SYBASE®
An SAP® Company

脚本化语言

用于 Perl 的 Adaptive Server® Enterprise 数据库驱动程序 15.7 SP100

文档 ID： DC01817-01-1570100-01
最后修订日期： 2013 年 5 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目录

用于 Perl 的 Adaptive Server Enterprise 数据库驱动程序

用于 Perl 脚本化语言的 Adaptive Server® Enterprise 数据库驱动程序能让 Perl 开发人员连接到 Adaptive Server 数据库以及使用 Perl 脚本查询或更改信息。

Perl 驱动程序模块

DBD::SybaseASE 是用于 Perl 脚本化语言的 Adaptive Server 数据库驱动程序。

用于 Perl 脚本化语言的 DBD::SybaseASE 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-Library 理解的形式。

通过使用 DBI 和 DBD::SybaseASE，您的 Perl 脚本可直接访问 Adaptive Server Enterprise 数据库服务器。

通用 Perl DBI API 规范定义一组方法，用以提供独立于实际使用数据库的数据库接口。

有关 Perl DBI 可编程 API 调用，可在 http://search.cpan.org/~timb/DBI-1.616/DBI.pm 中找到说明。

注意： DBD::SybaseASE 驱动程序无法在没有 DBI 的情况下正常工作。DBI 中包含所有用户可见 API。

必需组件

使用 Perl 编程语言访问 Adaptive Server 数据库需要以下组件：

- Perl 安装 – 数据库供应商未知的通用核心数据库 API。
- DBD::SybaseASE – 用于 Perl 脚本化语言的数据库驱动程序。
- CT-Library – (CT-Lib API) 是 Open Client 套件的一部分。CT-Library 向 Adaptive Server 发送命令并处理结果。
- Adaptive Server Enterprise
- Perl

版本要求

有关平台支持的信息，请参见所用平台的《软件开发工具包和 Open Server 安装指南》(Software Developers Kit and Open Server Installation Guide)。

- Adaptive Server Enterprise – 15.7 版或更高版本。
- Open Client 和 Open Server – 15.7 版或更高版本。

- Perl – 5.14.0 或 5.14.1 版。
- DBD::SybaseASE 驱动程序 – 没有特定的版本要求。
- CT-Library – (CT-Lib API) 15.7 版。
- Perl DBI – 1.616 版。

  Sybase® 安装程序不检查有无 Perl 安装或目标系统上是否安装了驱动程序依赖性。

注意： 针对您的平台发布的 Perl 驱动程序的建立模式还会决定 Perl 安装的建立模式和 DBI。作为示例，对于 Linux，此驱动程序是在启用线程化的情况下在 64 位模式中发布的。这意味着，必须在启用线程化的情况下在完全 64 位模式中配置 Perl。建立模式要求也适用于 DBI 接口。

安装和配置 Perl 驱动程序

用于 Perl 的数据库驱动程序是一个可通过 Sybase 安装程序安装的组件。

当您选择“自定义”**(Custom)** 作为安装类型时，用于 Perl 的数据库驱动程序是可选的安装组件。如果您选择的安装类型是“典型”**(Typical)** 或“完全”**(Full)**，则会缺省安装此驱动程序。有关安装和配置的说明，请参见针对所用平台的《软件开发工具包和 Open Server 安装指南》(Software Developers Kit and Open Server Installation Guide)。

开发 Perl 应用程序

使用 Perl DBI API 开发 Perl 应用程序。

支持 DSN 样式连接属性

驱动程序使用 DSN 机制以允许在连接时设置某些属性。

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

SybaseASE 驱动程序连接语法

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

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

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

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

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

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

属性和方法

当连接到服务器时，以下属性当前受到支持。

属性	说明
server	指定要连接到的服务器。 驱动程序当前假定已设置此选项。 如果未指定服务器，则使用 ENV{"DSQUERY"} 机制获取服务器名称。
database	指定服务器内在连接时充当目标数据库的数据库。 如果未指定数据库，则使用 master 数据库。
hostname	指定此进程的 `sysprocesses` 表的值部分中存储的主机名。 如果未指定主机名，则使用执行 Perl 应用程序的主机。
language	指定用于此连接的区域设置。 如果未指定语言，将使用名为 CS_LC_ALL 的内部缺省区域设置。
charset	指定用于此连接的字符集。 如果未指定字符集，将使用内部缺省值 **utf8**。
host; port	指定要使用的主机和端口组合，而不是依靠 interfaces 文件条目。 **注意:** 在 Perl DSN 语法中，host 和 port 是不同的选项。 当前不支持如下备用 DSN 形式: `host:port=mumbles:1234` 如果由于不想使用 interfaces 文件而指定 DSN 选项 host 和 port，则该 host 和 port 必须能够满足连接条件。 如果除 host 和 port 组合外还提供了 DSN 属性“server=”，则连接将失败。 因此，要么使用主机和端口建立连接，要么使用服务器建立连接。 两个 DSN 属性（server 和 host/port）互相排斥。
timeout	指定连接超时值。 设置为 0 或负值表示无超时值。
loginTimeout	指定登录超时值（秒）。 缺省值是 60 秒。 设置 **loginTimeout=*value in seconds*** 以启用此属性。
tds_keepalive	指定用于连接的 KEEP_ALIVE 属性。 设置 **tds_keepalive=*1*** 以启用此属性。
packetSize	指定用于连接的 TDS 包大小。 缺省情况下，驱动程序中设置的下限为 2048. 最大值由服务器确定，不在驱动程序中设置。
maxConnect	增加或减少允许的连接数。 值范围为 1 到 128；缺省值为 25。

属性	说明
encrypt-Password	指定是否使用口令加密。 设置 **encryptPassword=*1*** 以启用此属性。
sslCAFile	指定 `trusted.txt` 文件的替代位置。 指定最多为 256 个字符的绝对路径。
script-Name	指定驱动应用程序的顶级 Perl 脚本的选定名称。 此名称将作为应用程序名称显示在 `sysprocesses` 表中。 如果不指定该值，将使用从 Perl 内部环境获得的缺省应用程序名称。 此值最多可包含 256 个字符。 **注意：** 送入 SybaseASE 驱动程序的应用程序名称或通过 DSN **scriptName** 选项进行设置，或派生自 Perl 内部环境。
interfaces	指定 Sybase interfaces 文件的替代位置。 **sslCAFile** 和 **scriptName** 选项也具有同样约束。

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

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

特定于 DSN 的示例：

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

或者使用 DSQUERY 环境变量：

```
my $srv = $ENV{"DSQUERY"};
$dbh = DBI->connect("dbi:SybaseASE:server=$srv",  $user, $passwd);
$dbh = DBI-
>connect("dbi:SybaseASE:host=tzedek.sybase.com;port=8100", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:maxConnect=100", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:database=sybsystemprocs", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:charset=iso_1", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:language=us_english", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:packetSize=8192", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:interfaces=/opt/sybase/
interfaces", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:loginTimeout=240", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:timeout=240", $user, $passwd);
$dbh = DBI->connect("dbi:Sybase:scriptName=myScript", $user,
$password);
$dbh = DBI->connect("dbi:SybaseASE:hostname=pedigree", $user,
$password);
$dbh = DBI->connect("dbi:SybaseASE:encryptPassword=1", $user,
$password);
$dbh = DBI>connect("dbi:SybaseASE:sslCAFile=/usr/local/sybase/
trusted.txt", $user, $password,
AutoCommit => 1);
```

特定于 DSN 的示例组合：

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

当前不支持的 DSN 语法
当前不支持以下 DSN 语法：

- **tdsLevel**
- **kerberos**；例如：

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

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

当前支持的数据库句柄属性

下表列出了当前支持的数据库句柄属性。

属性	说明	缺省值
dbh->{AutoCommit} = (0|1);	禁用或启用 AutoCommit。	0（关闭）
dbh->{LongTruncOK} = (0|1);	禁用或启用 `text` 和 `image` 类型的截断。	0
dbh->{LongRea-dLen}=(int);	设置 `text` 和 `image` 数据的缺省读取块大小。 例如： **dbh->{LongReadLen} = 64000**。	32767
dbh->{syb_show_sql} =(0|1);	设置后，当前语句会包含在 **$dbh->errstr** 机制返回的错误字符串中。	0
dbh->{syb_show_eed} = (0|1);	设置后，扩展错误信息会包含在 **$dbh->errstr** 返回的错误字符串中。	0
dbh->{syb_chained_txn} = (0|1);	设置后，AutoCommit 关闭时将使用 CHAINED 事务。 仅在调用 **connect()** 时使用此属性： `$dbh = DBI->connect("dbi:SybaseASE:", $user, $pwd, {syb_chained_txn => 1});` 在关闭 AutoCommit 的情况下，任何时候使用 **syb_chained_txn** 都将强制在当前句柄上执行提交。 设置为 0 时，会根据需要发出显式 **BEGIN TRAN**。	0

属性	说明	缺省值
dbh->{syb_use_bin_0x} = (0|1);	设置后，结果字符串中的 BINARY 和 VARBINARY 值会带有“0x”前缀。	0
dbh->{syb_binary_images} = (0|1);	设置后，将以原始二进制格式返回 image 数据。 否则，image 数据会转换为十六进制字符串。	0
dbh->{syb_quoted_identifier} =(0|1);	在使用“标识符”进行引用时，允许使用与 Sybase 保留字冲突的标识符。	0
dbh->{syb_row-count}=(int);	如果设置为非零值，则 **SELECT** 返回的行数，或受 **UPDATE** 或 **DELETE** 语句影响的行数将限制为 *rowcount* 值。 将其重置为 0 会清除此限制。	0
dbh->{syb_flush_finish} = (0|1);	设置后，驱动程序会通过实际读取当前命令的所有剩余结果来清除它们。 可使用此设置替代驱动程序发出的 **ct_cancel()** 命令。	0
dbh->{syb_date_fmt} = datefmt string	此私有方法设置缺省日期转换和显示格式。 请参见“缺省日期转换和显示格式”。	
dbh->{syb_err_handler}	可以创建该 Perl 子例程以在进行常规错误处理前执行错误处理程序或进行报告。 对某类警告很有用。 请参见“错误处理”。	0（不存在）
dbh->{syb_failed_db_fatal} = (0|1)	如果 DSN 具有 **database=*mumbles*** 属性/值对，并且此数据库在连接时不存在，则 **DBI->connect()** 调用将失败。	0
dbh->{syb_no_child_con} =(0|1);	设置后，驱动程序将不允许在 **dbh** 上存在多个活动语句句柄。 在这种情况下，可以预准备某个语句，但必须执行完该语句才能尝试准备另一语句。	0
dbh->{syb_cancel_request_on_error}= (0|1);	设置后，当执行多个语句集且其中一个语句失败时，**sth->execute()** 也将失败。	1（打开）
dbh->{syb_bind_empty_string_as_null}= (0|1);	设置后，NULLABLE 列属性将返回一个空字符串（一个空格）用来表示 NULL 字符。	0
dbh->{syb_disconnect_in_child} = (0|1);	处理分叉上已关闭的连接。 如果子连接已断开，则 DBI 将导致连接关闭。	0
dbh->{syb_enable_utf8} = (0|1);	设置后，UNICHAR、UNIVARCHAR 和 UNITEXT 将转换为 **utf8**。	0
sth->syb_more_results} = (0|1);	请参见“多个结果集”。	

属性	说明	缺省值
sth->{syb_result_type} = (0|1);	设置后，将返回数值结果编号，而不是符号式 CS_ version。	0
sth->{syb_no_bind_blob} = (0|1);	设置后，在执行 **sth->{fetch}** 或其它变化形式时将不会返回 `image` 或 `text` 列。 请参见“文本和图像数据处理”。	0
sth->{syb_do_proc_status} = (0|1);	强制 **$sth->execute()** 读取在 SQL 流中执行的存储过程的返回状态。 如果返回状态非零，则 **$sth->execute()** 会返回 `undef`（即失败）。 设置此属性不会影响现有语句句柄。 但是，将会对设置后创建的语句句柄产生影响。 要恢复现有 **$sth** 句柄的行为，请执行： **$sth->{syb_do_proc_status} = 0**;	0

另请参见

- 错误处理 （第 17 页）
- 文本和图像数据处理 （第 15 页）
- 缺省日期转换和显示格式 （第 14 页）
- 使用多个语句 （第 8 页）

不支持的数据库句柄选项

不支持以下数据库句柄选项。

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

注意： 如果 Perl 脚本尝试使用这些选项，将生成错误。

Perl 支持的数据类型

Perl 驱动程序当前支持字符串、数值以及日期和时间数据类型。

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

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

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

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

另请参见

- 缺省日期转换和显示格式 （第 14 页）

使用多个语句

Adaptive Server 可在单个批处理中处理多语句 SQL。

例如：

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

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

多个结果集

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

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

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

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

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

多个结果集示例：

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

如果您需要多个结果集，Sybase 建议您使用此功能。

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

在 DatabaseHandle (dbh) 上使用多个活动语句

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

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

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

支持的字符长度

不同类型标识符支持的字符长度。

Sybase 标识符（如表和列）的名称可以在长度上超过 255 个字符。

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

配置区域设置和字符集

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

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

动态 SQL 支持、占位符和绑定参数

Perl 驱动程序支持动态 SQL（包括参数用法）。

例如：

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

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

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

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

以下为展示动态 SQL 支持的另一个示例：

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


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

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


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

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

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

以下为演示“?”样式占位符的另一个示例：

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

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

存储过程对占位符的支持

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

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

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

示例

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

以下示例说明了位置参数的用法：

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

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

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

如果存储过程使用输出参数返回数据，则必须先对其进行声明：

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

不能调用带有绑定参数的存储过程，如下所示：

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

工作方式如下：

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

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

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

参数示例

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

如果存储过程仅返回 OUTPUT 参数，可使用：

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

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

通用示例

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

OUTPUT 参数将作为特殊结果集中的单独一行返回。

参数类型

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

驱动程序支持以下 ODBC 样式的绑定类型：

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

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

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

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

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

以下面的存储过程定义为例：

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

将生成以下错误：

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

如下所示设置 **arithabort** 选项以忽略这些错误：

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

请参见 Adaptive Server 参考文档。

支持的私有驱动程序方法

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

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

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

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

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

请参见 Open Client 文档。

缺省日期转换和显示格式

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

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

此缺省区域设置还支持下列其它几种输入格式：

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

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

表 1. 支持的日期/时间格式

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

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

文本和图像数据处理

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

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

不能使用绑定参数插入文本或图像数据。

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

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

示例

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

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

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

调用顺序如下：

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

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

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

其它 TEXT/IMAGE API

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

例如：

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

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

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

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

在此示例中，考虑更新 id 列为 1 的图像列中的数据：

1. 为数据查找 CS_IODESC 数据：

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

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

2. 使用 CS_IODESC 值进行更新：

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

3. 设置将要插入的新数据项的大小，并不对此操作进行记录：

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

4. 要在单个块中传输数据，请执行：

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

5. 要提交操作，请执行：

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

错误处理

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

但驱动程序启动期间必须报告的错误或警告除外，因为这时尚无可用的上下文。

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

下面的示例显示了如何添加更多的详细错误消息（服务器消息）：

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

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

配置安全服务

使用 `ocs.cfg` 和 `libtcl.cfg` 文件配置安全选项。

1. 对于连接，使用 `ocs.cfg` 设置目录和安全属性。

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

2. 编辑 `libtcl.cfg` 以装载安全和目录服务驱动程序。

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

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

示例

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

示例 1

使用示例程序查看 Perl 中的存储过程的基本用法。

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

```
use strict;

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

require_version DBI 1.51;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

dump_info($sth);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

示例 2

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

```
use strict;
```

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

require_version DBI 1.51;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lastname            City
```

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

format STDOUT =

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

$dbh->disconnect;
```

Perl 错误消息

如果内部驱动程序 API 或 CT-Lib's API 失败，则 Perl 驱动程序将生成错误消息。 例如：

示例 1，消息 id 3

```
 OpenClient message: LAYER = (1) ORIGIN = (1) SEVERITY = (1) NUMBER =
(3)
 Message String: s_command_alloc() : SybaseASE : Internal error:

        The ct_cmd_alloc() api failed.
```

示例 2，消息 id 14

```
 OpenClient message: LAYER = (1) ORIGIN = (1) SEVERITY = (1) NUMBER =
(14)
 Message String: s_db_connect() : SybaseASE : Internal error:
        cannot change to database tempdb.
```

下表包含四个字段：

- 消息 ID，显示在 NUMBER = (<num>) 字段
- 带有参数的消息文本在驱动程序中的特定代码位置进行替换
- 严重级为警告或致命
- 可能的修复或解释

消息 ID	消息文本	严重级	修复/解释
1	%1! 无法分配内存。	致命错误	检查系统和内存资源。 %1 可包含 **cs_con_alloc()**、**ct_conn_alloc()** 和 **malloc()** 等函数名。
2	%1! 句柄为空值。	致命错误	内部驱动程序错误。 连接或 cmd 句柄为 NULL

消息 ID	消息文本	严重级	修复/解释
3	%1! api 已失败。	致命错误	内部驱动程序错误。 可由无效的 DSN 连接字符串或内部 CT-Lib 引发。 错误字符串包含失败 API 的函数名。
4	语句 id %1! 的句柄为空值。	保留	保留
5	语句 id %1! 失败。	致命错误	内部错误。 **ct_dynamic()** API 尝试释放失败的语句 id。
6	语句 id %1! 发送失败。	保留	保留
7	%1! 必须大于 0。	致命错误	DSN 字符串验证失败。 检查 DSN 字符串是否包含非法字符。
8	%1! 必须小于等于 %2!，设置为最大允许值。	警告	尝试配置的连接数超出当前允许的连接数 (128)。
9	ct_config(CS_SET, %1!) 失败。 提供的选项 %2! 非法或缺失。	致命错误	**ct_config()** API 失败，因为给定的选项无效。 检查 DSN 字符串和第二个参数中的字符串。
10	cs_locale(CS_SET, %1! %2!) 失败。	致命错误	**cs_locale()** API 失败。 错误字符串表示失败发生的位置。
11	cs_dt_info(CS_SET, CS_DT_CONVFMT) 失败。	致命错误	错误字符串将表示错误发生的位置。
12	ct_debug(CS_SET CS_DBG_PROTOCOL) 失败。	保留	保留
13	ct_con_props(CS_SET, %1!) 失败。 提供的选项 %2! 非法或缺失。	致命错误	**ct_con_props()** API 失败。 错误字符串表示失败发生的位置。
14	无法更改为数据库 %1!。	致命错误	驱动程序失败。 检查 DSN 以查看是否存在给定的数据库名称。
15	ct_command() 对 %1! 失败。	致命错误	**ct_command()** API 失败。 错误字符串表示失败的 CMD 类型。
16	ct_send() 对 %1! 失败。	致命错误	**ct_send()** API 失败。 %1 中包含的错误字符串将给出详细信息。

消息 ID	消息文本	严重级	修复/解释
17	ct_describe() 对列 %1! 失败。	致命错误	**ct_describe()** API 失败。 错误字符串表示错误发生的位置。
18	描述列 %2! 时 ct_compute_info() 在列 %1! 上失败。	致命错误	**ct_compute_info()** API 失败。 错误字符串表示涉及的列编号和操作类型。
19	转换失败 %1!。	警告	**cs_convert()** 已失败。 错误字符串表示错误发生的位置。
20	ct_param() 失败。	致命错误	**ct_param()** API 失败。 错误字符串表示失败位置。
21	ct_command() %1! 失败：语句 %2!。	致命错误	**ct_command()** API 失败。 错误字符串表示失败的 CMD 类型（包括语句）。
22	ct_results() 失败：%1!。	致命错误	**ct_results()** API 失败。 错误字符串表示驱动程序功能和失败的语句。
23	启用 autocommit 的情况下，%1! 命令无效。	警告	错误字符串表示尝试的 commit 或 rollback 无效。
24	ct_dynamic(CS_PREPARE) 在语句 %1! 上失败。	致命错误	Dynamic Prepare 失败。 错误字符串中提供了语句名称。
25	ct_dynamic(CS_DESCRIBE INPUT) 在语句 %1! 上失败。	致命错误	Dynamic Describe 失败。 错误字符串中提供了语句名称。
26	ct_dynamic(CS_EXECUTE) 在语句 %1! 上失败。	致命错误	Dynamic Execute 失败。 错误字符串中提供了语句名称。
27	%1! 数据库句柄处于非活动状态，未连接到服务器。	致命错误	尝试使用无效数据库句柄或非活动连接与服务器建立连接。
28	不允许子连接。	致命错误	如果数据库句柄处于活动状态且正在使用，则不允许子连接。
29	无法绑定占位符 %1!。	致命错误	尝试绑定占位符时出错。 错误字符串表示语句。
30	意外取消。	致命错误	处理行时遇到意外取消类型。

消息 ID	消息文本	严重级	修复/解释
31	来自 %1! 的意外返回代码。	致命错误	处理行时遇到意外返回代码。
32	为 syb_date_fmt 提供的格式 %1! 无效。	致命错误	在日期或时间转换前给定的日期格式无效。
33	致命错误：在未启用 autocommit 的情况下数据库句柄上存在多个活动语句句柄。	致命错误	用户错误；在用户 Perl 脚本中使用了多个活动句柄，但未启用 autocommit。
34	致命错误：提供的 DSN 选项无效或不受支持。	致命错误	如果 Perl 脚本具有不受支持或已过时的选项，DSN 分析将发生致命错误。

其它资源

有关使用 Perl 驱动程序的其它信息。

- DBI 驱动程序的构建、测试和安装：
  http://dbi.perl.org/
- Perl DBI 用户可编程 API 调用：
  http://search.cpan.org/~timb/DBI-1.616/DBI.pm
- 有关配置信息的 Open Client 和 Open Server 文档：
  《适用于 UNIX 的 Open Client 和 Open Server 配置指南》>“配置文件”
- 初始化应用程序，以使它从系统配置的角度是使用特定语言和相关文化习惯执行的：
  《适用于 UNIX 的 Open Client 和 Open Server 的配置指南》>“本地化”
- 适用于所有 Open Client 和 Open Server 产品的平台相关问题：
  《适用于 UNIX 的 Open Client 和 Open Server 的程序员补充说明》
- 使用 Open Client 和 Open Server 运行时配置文件：
  《Open Client Client-Library/C 参考手册》>“使用运行时配置文件”(Using the runtime configuration file) >“Open Client 和 Open Server 运行时配置文件语法”(Open Client and Open Server runtime configuration file syntax)
- 使应用程序能够支持多种语言和文化习惯：
  《适用于 UNIX 的 Open Client 和 Open Server 国际开发人员指南》(Open Client and Open Server International Developers Guide for UNIX) >“了解国际化和本地化”(Understanding Internationalization and Localization)
- 平台支持：

针对所用平台的《软件开发工具包和 Open Server 安装指南》(Software Developer Kit and Open Server Installation Guide)。

词汇表

特定于脚本化语言的术语词汇表。

- **Client-Library** – Open Client 的一部分，一组用于编写客户端应用程序的例程。Client-Library 专用于容纳游标和 Sybase 产品系列中的其它高级功能。
- **CPAN** – 综合 Perl 存档网络。用于存储大量 Perl 软件和文档的 Web 站点。请访问 http://www.cpan.org。
- **CS-Library** – 在 Open Client 和 Open Server 产品中附带提供，一组对 Client-Library 和 Server-Library 应用程序都可用的实用程序例程。
- **CT-Library** – (CT-Lib API) 是 Open Client 套件的一部分，是让脚本化应用程序连接到 Adaptive Server 所必需的。
- **DBD** – 特定于数据库供应商的驱动程序，将 DBI 数据库 API 调用转换为可被目标数据库 SDK 理解的形式。
- **DBI** – 通用核心数据库 API，数据库供应商未知，是 Perl 应用程序中的数据库访问的最新标准。请访问 http://dbi.perl.org。
- **驱动程序** – 用于构建 DBD::SybaseASE 的 Perl 和 C 代码集合。
- **扩展或模块** – 可以通过用 Perl 或者 Perl 与 C 组合编写的模块来扩展 Perl 语言。在本文档中，扩展和模块是同一个意思。
- **Perl 目录树** – 为以下各项之一：
  - 完整 Perl 安装，当系统经过配置并装有操作系统时，它安装为二进制模块。完整 Perl 安装有时叫做系统 (Perl) 树，由系统帐户（root、admin）拥有。
  - 私有 Perl（目录）树，由系统帐户以外的用户根据源构建，通常安装在系统 Perl 树以外的目录中。这允许在不破坏系统树的情况下使用新功能和错误修复测试。私有目录树通常由构建该树的帐户拥有。
- **Perl 脚本** – Perl 是一种脚本化语言，广泛用于系统和数据库管理中。请访问 http://www.perl.org。
- **线程** – 是通过 Open Server 应用程序和库代码完成的执行的路径，以及路径的关联堆栈空间、状态信息和事件处理程序。
- **Transact-SQL** – 数据库语言 SQL 的增强版。应用程序可以使用 Transact-SQL 与 Adaptive Server Enterprise 通信。

索引