新增功能公告
适用于 Windows、 Linux 和 UNIX 的 Open Server™ 15.7 和 SDK 15.7

文档 ID：DC00571-01-1570-02
最后修订日期：2012 年 6 月 29 日

产品平台和兼容性

表 1 列出了支持 Open Server™ 和 SDK 的平台：

表 1：支持 Open Server 和 SDK 的平台

平台
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

注释 并非所有 Open Server 和 SDK 组件都可在上面所列平台上使用。有关每个平台上可用组件的完整列表，请参见第 6 页上的 “产品组件”。

表 2 列出了构建和测试 Open Server 及 SDK 产品时所使用的平台、编译器和第三方产品：

表 2：Open Server 和 SDK 平台兼容性表

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 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不适用	不适用
HP-UX Itanium 64 位	HP 11.31	HP ANSI C A.06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.16（含有 OpenSSL 0.9.81）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)

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
IBM AIX 32 位	AIX 6.1	XL C 10.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.1	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不适用	不适用
IBM AIX 64 位	AIX 6.1	XL C 10.1	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Linux x86 32 位	Red Hat Enterprise Linux 5.3	gcc 4.1.2 20060404 kernel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	不适用	不适用
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 2.4.16（含有 OpenSSL 0.9.8l）	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 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不适用	不适用
Linux on POWER 64 位	Red Hat Enterprise Linux 5.3	XL C 10.1	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)
Microsoft Windows x86 32 位	Windows 2008 R2 Service Pack 1 Windows XP Service Pack 1 （仅限 ODBC/ OLE DB）	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 4.0、 MIT 2.6.4	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	不适用	不适用
Microsoft Windows x86-64 64 位	Windows 2008 R2 Service Pack 1 Windows XP Service Pack 1 （仅限 ODBC/ OLE DB）	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 2.1	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	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)

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 SPARC 32 位	Solaris 10	Solaris Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、 MIT 1.4.2	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不适用	不适用
Solaris SPARC 64 位	Solaris 10，修补程序级别为 144488-17 或更高；对于 *SUNWlibC*，修补程序级别为 119963-24 或更高	Solaris Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、 MIT 1.4.2	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	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 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	不适用	不适用
Solaris x86-64 64 位	Solaris 10	Solaris Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.16（含有 OpenSSL 0.9.8l）	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	不适用	5.3.6	2.6、2.7 和 3.1 (DBAPI 2.0)

图例：不适用 = 脚本在该平台上不可用或者无法与 SDK 一起工作。

注释 有关最新 Open Server 和 SDK 证书支持的信息，请参见 (http://certification.sybase.com/ucr/search.do) 上的 Sybase® 平台证书页。

注释 Microsoft 已结束对 Visual Studio 2005 的主流支持。尽管 SDK 当前仍支持 Visual Studio Compiler 2005 及更高版本，但还是建议您尽快更新到 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 日或之后的修补程序包）。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。

产品组件

Open Server 15.7 和 SDK 15.7 引入了新的功能，如 Bulk-Library select into 日志记录、大对象存储过程参数支持、对 Adaptive Server® Enterprise 中的非物化列的支持，以及对 jConnect™ for JDBC™ 和 Adaptive Server 驱动程序及提供程序的更新。Open Server 15.7 和 SDK 15.7 还支持将 Perl、PHP 和 Python 脚本化语言用于 Adaptive Server。

随 15.7 一起发布的产品有：

- Open Server
- 软件开发工具包
- SDK DB-Library Kerberos Authentication Option

Open Server

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 Open Client™ 或 jConnect for JDBC 例程提交的客户端请求。表 3 列出了 Open Server 组件和支持这些组件的平台。

表 3：Open Server 组件和支持的平台

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

软件开发工具包

软件开发工具包 (SDK) 是一组用于开发客户端应用程序的库和实用程序。表 4 列出了 SDK 组件和支持这些组件的平台。

表 4：SDK 组件和支持的平台

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

SDK 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
jConnect for JDBC	所有平台
Sybase 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
Sybase Adaptive Server Enterprise OLE DB 提供程序	• Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位

SDK 组件	平台
Adaptive Server Enterprise ADO.NET 数据提供程序	• Microsoft Windows x86 32 位 • Microsoft Windows x86-64 64 位
语言模块	所有平台
用于 Python 的 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 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 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

SDK DB-Library Kerberos Authentication Option

Sybase SDK DB-Library™ Kerberos Authentication Option 允许在 DB-Library 上使用 MIT Kerberos 安全机制，并且它可用于以下平台：

- Linux x86 32 位
- Microsoft Windows x86 32 位
- Solaris SPARC 32 位
- Solaris SPARC 64 位

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 文件具有下列更为严格的权限：

表 5：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)

注释 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)）。

注释 Microsoft Windows 上的文件的权限保持不变。

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，您可以使用新的 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。

```
%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* （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

说明

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。
- 仅在调用 ct_send() 结束命令后，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 并将参数发送到其数据驻留于不同内存位置的服务器。因此，该示例还演示如何在再次调用 ct_send_params() 之前，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

说明

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

说明

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

说明 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*

  由零个或更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 ASE 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。这减少了运行 alter table drop column 所需的时间量。请参见 ASE 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，以便实现完整的数据恢复。以前版本的 bcp 在 fast 模式下只记录页分配。请参见 ASE 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》。

例外处理

jConnect for JDBC 中的例外处理有所增强。当例外消息中含有使用 getcause() 的指令时，可使用 getCause() 方法获取导致例外的原因。

可提高性能的新 jConnect 连接属性

从 ESD #4 开始，jConnect for JDBC 具有下列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： • 0 - 在 select、 insert 和 update 操作中不抑制 TDS_PARAMFMT。 • 1 - 缺省值；在可能的情况下抑制 TDS_PARAMFMT。	1
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： • 0 - 即便行格式未发生更改，也发送 TDS_ROWFMT 或 TDS_ROWFMT2 数据。 • 1 - 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。	1

新的 jConnect 连接属性

从 ESD #4 开始， jConnect for JDBC 具有下列新连接属性：

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： • 0 - 缺省值；客户端发送的字符串值 “按原样” 存储。 • 1 - 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。	0

属性	说明	缺省值
SUPPRESS_ CONTROL_TOKEN	取消发送控制令牌。 值： • 0 - 缺省值；发送控制令牌。 • 1 - 取消发送控制令牌。	0

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

注释 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 (https://community.jboss.org/wiki/HibernateSybaseintegration) 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 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 版和更高版本的 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 连接属性以编程方式设置相关连接属性，以实现相同的效果：

```
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);
```

支持在不跟踪 ODBC 驱动程序管理器的情况下进行动态记录

Adaptive Server Enterprise ODBC 驱动程序 15.7 引入了在不跟踪 ODBC 驱动程序管理器的情况下进行应用程序记录的功能。请参见第 63 页上的 “没有 ODBC 驱动程序管理器跟踪的日志记录”。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。

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 配置文件不可与 SQL_OPT_TRACE 一起使用。

TDS 协议捕获的动态控制

使用 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 PDU 满包以精确捕获 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 和更高版本的功能。

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 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 Adaptive Server 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，请将新的 UseAseDecimal 连接属性设置为 1。缺省情况下，UseAseDecimal 设为 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 (Doc CR 705592)

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

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 Retry Count、在 LDAP 中指定为 sybaseRetryCount。 • 可将 RetryDelay 在 SQL.INI 中指定为 Loop Delay、在 LDAP 中指定为 sybaseRetryDelay。	否	0
SuppressControlTokens	指定 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 应用程序应仅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 应仅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
SuppressRowFormat2	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

自 ESD #4 起，用于 PHP 的 Adaptive Server 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 的永久连接。

API 类型	API	说明
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。
	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_unbuffered_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 变量与远程过程参数绑定。
	sybase_rpc_execute	（**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_array()	（**新增**）采用关联数组和 / 或数值数组的形式获取结果行。
	sybase_fetch_assoc()	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_object()	（**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_result()	释放与结果集相关联的全部内存。
	sybase_next_result()	（**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。
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*，它是 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
```

客户端拒绝的记录行

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 和 Open Server 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 的以下平台兼容：

- 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

您可以提高采用 ODBC 驱动程序时预准备语句的性能，方法是：在重新执行预准备语句时，抑制参数格式元数据。 Adaptive Server 15.7 ESD#1 及更高版本支持参数格式元数据抑制。

要抑制参数格式元数据，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。

注释 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的抑制， Adaptive Server 则忽略参数设置。

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;
SuppressParamFormat=1;
```

抑制行格式元数据以提高查询性能

您可以通过指示 Adaptive Server 抑制会话中重新执行的查询的行格式元数据 （TDS_ROWFMT 或 TDS_ROWFMT2），从而提高采用 ODBC 驱动程序和 ADO.NET 数据提供程序时的重复查询性能。 Adaptive Server 15.7 ESD#1 及更高版本支持行格式元数据抑制。

要抑制行格式元数据，则使用 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 不在可能情况下发送行格式元数据。

注释 仅当连接的 Adaptive Server 支持该功能时，才可抑制行格式元数据。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的抑制， Adaptive Server 则忽略该参数设置。

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

不得将 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》。有关用于 Python 的 Adaptive Server Enterprise 扩展模块的使用信息，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

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。此变量是从应用程序所在的目录中初始化的，而且位于环境变量 PYTHONPATH （使用和 shell 变量 PATH 相同的语法，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

本节介绍 Open Client 15.7 和 Open Server 15.7 中引入的新功能。

大对象定位符支持

Open Client 和 Open Server 15.7 版支持大对象 (LOB) 定位符。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 的类型常量有：

- 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。

表 6：支持的 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。
- 可以将传入 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 终结符计算在内。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。若要显式请求 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 属性供 Adaptive Server 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_CUSTOM_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 版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 版或更高版本以及 bcp 和 bulklib 15.7 版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

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）的应用程序特定设置解析 *ocs.cfg* 运行时配置文件，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 指定在将发现的无名应用程序（应用程序未显式设置 CS_APPNAME）的名称 （从操作系统获取的可执行程序名）转换为大写或小写后，是否按原样将此名称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，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 failed to create rows in target table.Column %1! would start at an offset over 8191 bytes; this starting location cannot be represented accurately in the table's (row) format.`
- `BCP failed to create rows in target table.Column %1! starts at an offset greater than %2! bytes; this starting location is not permitted by the current database configuration.`

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。

注释 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。

有关在游标关闭时释放锁的支持

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

示例 1 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);
```

示例 2 声明一个在关闭时释放共享锁的半不敏感可滚动游标：

```
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) 数据类型能否作为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。此功能在缺省情况下处于打开状态。

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，则会按数据流发送 LOB 数据，即使用一组 ct_send_data() 调用成块发送数据。块大小为零表示数据流结束。

示例 1 将少量 LOB 数据作为存储过程的输入参数发送：

```
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 数据是通过数据流发送到服务器的，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

Open Server 可以使用 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。请参见第 60 页上的“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 *dsn userid password*

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 *dsn userid password*

显示 Adaptive Server 版本以及该 Adaptive Server 上安装的 ODBC 和 OLEDB MDA 脚本的版本。参数 *dsn* 需要三个变量，分别是 Adaptive Server 的数据源名称、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。已经对此属性进行了更新，使得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。

没有 ODBC 驱动程序管理器跟踪的日志记录

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 日志的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 日志的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。

注释 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=false

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，SETMAXROWS_AFFECTS_SELECT_ONLY 会被忽略。

TDS ProtocolCapture

Adaptive Server Enterprise ODBC 驱动程序 15.7 引入了 ProtocolCapture 连接字符串属性，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。在处理事务量很大时，这可能会导致内存和资源的低效使用。

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 一直返回 SQL_BATCH_EXECUTING，直到最后一批参数被发送并处理。它在最后一批参数被处理后返回执行的状态。

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 Sybase 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。

  只有未提交的事务是可以回退的。
- SQL_BATCH_DISABLED – （缺省值） Adaptive Server Enterprise ODBC 驱动程序在处理最后一批参数后返回到此状态。可以手动将 SQL_ATTR_BATCH_PARAMS 设置为此值。

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);
```

考虑事项

- 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。

注释 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，客户端应用程序就无法将 LOB 和非 LOB 预准备语句 setter 方法混合在同一个批中。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，原因是 setString 方法和 setClob 方法用在同一列的同一批中。

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》。

有关在游标关闭时释放锁的支持

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》中的第 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。

对非物化列的支持

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 中处理非物化列。 Adaptive Server Enterprise ODBC 驱动程序的早期版本在表定义中包含非物化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物化列进行批量插入， 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 会在客户端物化 LOB 数据。可以将整个 LOB API 用于客户端物化 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 客户端无法使用 LOB 定位符，除非连接到支持它的 Adaptive Server。

注释 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。 Sybase 建议您让结果集小一些，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 的情况下建立与 Adaptive Server 的连接。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。

注释 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物化 LOB 定位符，即使连接的 Adaptive Server 能够支持它们时也是如此。这会增大客户端使用的内存，而且可能会降低性能。因此，建议您在设置了 autocommit off 时使用 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

若要启用 LOB 定位符支持，请在 EnableLOBLocator 连接属性设置为 1 的情况下建立与 Adaptive Server 的连接。当 EnableLOBLocator 设置为 0（缺省值）时， ODBC 驱动程序无法检索 LOB 列的定位符。当启用 LOB 定位符时，连接应设置为 autocommit off。

还必须在程序中包括 *sybasesqltypes.h* 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

用于 Adaptive Server 定位符数据类型的 ODBC 数据类型映射为：

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

Adaptive Server 定位符数据类型支持的转换为：

	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 ODBC API 方法

- SQLBindCol – *TargetType* 可以是任意 ODBC C 定位符数据类型。
- SQLBindParameter – *ValueType* 可以是任意 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 ODBC SQL 定位符数据类型。

请参见 *Microsoft ODBC API Reference*（《Microsoft ODBC API 参考》）。

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。

注释 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。

本节讨论如何对 LOB 定位符执行多种运算。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 可创建 text_locator 并 （可选）初始化以为其赋值。 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* – 一个 varchar 或 text 值，用于初始化新定位符。
输出参数	无。
结果集	一个类型为 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 可创建 unitext_locator 并 （可选）初始化以为其赋值。 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* – 一个 univarchar 或 unitext 值，用于初始化新定位符。

输出参数　　无。

结果集　　一个类型为 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 可创建 image_locator 并 （可选）初始化以为其赋值。 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* – 一个 varbinary 或 image 值，用于初始化新定位符。

输出参数　　无。

结果集　　一个类型为 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　　一个含有 *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　　一个含有 *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　一个含有 *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* – 一个 text_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　一个类型为 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　一个类型为 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 一个类型为 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 可查找 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 可查找 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 可查找 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 可查找 unitext 定位符引用的 LOB 列的字节长度。 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 可查找 image 定位符引用的 LOB 列的字节长度。 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。

释放 LOB 定位符

使用 deallocate locator。请参见 Adaptive Server Enterprise《参考手册：命令》。

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。可以在 (http://www.python.org/dev/peps/pep-0249) 中阅读 Python Database API 规范。

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。此模块适用于 PHP 5.3.6 版。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 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

当前版本的 Open Server 和 SDK 不支持以下功能。

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。

15.7 版本中的文档以 HTML 格式提供，专门为了便于访问。可通过适用的技术（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 World Wide Web 协会 (W3C) 针对 Web 站点的原则。

您可能需要对辅助功能工具进行配置以实现最优化。某些屏幕阅读器按照大小写来辨别文本，例如将 ALL UPPERCASE TEXT 看作首字母缩写，而将 MixedCase Text 看作单词。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 Accessibility (http://www.sybase.com/accessibility)。Sybase 辅助功能站点包括指向“第 508 节”和 W3C 标准的相关信息的链接。