SYBASE®
An SAP® Company

程序员指南

用于 Python 的 Adaptive Server® Enterprise 扩展模块 15.7 SP100

文档 ID： DC01821-01-1570100-01
最后修订日期： 2013 年 5 月

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

目录

用于 Python 的 Adaptive Server Enterprise 扩展模块

用于 Python 的扩展模块 (sybpydb) 提供一个特定于 Sybase® 的 Python 界面，用于针对 Adaptive Server® Enterprise 数据库执行查询。

此扩展模块实施具有扩展的 Python Database API 规范 2.0 版。有关 API 规范的详细信息，请参见 http://www.python.org/dev/peps/pep-0249。

必需组件

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

- sybpydb - 用于 Python 脚本化语言的扩展模块。
- Open Client SDK - 提供应用程序开发工具，用以访问数据源、信息应用程序或系统服务。

版本要求

用于 Python 的 Adaptive Server Enterprise 扩展模块的版本要求如下所示。

- Adaptive Server Enterprise - 15.7 或更高版本
- Python 安装 - 2.6、2.7 或 3.1 版内置线程化模式
- Open Client SDK - 15.7 或更高版本

注意： 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)。

安装用于 Python 的扩展模块

用于 Python 的扩展模块是一个可通过 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 Python 的扩展模块是可选的安装组件。 如果您选择的安装类型是“典型”(Typical) 或“完全”(Full)，则会缺省安装此扩展模块。 有关完整安装和配置说明，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)。

用于 Python 的扩展模块的配置概述

完成 Python 应用程序的基本配置任务以建立连接和执行命令。

- Python 模块搜索路径
- 目标服务器的名称和地址
- 安全和目录服务
- 通过 `ocs.cfg` 文件进行的运行时配置

Python 模块搜索路径

Python 在 Python 变量 `sys.path` 给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，或者如果找不到该文件，则搜索在与安装有关的缺省路径中继续。

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

目标服务器名称和地址

当 Python 应用程序连接到 Adaptive Server 时，将从以下其中一个来源获取目标服务器的名称。

1. 服务器名称（如果已在 **connect** 方法中指定）。

2. DSQUERY 环境变量（如果应用程序不在 **connect** 方法中指定目标服务器）。
3. SYBASE 环境变量（如果未设置 DSQUERY）。

目标服务器的地址是从目录服务或从与平台有关的 interfaces 文件中获得的。 在 interfaces 文件或 LDAP 目录服务中创建一个服务器条目。 请参见所用平台的《Open Client 和 Open Server 配置指南》(Open Client and Open Server Configuration Guide)。

安全和目录服务

目录驱动程序或安全驱动程序应在 `libtcl.cfg` 文件中进行配置，具体位置如下：

- [DIRECTORY] 部分的目录驱动程序。
- [SECURITY] 部分的安全性驱动程序。

请参见所用平台的《Open Client 和 Open Server 配置指南》(Open Client and Open Server Configuration Guide)。

运行时配置

使用运行时配置文件 `ocs.cfg` 设置这些值。

- 属性值
- 服务器选项值
- 服务器功能
- 调试选项

有关文件语法以及可在该文件中设置的属性的信息，请参见《Open Client Client-Library/C 参考手册》中的“使用 Open Client 和 Open Server 运行时配置文件”(Using the Open Client and Open Server Runtime Configuration File)。

开发 **Python** 应用程序

有关如何使用 sybpydb 模块编写 Python 应用程序的概述。

线程安全性

线程可共享模块。 但是，某个连接、其属性以及使用此连接创建的任何对象在不使用锁定机制时均无法共享。

例如，游标即属于根据连接创建的对象。 要在多个线程间使用连接，该连接（其属性和创建的对象）必须使用信号进行包装才能实现资源锁定。

参数样式

此模块支持问号样式的参数标记格式。

装载用于 Python 的扩展模块

使用 import 语句装载用于 Python 的扩展模块。
要通过 Python 脚本使用 sybpydb 模块，必须先装载该模块，即通过以下方式将其导入：

```
import sybpydb
```

使用 DSN 样式的连接字符串属性建立和关闭到 Adaptive Server 的连接

使用 **connect** 方法打开数据库连接。 此方法支持 DSN 样式的连接属性。

此方法接受以下关键字参数：

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

    注意： 如果存在没有值的布尔型名称，则必须将布尔类型设置为 true。

  例如：

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

有效的属性名称和值

下表列出了 **dsn** 关键字参数的有效属性名称和值。

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

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

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

批量复制支持

对批量复制操作的支持是 Python DBAPI 的一种扩展功能。 可用于批量复制行。

批量复制操作示例：

```
import syDbpydb
conn =
sybpydb.connect(dsn="user=john;bulklogin=true;chainxacts=off")
cur = conn.cursor()
cur.execute("create table mytable (id int identity, name
varchar(20))")
cur.close()

blk = conn.blkcursor()
blk.copy("mytable", direction="in", )
blk.rowxfer(["Mark"])
blk.rowxfer(["Leanne"])
blk.rowxfer(["Stanley"])
blk.done()
blk.close()

conn.close()
```

使用 Python 访问和更新数据

建立连接后，使用游标对象管理读取操作的上下文。

游标对象提供准备和执行查询的方法，并可以从结果集中读取行数据。游标对象是使用 **cursor** 方法从连接对象中获得的。此示例显示应用程序如何访问和更新数据：

```
import sybpydb

   #Create a connection.
   conn = sybpydb.connect(user='sa')

   # Create a cursor object.
   cur = conn.cursor()

   cur.execute("drop table footab")
   cur.execute("create table footab ( id integer, first char(20)
null, last char(50) null)")
   cur.execute("insert into footab values( ?, ?, ? )", (1, "John",
"Doe"))
   cur.execute("select * from footab")
   rows = cur.fetchall()
   for row in rows:
       print "-" * 55
```

```
       for col in range (len(row)):
          print "%s" % (row[col]),

   #Close the cursor object
   cur.close()

   #Close the connection
   conn.close()
```

将输入和输出参数传递到存储过程

从 15.7 ESD#3 版本开始，用于 Python 的 Adaptive Server Enterprise 扩展模块支持将输入和输出参数传递到存储过程。

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)
```

扩展模块提供了可带有 OutParam 参数的构造方法用于指定输出参数。该支持是对 Python DBAPI 规范的扩展。**callproc()** 方法返回所有传递到该方法的参数的列表。如果存在输出参数，而不存在从存储过程生成的结果集，则 **callproc()** 完成后，列表中将包含经修改的输出值。但是，如果存在结果集，则直到使用 **fetch*()** 方法检索来自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，列表中才会包含经修改的输出值。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` 中提供了更多不同输出参数类型的示例。

计算行处理

从 15.7 ESD#3 版本开始，用于 Python 的 Adaptive Server Enterprise 扩展模块支持计算行处理。

示例程序 `compute.py` 中提供了示例。

动态语句和存储过程的参数支持

从 15.7 ESD#4 版本开始，用于 Python 的 Adaptive Server Enterprise 扩展模块支持 decimal、money 和 LOB 作为参数用于动态语句和存储过程。

Decimal 和 money 类型参数

以下是 decimal 和 money 用作存储过程的参数的示例用法：

```
cur.execute("""
    create procedure pyproc
    @m1 money,
    @m2 money output,
    @d1 decimal(5,3),
    @d2 decimal(5,3) output,
    as
    begin
        select @d2 = @d1
        select @m2 = @m1
    end
    """)

dec_in = decimal.Decimal('1.23')
dec_out = sybpydb.OutParam(decimal.Decimal('0.00'))

dec_in = decimal.Decimal('1.23')
dec_out = sybpydb.OutParam(decimal.Decimal('0.00'))
m_in = decimal.Decimal('9.87')
m_out = sybpydb.OutParam(decimal.Decimal('0.00'))
vals = cur.callproc('pyproc', (int_out, dec_in, dec_out, m_in,
m_out))
```

```
print ("Status = %d" % cur.proc_status)
print ("decimal = %s" % vals[1])
print ("money= %s" % vals[3])
```

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

请参见演示使用不同数据类型参数（包括 date、time、datetime、float, 和 integer）调用存储过程的 `callproc.py` 示例。该示例还演示了输出参数处理。

用于 Python 的扩展模块的 API 参考

sybpydb 扩展接口 API。

模块接口方法

用于模块接口的 API。

connect

构建一个连接对象，用以代表与数据库的连接。此方法接受关键字参数：

- **user** – 连接用来登录到服务器的用户登录名。
- **password** – 连接在登录到服务器时使用的口令。
- **servername** – 定义客户端程序连接到的 Adaptive Server 名称。如果未指定此参数，则 DSQUERY 环境变量会定义 Adaptive Server 名称。

```
sybpydb.connect(user='name', password='password string',
servername='ase servername')
```

模块接口常量

用于扩展模块接口的常量。

apilevel

用以声明所支持的 DB API 级别的字符串常量。缺省值为 2.0。

```
class sybpydb.apilevel
```

paramstyle

用以声明接口需要的参数标记格式设置类型的字符串常量。此常量的值为 **qmark**。接口需要问号样式参数格式设置，例如：

```
'...WHERE name=?'
```

```
class sybpydb.paramstyle
```

threadsafety

用以声明接口支持的线程安全性级别的整数常量。此模块的 **threadsafety** 常量值为 1，这表示此模块是可以共享的，但连接不是。

```
class sybpydb.threadsafety
```

连接对象方法

用于连接对象的 API。

close()

关闭与服务器的连接。连接在调用之后不可用，如果尝试任何操作，则会引发异常。这同样适用于尝试访问连接的游标对象。

```
connection.close()
```

commit()

执行命令 **commit**。

```
connection.commit()
```

rollback()

执行命令 **rollback**。

```
connection.rollback()
```

cursor()

此方法构建使用此连接的新游标对象。

```
connection.cursor()
```

messages()

这是一个 Python 列表对象，模块会将模块为此连接所接收的所有消息的元组（异常类和异常对象）都附加到该列表对象。同一连接对象的任意游标的错误会附加到该游标的连接对象的消息属性。

```
connection.messages()
```

用法示例：

```
try:
    cur.execute("select ...")
except sybpydb.Error:
    for err in cur.connection.messages:
        print("Exception %s, Value %s", % err[0], err[1])
```

游标对象方法

用于游标对象的 API。

close

```
cursor.close()
```

callproc

调用具有给定名称的数据库存储过程。读取所有结果集和行后，使用 **proc_status** 属性检查存储过程的状态结果。

```
cursor.callproc()
```

execute

准备并执行查询。

```
cursor.execute()
```

executemany

准备数据库操作，并针对序列 **seq_of_parameters** 中找到的所有参数序列执行该操作。

```
cursor.executemany()
```

fetch

读取查询结果集的下一行，返回单个序列，或者如果没有更多数据可用，则返回“无”(None)。

```
cursor.fetch()
```

fetchmany

读取查询结果的下一个行集，返回一组序列，例如，一个元组列表。如果没有行可用，则返回空序列。

```
cursor.fetchmany()
```

fetchall

读取查询结果的所有（剩余）行，将其作为一组序列返回。

```
cursor.fetchall()
```

description

用以描述列信息的只读属性。

```
cursor.description()
```

nextset

强制游标跳到下一个可用集合，放弃当前集合中的所有剩余行。

```
cursor.nextset()
```

arraysize

此读/写属性指定 **fetchmany()** 一次读取的行数。它的缺省值为 1，表示一次只读取一行。

```
cursor.arraysize()
```

proc_status

存储过程的返回状态结果。可在读取所有结果集和行之后使用此成员检查存储过程的状态。

```
cursor.proc_status
```

警告和错误消息

所有错误和警告信息都可通过异常和子类找到。

警告

针对警告引发的异常。Python `StandardError` 异常的子类。

```
sybpydb.Warning
```

错误

此异常是所有其它由 sybpydb 定义的异常的基类。**Error** 是 Python `StandardError` 异常的子类。

```
sybpydb.Error
```

InterfaceError

针对与数据库接口（而不是数据库自身）有关的错误引发的异常。它是 `Error` 的子类。

```
sybpydb.InterfaceError
```

DatabaseError

针对与数据库有关的错误引发的异常。它是 `Error` 的子类。

```
sybpydb.DatabaseError
```

DataError

针对与所处理数据的问题有关的错误引发的异常。它是 `DatabaseError` 的子类。

```
sybpydb.DataError
```

OperationalError

针对与数据库的操作问题有关但未必受程序员控制的错误引发的异常。它是 `DatabaseError` 的子类。

```
sybpydb.OperationalError
```

IntegrityError

当数据库的关系完整性受影响时引发的异常。它是 `DatabaseError` 的子类。

```
sybpydb.IntegrityError
```

InternalError

当数据库遇到内部错误时引发的异常。它是 DatabaseError 的子类。

```
sybpydb.InternalError
```

ProgrammingError

针对编程错误引发的异常。它是 DatabaseError 的子类。

```
sybpydb.ProgrammingError
```

NotSupportedError

在使用了不支持的方法或数据库 API 时引发的异常。它是 DatabaseError 的子类。

```
sybpydb.NotSupportedError
```

BulkCursor 对象构造方法

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

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

用法

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

close()

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

用法

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

copy()

BulkCursor 对象的 **copy()** 方法可初始化批量操作。

该方法接受以下参数：

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

用法

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

```
blk = conn.blkcursor()
blk.copy("mytable", direction="out")
```

done()

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

用法

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

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。
- **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。
- **扩展或模块** – 可以通过用 Python 编写的模块来扩展 Python 语言。
- **Python** – 是一种解释型的、多用途的高级别编程语言。有关详细信息，请转到 http://www.python.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