQL_BATCH_DISABLED – （缺省值）Adaptive Server Enterprise ODBC 驱动程序在处理最后一批参数后恢复为此状态。 不能手动将 SQL_ATTR_BATCH_PARAMS 设置为此值。

管理数据批的示例

下面提供了两个有关管理数据批的示例。

示例 1
向服务器发送一批参数而不绑定参数数组：

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to start
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
   (SQLPOINTER)SQL_BATCH_ENABLED, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Bind the parameters. This can be done once for the entire batch
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
   SQL_C_LONG, SQL_INTEGER, l1, 0, &c1, l1, &l1);
sr = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_LONGVARCHAR, l2, 0, buffer, l2, &l2);
}

// Run a batch of 10 for (int i = 0; i < 10; i++)
{
   c1 = i;
   memset(buffer, 'a'+i, l2);
   sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
   printError(sr, SQL_HANDLE_STMT, stmt);
}
```

示例 2
结束并关闭一批参数：

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to end
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
  (SQLPOINTER)SQL_BATCH_LAST_NO_DATA, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Call SQLExecDirect one more time to close the batch
// - Due to SQL_BATCH_LAST_NO_DATA, this will not
// process the parameters
sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
```

ODBC 数据批处理注意事项

ODBC 数据批处理功能的特定注意事项。

- 此功能仅支持不返回结果集或具有输出参数的语句和存储过程。
- 不支持异步模式。 当处于批处理模式时，应用程序无法在同一连接上执行批处理以外的任何语句。
- 不支持 SQL_DATA_AT_EXEC。 将 LOB 参数作为一般参数绑定。
- 在不绑定参数数组且设置了 SQL_ATTR_PARAM_STATUS_PTR 的情况下对数据进行批处理时，Adaptive Server Enterprise ODBC 驱动程序将从 SQLSetStmtAttr 的 StringLength 参数中检索数组元素数，而不是从 SQL_ATTR_PARAMSET_SIZE 中检索。

jConnect 中的优化批处理

jConnect for JDBC 7.07 实施内部算法来加速 **PreparedStatement** 对象的批处理操作。

此算法在 HOMOGENEOUS_BATCH 连接属性设置为 true 时被调用。

注意： 只有客户端应用程序连接到支持同类批处理的服务器时，此功能才可用。Adaptive Server Enterprise 15.7 引入了对同类批处理的支持。

以下示例说明使用 **addBatch** 和 **executeBatch** 方法的 **PreparedStatement** 批处理操作：

```
String sql = "update members set lastname = ? where member_id = ?";
prep_stmt = connection.prepareStatement(sql);
prep_stmt.setString(1, "Forrester");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();
prep_stmt.setString(1, "Robinson");
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();
prep_stmt.setString(1, "Servo");
prep_stmt.setLong(2, 45131);
prep_stmt.addBatch();
prep_stmt.executeBatch();
```

其中，connection 表示连接实例，prep_stmt 表示预准备语句实例，而 ? 表示预准备语句的参数占位符。

含有 LOB 列的同类批处理

如果 HOMOGENEOUS_BATCH 和 ENABLE_LOB_LOCATORS 属性设置为 true，客户端应用程序则无法将 LOB 和非 LOB 预准备语句 setter 方法混合在同一批中。

例如，以下语句是无效的：

```
String sql = "update members SET catchphrase = ? WHERE member_id
= ?";
prep_stmt = connection.prepareStatement(sql);
prep_stmt.setString(1, "Push the button, Frank!");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();
Clob myclob = con.createClob();
myclob.setString(1, "Hi-keeba!");
prep_stmt.setClob(1, myclob);
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();
pstmt.executeBatch();
```

其中，catchphrase 是类型为 text 的列。 此代码失败的原因是 **setString** 方法和 **setClob** 方法用在同一列的同一批处理中。

没有行累计的 jConnect 参数批处理

jConnect for JDBC 7.07 添加了 SEND_BATCH_IMMEDIATE 连接属性。

当设置为 true 时，jConnect 在调用 **PreparedStatement.addBatch()** 后立即发送当前行的参数。 这最大限度地减少了客户端内存的使用，并让服务器有更多时间来处理批参数。

SEND_BATCH_IMMEDIATE 的缺省值为 false，当设置为缺省值时，它会通知 jConnect 仅在调用 **PreparedStatement.executeBatch()** 后才发送批参数（和以前一样）。

遇到错误仍执行的 jConnect 批更新增强

jConnect for JDBC 7.07 引入了 EXECUTE_BATCH_PAST_ERRORS 连接属性，当设置为 true 时，能让批更新操作忽略在执行各个语句时遇到的非致命错误并完成批更新。

当设置为 false（缺省值）时，如果遇到错误，批更新便会中止，和旧版本中一样。

有关批更新用法和其结果解释的信息，请参见《jConnect for JDBC 程序员参考》。

支持在游标关闭时释放锁

Adaptive Server 15.7 扩展了 **declare cursor** 语法，使其包括 **release_locks_on_close** 选项，用以在游标关闭时在隔离级别 2 和 3 释放共享游标锁。 Adaptive Server Enterprise ODBC 驱动程序 15.7 和 jConnect for JDBC 7.07 支持 release-lock-on-close 的语义。

若要将此功能应用到所有在 Adaptive Server Enterprise ODBC 驱动程序连接上创建的只读游标，请将 ReleaseLocksOnCursorClose 连接属性设置为 1。 ReleaseLocksOnCursorClose 的缺省值为 0。

若要在 jConnect for JDBC 连接上应用，请将 RELEASE_LOCKS_ON_CURSOR_CLOSE 连接属性设置为 true。 RELEASE_LOCKS_ON_CURSOR_CLOSE 的缺省值为 false。

通过这些连接属性应用的设置是静态的，一旦连接建立后便无法更改。 此设置只有在连接到支持 **release_locks_on_close** 的服务器时才会生效。

有关 **release_locks_on_close** 的信息，请参见《Adaptive Server Enterprise 参考手册：命令》。

select for update 支持

Adaptive Server 15.7 支持 **select for update**，它可以为同一事务内的后续更新锁定行，并支持可更新游标的排它锁。

请参见《Adaptive Server Enterprise Transact-SQL 用户指南》的第 2 章“查询： 从表中选择数据”。

当 **for update** 子句添加到 **select** 语句以及客户端内打开的任何可更新游标中后，此功能便可自动用于客户端。 有关创建可更新游标的信息，请参见《Adaptive Server Enterprise ODBC 驱动程序用户指南》和《jConnect for JDBC 程序员参考》。

对扩展可变长度行的支持

如果可变长度列从行起始位置后超过 8191 字节处开始，则配置了 16K 逻辑页大小的低于 15.7 版的 Adaptive Server 无法创建含有可变长度行的仅数据锁定 (DOL) 表。 从 Adaptive Server 15.7 开始删除了这一限制。

请参见 Adaptive Server Enterprise《性能和调优系列： 物理数据库调优》中的第 2 章“数据存储”。

ODBC 和 JDBC 客户端不需要特殊配置即可使用此功能。 当连接到配置为接收宽 DOL 行的 Adaptive Server 15.7 版时，这些客户端会自动使用宽偏移插入记录。 如果客户

端尝试向 Adaptive Server 的早期版本或向已禁用宽 DOL 行选项的 15.7 版 Adaptive Server 发送宽 DOL 行，则会收到错误消息。

对非实现列的支持

已经对 Adaptive Server Enterprise ODBC 驱动程序 15.7 中的批量插入例程进行了增强，用以在 Adaptive Server 15.7 中处理非实现列。

Adaptive Server Enterprise ODBC 驱动程序的早期版本在表定义中包含非实现列时无法将数据批量插入 Adaptive Server。 如果 Adaptive Server Enterprise ODBC 驱动程序的早期版本尝试对非实现列进行批量插入，Adaptive Server 会引发错误。

行内和行外 LOB 存储支持

在 Adaptive Server 15.7 中，当有足够的内存存储整个行时，标为行内存储的 LOB 列将存储在行内。

如果由于对行中的列进行更新而使行大小增大并超过定义的限制值，在行内存储的 LOB 列会移动到行外以使其仍不超过限制值。 请参见《Adaptive Server Enterprise Transact-SQL 用户指南》中的第 21 章“行内、行外 LOB”。

Adaptive Server Enterprise ODBC 驱动程序 15.7 和 jConnect for JDBC 7.07 中的批量插入例程支持 Adaptive Server 中的 `text`、`image` 和 `unitext` LOB 列的行内和行外存储。 早期客户端版本中的批量插入例程始终将 LOB 列存储在行外。

作为存储过程参数的大对象

在 Adaptive Server 15.7 中引入了将 LOB 数据作为存储过程输入参数传送的做法。

jConnect for JDBC 7.07、Adaptive Server Enterprise ODBC 驱动程序 15.7、Adaptive Server Enterprise OLE DB 提供程序 15.7 和 Adaptive Server Enterprise ADO.NET 数据提供程序 15.7 支持将 `text`、`unitext` 和 `image` 用作存储过程中的输入参数以及参数标记数据类型。

大对象定位符支持

jConnect for JDBC 7.07 和 Adaptive Server Enterpris ODBC 驱动程序 15.7 支持大对象 (LOB) 定位符。

LOB 定位符中包含指向 LOB 数据的逻辑指针，而不是数据本身，减少了通过网络在 Adaptive Server 和其客户端之间传送的数据量。 Adaptive Server 15.7 中引入了对 LOB 定位符的服务器支持。

当连接到支持 LOB 定位符的 Adaptive Server 并关闭 **autocommit** 时，jConnect for JDBC 7.07 使用服务器端定位符访问 LOB 数据。否则，jConnect 会在客户端实现 LOB 数据。可以将整个 LOB API 用于客户端实现的 LOB 数据，但由于数据较大，API 性能可能会与用于 LOB 定位符时不同。

Adaptive Server Enterprise ODBC 驱动程序 15.7 客户端只有连接到支持它的 Adaptive Server 才能使用 LOB 定位符。

注意： 当您使用 LOB 定位符时，检索每个行上都包括 LOB 数据的大结果集可能会影响应用程序的性能。Adaptive Server 将 LOB 定位符作为结果集的一部分返回，并获取 LOB 数据，jConnect 和 ODBC 驱动程序必须高速缓存剩余的结果集。我们建议您控制结果集的大小，或者启用游标支持来限制要高速缓存的数据的大小。

jConnect for JDBC 支持

要启用 LOB 定位符支持，请在 ENABLE_LOB_LOCATORS 连接属性设置为 true 的情况下与 Adaptive Server 建立连接。

启用后，客户端应用程序便可以使用 **java.sql** 软件包中的 **Blob**、**Clob** 和 **NClob** 类访问定位符。

注意： 如果 LOB 定位符和 **autocommit** 都已启用，jConnect 会自动将 LOB 定位符切换为客户端实现的 LOB 定位符，即使连接的 Adaptive Server 能够支持它们时也是如此。这会增加客户端使用的内存，而且可能会降低性能。因此，建议您在设置 **autocommit off** 的条件下使用 LOB 定位符。

有关 **Blob**、**Clob** 和 **NClob** 类的详细信息，请参见 Java 文档。

Adaptive Server Enterprise ODBC 驱动程序支持

要启用 LOB 定位符支持，请在 ENABLE_LOB_LOCATORS 连接属性设置为 true 的情况下与 Adaptive Server 建立连接。

当 EnableLOBLocator 设置为缺省值 0 时，ODBC 驱动程序不能检索 LOB 列的定位符。当启用 LOB 定位符时，连接应设置为 **autocommit off**。

ODBC 源程序中还必须包括 `sybasesqltypes.h` 头文件。`sybasesqltypes.h` 文件位于 ODBC 安装目录下的 `include` 目录中。

用于定位符支持的 ODBC 数据类型映射

该表列出了 Adaptive Server 定位符数据类型与 ODBC 数据类型之间的映射。

ASE 数据类型	ODBC SQL 类型	ODBC C 类型
`text_locator`	SQL_TEXT_LOCATOR	SQL_C_TEXT_LOCATOR
`image_locator`	SQL_IMAGE_LOCATOR	SQL_C_ IMAGE_LOCATOR
`unitext_locator`	SQL_UNITEXT_LOCATOR	SQL_C_ UNITEXT_LOCATOR

支持的转换

该表列出了 Adaptive Server 定位符数据类型所支持的转换。

	SQL_C_TEXT_LOCATOR	SQL_C_IMAGE_LOCATOR	SQL_C_UNITEXT_LOCATOR
SQL_TEXT_LOCATOR	X		
SQL_IMAGE_LOCATOR		X	
SQL_UNITEXT_LOCATOR			X
SQL_LONGVARCHAR			
SQL_WLONGVARCHAR			
SQL_LONGVARBINARY			
图例：x = 支持的转换。			

支持 LOB 定位符的方法

ODBC API 方法支持 LOB 定位符。

- **SQLBindCol** – *TargetType* 可以是任意的 ODBC C 定位符数据类型。
- **SQLBindParameter** – *ValueType* 可以是任意的 ODBC C 定位符数据类型。*ParameterType* 可以是任意 ODBC SQL 定位符数据类型。
- **SQLGetData** – *TargetType* 可以是任意 ODBC C 定位符数据类型。
- **SQLColAttribute** – SQL_DESC_TYPE 和 SQL_DESC_CONCISE_TYPE 描述符可以返回任意 ODBC SQL 定位符数据类型。
- **SQLDescribeCol** – 数据类型指针可以是任意的 ODBC SQL 定位符数据类型。

请参见《Microsoft ODBC API 参考》(Microsoft ODBC API Reference)。

预取 LOB 数据的隐式转换

在 Adaptive Server Enterprise ODBC 驱动程序 15.7 中，当 Adaptive Server 返回 LOB 定位符时，您可以使用 **SQLGetData** 和 **SQLBindCol** 通过将列绑定到 SQL_C_CHAR 或 SQL_C_WCHAR（对于 `text` 定位符）或 SQL_C_BINARY（对于 `image` 定位符）来检索基础预取 LOB 数据。

设置 SQL_ATTR_LOBLOCATOR 属性可在连接中启用或禁用定位符。 如果在连接字符串中指定了 EnableLOBLocator，则会使用 EnableLOBLocator 的值初始化 SQL_ATTR_LOBLOCATOR，否则，它设置为 SQL_LOBLOCATOR_OFF（缺省值）。 若要启用定位符，请将该属性设置为 SQL_LOBLOCATOR_ON。 使用 **SQLSetConnectAttr** 可设置该属性的值，使用 **SQLGetConnectAttr** 可检索其值。

使用 **SQLSetStatementAttr** 可设置 SQL_ATTR_LOBLOCATOR_FETCHSIZE 以指定要检索的 LOB 数据的大小（对于二进制数据以字节为单位，对于字符数据以字符为单位）。 缺省值 0 表示不请求预取数据，值为 -1 将检索整个 LOB 数据。

注意： 如果要检索的列的基础LOB数据大小超过您设置的预取数据大小，则在ODBC客户端尝试直接检索该数据时会引发本机错误 3202。 发生这种情况时，客户端可以通过调用 **SQLGetData** 获取基础定位符并执行定位符的所有可用操作，来检索完整数据。

示例 1

当预取数据代表完整 LOB 值时，使用 **SQLGetData** 检索 `image` 定位符：

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_ON,
   0);

// Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt, SQL_ATTR_LOBLOCATOR_FETCHSIZE,
(SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);

printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);

sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

//Retrieve the binary data  (Complete Data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);
printError(sr, SQL_HANDLE_STMT, stmt);


//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);
```

```
SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

示例 2

当预取数据代表截断 LOB 值时，使用 **SQLGetData** 检索 `image` 定位符：

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
   (SQLPOINTER)SQL_LOCATOR_ON, 0);

//Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt,
   SQL_ATTR_LOBLOCATOR_FETCHSIZE, (SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);
printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

// Retrieve the binary data(Truncated data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);

if(sr == SQL_SUCCESS_WITH_INFO)
{
   SQLTCHAR errormsg[ERR_MSG_LEN];
   SQLTCHAR sqlstate[SQL_SQLSTATE_SIZE+1];
   SQLINTEGER nativeerror = 0;
   SQLSMALLINT errormsglen = 0;
```

```
   retcode = SQLGetDiagRec(handleType, handle, 1, sqlstate,
&nativeerror,
      errormsg, ERR_MSG_LEN, &errormsglen);

   printf("SqlState: %s Error Message: %s\n", sqlstate, errormsg);

   //Handle truncation of LOB data; if data was truncated call
SQLGetData to
   // retrieve the locator.

   /* Warning returns truncated LOB data */
   if (NativeError == 32028) //Error code may change
   {
      BYTE ImageLocator[SQL_LOCATOR_SIZE];
      sr = SQLGetData(stmt, 1, SQL_C_IMAGE_LOCATOR, &ImageLocator,
         sizeof(ImageLocator), &Len);
      printError(sr, SQL_HANDLE_STMT, stmt);

      /*
         Perform locator specific calls using image Locator on a
separate
         statement handle if needed
      */
   }
}

//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

使用定位符访问和处理 LOB

ODBC API 不直接支持 LOB 定位符。 ODBC 客户端应用程序必须使用 Transact-SQL 函数，才能对定位符进行运算和处理 LOB 值。 Adaptive Server Enterprise ODBC 驱动程序引入了若干存储过程来达成必需 Transact-SQL 函数的使用。

可对 LOB 定位符执行各种操作。 这些参数的输入和输出值可以是 Adaptive Server 可隐式转换为存储过程定义的任意类型。

有关此处所列的 Transact-SQL 命令和函数的详细信息，请参见《Adaptive Server Enterprise 参考手册：构件块》 中的“Transact-SQL 函数”。

初始化 Text 定位符

使用 **sp_drv_create_text_locator** 创建 `text_locator` 并选择某一值对其进行初始化。 **sp_drv_create_text_locator** 访问 Transact-SQL 函数 **create_locator**。

语法

```
sp_drv_create_text_locator [init_value]
```

输入参数
init_value - `varchar` 或 `text` 值，用于初始化新定位符。

输出参数
无。

结果集
类型为 `text_locator` 的列。 定位符引用的 LOB 附带了 *init_value*。

初始化 Unitext 定位符

使用 **sp_drv_create_unitext_locator** 创建 `unitext_locator` 并选择某一值对其进行初始化。 **sp_drv_create_unitext_locator** 访问 Transact-SQL 函数 **create_locator**。

语法

```
sp_drv_create_unitext_locator [init_value]
```

输入参数
init_value - `univarchar` 或 `unitext` 值，用于初始化新定位符。

输出参数
无。

结果集
类型为 `unitext_locator` 的列。 定位符引用的 LOB 附带了 *init_value*。

初始化 Image 定位符

使用 **sp_drv_create_image_locator** 创建 `image_locator` 并选择某一值对其进行初始化。 **sp_drv_create_image_locator** 访问 Transact-SQL 函数 **create_locator**。

语法

```
sp_drv_create_image_locator [init_value]
```

输入参数
init_value - `varbinary` 或 `image` 值，用于初始化新定位符。

输出参数
无。

结果集
类型为 `image_locator` 的列。 定位符引用的 LOB 附带了 *init_value*。

从 Text 定位符中获取完整 Text 值
使用 **sp_drv_locator_to_text**，它可访问 Transact-SQL 函数 **return_lob**

语法
```
sp_drv_locator_to_text locator
```

输入参数
locator - 要检索其值的 `text_locator`。

输出参数
无。

结果集
含有 *locator* 所引用的 `text` 值的列。

从 Unitext 定位符中获取完整 Unitext 值
使用 **sp_drv_locator_to_unitext**，它可访问 Transact-SQL 函数 **return_lob**

语法
```
sp_drv_locator_to_unitext locator
```

输入参数
locator - 要检索其值的 `unitext_locator`。

输出参数
无。

结果集
含有 *locator* 所引用的 `unitext` 值的列。

从 Image 定位符中获取完整 Image 值
使用 **sp_drv_locator_to_image**，它可访问 Transact-SQL 函数 **return_lob**。

语法
```
sp_drv_locator_to_image locator
```

输入参数
locator - 要检索其值的 `image_locator`。

输出参数
无。

结果集
含有 *locator* 所引用的 `image` 值的列。

从 Text 定位符中获取子字符串

使用 **sp_drv_text_substring**，它可访问 Transact-SQL 函数 **substring**。

语法

```
sp_drv_text_substring locator, start_position, length
```

输入参数

- *locator* - 引用要处理的数据的 `text_locator`。
- *start_position* - 一个 `integer` 值，指定要读取和检索的第一个字符的位置。
- *length* - 一个 `integer` 值，指定要读取的字符数。

输出参数
无。

结果集
类型为 `text` 的列，其中包含检索到的子字符串。

从 Unitext 定位符中获取子字符串

使用 **sp_drv_unitext_substring**，它可访问 Transact-SQL 函数 **substring**。

语法

```
sp_drv_unitext_substring locator, start_position, length
```

输入参数

- *locator* - 一个 `unitext_locator` 值，引用要处理的数据。
- *start_position* - 一个 `integer` 值，指定要读取和检索的第一个字符的位置。
- *length* - 一个 `integer` 值，指定要读取的字符数。

输出参数
无。

结果集
类型为 `unitext` 的列，其中包含检索到的子字符串。

从 Image 定位符中获取子字符串

使用 **sp_drv_image_substring**，它可访问 Transact-SQL 函数 **substring**。

语法

```
sp_drv_image_substring locator, start_position, length
```

输入参数

- *locator* - 一个 `image_locator` 值，引用要处理的数据。
- *start_position* - 一个 `integer` 值，指定要读取和检索的第一个字节的位置。
- *length* - 一个 `integer` 值，指定要读取的字节数。

输出参数

无。

结果集

类型为 `image` 的列，其中包含检索到的子字符串。

在指定的位置插入 Text

使用 **sp_drv_text_setdata**，它可访问 Transact-SQL 函数 **setadata**。

语法

```
sp_drv_text_setdata locator, offset, new_data, data_length
```

输入参数

- *locator* - 一个 `text_locator` 值，引用要插入到的 `text` 列。
- *offset* - 一个 `integer` 值，指定开始写入新内容的位置。
- *new_data* - 要插入的 `varchar` 或 `text` 数据。

输出参数

data_length - 一个 `integer` 值，包含写入的字符数。

结果集

无。

在指定的位置插入 Unitext

使用 **sp_drv_unitext_setdata**，它可访问 Transact-SQL 函数 **setadata**。

语法

```
sp_drv_unitext_setdata locator, offset, new_data, data_length
```

输入参数

- *locator* - 一个 `unitext_locator` 值，引用要插入到的 `unitext` 列。
- *offset* - 一个 `integer` 值，指定开始写入新内容的位置。
- *new_data* - 要插入的 `univarchar` 或 `unitext` 数据。

输出参数
data_length - 一个 `integer` 值，包含写入的字符数。

结果集
无。

在指定的位置插入 Image
使用 **sp_drv_image_setdata**，它可访问 Transact-SQL 函数 **setadata**。

语法

```
sp_drv_image_setdata locator, offset, new_data, datalength
```

输入参数

- *locator* - 一个 `image_locator` 值，引用要插入到的 `image` 列。
- *offset* - 一个 `integer` 值，指定开始写入新内容的位置。
- *new_data* - 要插入的 `varbinary` 或 `image` 数据。

输出参数
data_length - 一个 `integer` 值，包含写入的字节数。

结果集
无。

插入定位符引用的 Text
使用 **sp_drv_text_locator_setdata**，它可访问 Transact-SQL 函数 **setadata**。

语法

```
sp_drv_text_locator_setdata locator, offset, new_data_locator,
data_length
```

输入参数

- *locator* - 一个 `text_locator` 值，引用要插入到的 `text` 列。
- *offset* - 一个 `integer` 值，指定开始写入新内容的位置。
- *new_data_locator* - 一个 `text_locator` 值，引用要插入的 `text` 数据。

输出参数
data_length - 一个 integer 值，包含写入的字符数。

结果集
无。

插入定位符引用的 Unitext
使用 **sp_drv_unitext_locator_setdata**，它可访问 Transact-SQL 函数 **setadata**。

语法

```
sp_drv_unitext_locator_setdata locator, offset, new_data_locator,
data_length
```

输入参数

- *locator* - 一个 unitext_locator 值，引用要插入到的 unitext 列。
- *offset* - 一个 integer 值，指定开始写入新内容的位置。
- *new_data_locator* - 一个 unitext_locator 值，引用要插入的 unitext 数据。

输出参数
data_length - 一个 integer 值，包含写入的字符数。

结果集
无。

插入定位符引用的 Image
使用 **sp_drv_image_locator_setdata**，它可访问 Transact-SQL 函数 **setadata**。

语法

```
sp_drv_image_locator_setdata locator, offset, new_data_locator,
   datalength
```

输入参数

- *locator* - 一个 image_locator 值，引用要插入到的 image 列。
- *offset* - 一个 integer 值，指定开始写入新内容的位置。
- *new_data_locator* - 一个 image_locator 值，引用要插入的 image 数据。

输出参数
data_length - 一个 integer 值，包含写入的字节数。

结果集
无。

截断基础 LOB 数据

使用 **truncate lob** 可截断 LOB 定位符引用的 LOB 数据。

请参见《Adaptive Server Enterprise 参考手册： 命令》。

查找基础 Text 数据的字符长度

使用 **sp_drv_text_locator_charlength** 查找 `text` 定位符引用的 LOB 列的字符长度。**sp_drv_text_locator_charlength** 访问 Transact-SQL 函数 **char_length**。

语法

```
sp_drv_text_locator_charlength locator, data_length
```

输入参数

locator - 一个 `text_locator` 值，引用要处理的 `text` 列。

输出参数

data_length - 一个 `integer` 值，指定 *locator* 引用的 `text` 列的字符长度。

结果集

无。

查找基础 Text 数据的字节长度

使用 **sp_drv_text_locator_bytelength** 查找 `text` 定位符引用的 LOB 列的字节长度。**sp_drv_text_locator_bytelength** 访问 Transact-SQL 函数 **data_length**。

语法

```
sp_drv_image_locator_bytelength locator, data_length
```

输入参数

locator - 一个 `text_locator` 值，引用要处理的 `text` 列。

输出参数

data_length - 一个 `integer` 值，指定 *locator* 引用的 `text` 列的字节长度。

结果集

无。

查找基础 Unitext 数据的字符长度

使用 **sp_drv_unitext_locator_charlength** 查找 `unitext` 定位符引用的 LOB 列的字符长度。 **sp_drv_unitext_locator_charlength** 访问 Transact-SQL 函数 **char_length**。

语法

```
sp_drv_unitext_locator_charlength locator, data_length
```

输入参数
locator - 一个 `unitext_locator` 值，引用要处理的 `unitext` 列。

输出参数
data_length - 一个 `integer` 值，指定 *locator* 引用的 `unitext` 列的字符长度。

结果集
无。

查找基础 Unitext 数据的字节长度
使用 **sp_drv_unitext_locator_bytelength** 查找 `unitext` 定位符引用的 LOB 列的字符长度。 **sp_drv_unitext_locator_bytelength** 访问 Transact-SQL 函数 **data_length**。

语法

```
sp_drv_image_locator_bytelength locator, data_length
```

输入参数
locator - 一个 `unitext_locator` 值，引用要处理的 `unitext` 列。

输出参数
data_length - 一个 `integer` 值，指定 *locator* 引用的 `unitext` 列的字节长度。

结果集
无。

查找基础 Image 数据的字节长度
使用 **sp_drv_image_locator_bytelength** 查找 `image` 定位符引用的 LOB 列的字符长度。 **sp_drv_image_locator_bytelength** 访问 Transact-SQL 函数 **data_length**。

语法

```
sp_drv_image_locator_bytelength locator, data_length
```

输入参数
locator - 一个 `image_locator` 值，引用要处理的 `image` 列。

输出参数
data_length - 一个 `integer` 值，指定 *locator* 引用的 `image` 列的字节长度。

结果集
无。

查找搜索字符串在定位符引用的 Text 列内的位置
使用 **sp_drv_varchar_charindex**，它可访问 Transact-SQL 函数 **charindex**。

语法

```
sp_drv_varchar_charindex search_string, locator, start, position
```

输入参数

- *search_string* - 要搜索的类型为 varchar 的文字。
- *locator* - 一个 text_locator 值，引用要在其中进行搜索的 text 列。
- *start* - 一个整数值，指定开始搜索的位置。 第一个位置是 1。

输出参数
position - 一个 integer 值，指定 *search_string* 在 *locator* 引用的 LOB 列中的开始位置。

结果集
无。

查找一个 Text 定位符引用的字符串在另一个定位符引用的 Text 列中的位置
使用 **sp_drv_textlocator_charindex**，它可访问 Transact-SQL 函数 **charindex**。

语法

```
sp_drv_textlocator_charindex search_locator, locator, start,
position
```

输入参数

- *search_locator* - 一个 text_locator 值，指向要搜索的文字。
- *locator* - 一个 text_locator 值，引用要在其中进行搜索的 text 列。
- *start* - 一个整数值，指定开始搜索的位置。 第一个位置是 1。

输出参数
position - 一个 integer 值，指定文字在 *locator* 引用的 LOB 列中的开始位置。

结果集
无。

查找搜索字符串在定位符引用的 Unitext 列内的位置
使用 **sp_drv_univarchar_charindex**，它可访问 Transact-SQL 函数 **charindex**。

语法

```
sp_drv_univarchar_charindex search_string, locator, start, position
```

输入参数

- *search_string* - 要搜索的类型为 `univarchar` 的文字。
- *locator* - 一个 `unitext_locator` 值，引用要在其中进行搜索的 `unitext` 列。
- *start* - 一个整数值，指定开始搜索的位置。 第一个位置是 1。

输出参数
position - 一个 `integer` 值，指定 *search_string* 在 *locator* 引用的 LOB 列中的开始位置。

结果集
无。

查找一个 Unitext 定位符引用的字符串在另一个定位符引用的 Unitext 列中的位置
使用 **sp_drv_unitext_locator_charindex**，它可访问 Transact-SQL 函数 **charindex**。

语法

```
sp_drv_charindex_unitextloc_in_locator search_locator, locator,
start,
position
```

输入参数

- *search_locator* - 一个 `unitext_locator` 值，指向要搜索的文字。
- *locator* - 一个 `unitext_locator` 值，引用要在其中进行搜索的 `text` 列。
- *start* - 一个整数值，指定开始搜索的位置。 第一个位置是 1。

输出参数
position - 一个 `integer` 值，指定文字在 *locator* 引用的 LOB 列中的开始位置。

结果集
无。

查找字节序列在 Image 定位符引用的列内的位置
使用 **sp_drv_varbinary_charindex**，它可访问 Transact-SQL 函数 **charindex**。

语法

```
sp_drv_varbinary_charindex byte_sequence, locator, start, position
```

输入参数

- *byte_sequence* - 要搜索的类型为 `varbinary` 的字节序列。
- *locator* - 一个 `image_locator` 值，引用要在其中进行搜索的 `image` 列。

- *start* - 一个整数值，指定开始搜索的位置。 第一个位置是 1。

输出参数
position - 一个 `integer` 值，指定 *search_string* 在 *locator* 引用的 LOB 列中的开始位置。

结果集
无。

查找一个 Image 定位符引用的字节序列在另一个定位符引用的 Image 列中的位置
使用 **sp_drv_image_locator_charindex**，它可访问 Transact-SQL 函数 **charindex**。

语法

```
sp_drv_image_locator_charindex sequence_locator, locator, start,
start_position
```

输入参数

- *sequence_locator* - 一个 `image_locator` 值，指向要搜索的字节序列。
- *locator* - 一个 `image_locator` 值，引用要在其中进行搜索的 `image` 列。
- *start* - 一个整数值，指定开始搜索的位置。 第一个位置是 1。

输出参数
start_position - 一个 `integer` 值，指定字节序列在 *locator* 引用的 LOB 列中的开始位置。

结果集
无。

检查 text_locator 引用是否有效
使用 **sp_drv_text_locator_valid**，它访问 **locator_valid**。

语法

```
sp_drv_text_locator_valid locator
```

输入参数
locator - 要验证的 `text_locator`。

输出参数
一个 `bit`，代表以下值之一：

- 0 - false，*locator* 无效。
- 1 - true，*locator* 有效。

结果集
无。

检查 unitext_locator 引用是否有效
使用 **sp_drv_unitext_locator_valid**，它访问 **locator_valid**。

语法
```
sp_drv_unitext_locator_valid locator
```

参数
locator - 要验证的 `unitext_locator`。

输出参数
一个 `bit`，代表以下值之一：

- 0 - false，*locator* 无效。
- 1 - true，*locator* 有效。

结果集
无。

检查 image_locator 引用是否有效
使用 **sp_drv_image_locator_valid**，它访问 **locator_valid**。

语法
```
sp_drv_image_locator_valid locator
```

参数
locator - 要验证的 `image_locator`。

输出参数
一个 `bit`，代表以下值之一：

- 0 - false，*locator* 无效。
- 1 - true，*locator* 有效。

结果集
无。

LOB 定位符释放
使用 **deallocate locator** 释放 LOB 定位符。

请参见《Adaptive Server Enterprise 参考手册： 命令》。

LOB 定位符示例

下面提供了六个 LOB 定位符示例。

示例 1

分配句柄并建立连接：

```
// Assumes that DSN has been named "sampledsn" and
// UseLobLocator has been set to 1.

SQLHENV environmentHandle = SQL_NULL_HANDLE;
SQLHDBC connectionHande = SQL_NULL_HANDLE;
SQLHSTMT statementHandle = SQL_NULL_HANDLE;
SQLRETURN ret;

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &environmentHandle);
SQLSetEnvAttr(environmentHandle, SQL_ATTR_ODBC_VERSION,
SQL_ATTR_OV_ODBC3);
SQLAllocHandle(SQL_HANDLE_DBC, environmentHandle,
&connectionHandle);
Ret = SQLConnect(connectionHandle, "sampledsn",
   SQL_NTS, "sa", SQL_NTS, "Sybase",SQL_NTS);
```

示例 2

选择列并检索定位符：

```
// Selects and retrieves a locator for bk_desc, where
// bk_desc is a column of type text defined in a table
// named books. bk_desc contains the text "A book".

SQLPrepare(statementHandle, "SELECT bk_desc FROM books
   WHERE bk_id =1", SQL_NTS);

SQLExecute(statementHandle);
BYTE TextLocator[SQL_LOCATOR_SIZE];
SQLLEN Len = 0;
ret = SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);

If(Len == sizeof(TextLocator))
{
   Cout << Locator was created with expected size <<
   Len;
}
```

示例 3

确定数据长度：

```
SQLLEN LocatorLen = sizeof(TextLocator);
ret = SQLBindParameter(statementHandle, 1,
   SQL_PARAM_INPUT, SQL_C_TEXT_LOCATOR,
   SQL_TEXT_LOCATOR, SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), &LocatorLen);
```

```
SQLLEN CharLenSize = 0;
SQLINTEGER CharLen = 0;
ret = SQLBindParameter(statementHandle, 2,
SQL_PARAM_OUTPUT, SQL_C_LONG,SQL_INTEGER,0 , 0,
&CharLen, sizeof(CharLen), &CharLenSize);
SQLExecDirect(statementHandle,
   "{CALL sp_drv_text_locator_charlength( ?,?) }" , SQL_NTS);

cout<< "Character Length of Data " << charLen;
```

示例 4

向 LOB 列中附加文本：

```
SQLINTEGER retVal = 0;
SQLLEN Col1Len = sizeof(retVal);
SQLCHAR appendText[10]=” abcdefghi on C++” ;

SQLBindParameter(statementHandle, 14,
   SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &retVal, 0,
Col1Len);

SQLBindParameter(statementHandle, 21, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, &TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 32, SQL_PARAM_INPUT,
   SQL_C_SLONG, SQL_INTEGER, 0, 0, &charLen, 0, SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 43, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 10, 0, append_text,
   sizeof(append_text), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle,
   "{? = CALL sp_drv_setdata_text (?, ?, ?,?) }" , SQL_NTS);

SQLFreeStmt(statementHandle, SQL_CLOSE);
```

示例 5

从 LOB 定位符中检索 LOB 数据。

```
SQLCHAR description[512];
SQLLEN descriptionLength = 512;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle, "{CALL sp_drv_locator_to_text(?)}",
SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, 1,SQL_C_CHAR, description,
```

```
   descriptionLength, &descriptionLength)

Cout << "LOB data referenced by locator:" << description
   << endl;

Cout << "Expected LOB data:A book on C++" << endl;
```

示例 6

将数据从客户端应用程序传输到 LOB 定位符。

```
description = "A lot of data that will be used for a lot
   of inserts, updates and deletes"; descriptionLength = SQL_NTS;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 512, 0, description,
   sizeof(description), &descriptionLength);

SQLExecDirect(statementHandle,
   "{CALL sp_drv_create_text_locator(?)}", SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);
```

用于 Python 的 Adaptive Server Enterprise 扩展模块

用于 Python 的 Adaptive Server Enterprise 扩展模块提供一个特定于 Sybase 的 Python 界面，用于针对 Adaptive Server 数据库执行查询。

此模块实施具有扩展的 Python Database API 规范 2.0 版，适用于 Python 2.6、2.7 和 3.1 版。 可以阅读 Python Database API 规范 http://www.python.org/dev/peps/pep-0249。

可以从 SDK 安装程序中安装用于 Python 的 Adaptive Server Enterprise 扩展模块。 有关安装说明，请参见《软件开发工具包和 Open Server 安装指南》和《软件开发工具包和 Open Server 发行公告》。 有关使用用于 Python 的 Adaptive Server Enterprise 扩展模块的信息，请参见《用于 Python 的 Adaptive Server Enterprise 扩展模块程序员指南》。

用于 PHP 的 Adaptive Server Enterprise 扩展模块

用于 PHP 的 Adaptive Server Enterprise 扩展模块提供一个界面，用于针对 Adaptive Server 数据库执行查询并处理查询结果，而且包括数据库访问所必需的 PHP API。

此模块用于 PHP 5.3.6 版。 有关使用用于 PHP 的 Adaptive Server Enterprise 扩展模块的信息，请参见《用于 PHP 的 Adaptive Server Enterprise 扩展模块程序员指南》。

用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序

用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序是通过通用 Perl DBI 接口调用的，并将 Perl DBI API 调用转换成一种可被 Adaptive Server 通过 Open Client SDK 使用 CT-Lib 理解的形式。

它让 Perl 脚本直接访问 Adaptive Server Enterprise 数据库服务器。 此驱动程序用于 Perl 5.14 版和 DBI 1.616 版。

可以在 http://search.cpan.org/~timb/DBI-1.616/DBI.pm 中阅读 Perl DBI 规范。 有关使用用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序的信息，请参见《用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序程序员指南》。

废弃的功能

Open Server 和 SDK 的当前版本不支持某些库和实用程序文件。

DCE 服务库

分布式计算环境 (DCE) 目录服务库 `libsybddce.dll` 和 DCE 安全性服务库 `libsybsdce.dll` 已从用于 Windows 32 位平台的 Open Client 和 Open Server 中删除。

在低于 15.7 版的 Open Client 和 Open Server 中，这些库位于 `%SYBASE%\OCS-15_0\dll` 目录中。

dsedit_dce 实用程序文件

dsedit_dce X-Windows 缺省值文件 `OCS-15_0/xappdefaults/Dsedit_dce` 和 **dsedit_dce** 帮助文件 `OCS-15_0/sybhelp/dsedit_dceHelpTextMsgs` 已删除。

不支持的平台

Open Server 和 SDK 不支持 HP-UX PA-RISC 和 Mac OS。

辅助功能特性

第 508 节要求美国联邦机构的电子和信息技术必须便于残障人士访问。 Sybase 坚决支持第 508 节要求，并已生产出一系列符合第 508 节要求的 Sybase 产品，其中包括 Open Client 和 Open Server 版本 15.7。

为便于访问，15.7 版本中的文档专门以 HTML 格式提供。 可以利用适应性技术（如屏幕阅读器）浏览 HTML 文档，也可以用屏幕放大器进行查看。 Open Client 和 Open Server 文档已经过测试，符合美国政府“第 508 节：辅助功能”的要求。 符合“第 508 节”的文档一般也符合非美国的辅助功能原则，如针对网站的 World Wide Web 协会 (W3C) 原则。

您可能需要对辅助功能工具进行配置以实现最优化。 一些屏幕阅读器根据大小写判断文本；例如将 ALL UPPERCASE TEXT 称为英文缩写，但将 MixedCase Text 称为单词。 对工具进行配置，规定语法约定，您可能会感觉更方便。 有关工具的信息，请查阅文档。

有关 Sybase 如何支持辅助功能的信息，请参见 Sybase 辅助功能。 Sybase 辅助功能网站包括指向“第 508 节”和 W3C 标准的相关信息的链接。

索引