



脚本化语言

用于 Python 的 Adaptive Server® Enterprise 扩展模块 15.7

文档 ID: DC01821-01-1570-02

最后修订日期: 2012 年 6 月

版权所有 © 2012 Sybase, Inc. 保留所有权利。

除非新版本或技术声明中另有说明, 否则本出版物适用于 Sybase 软件及所有后续版本。本文档中的信息如有更改, 恕不另行通知。本出版物中描述的软件按许可证协议提供, 其使用或复制必须符合协议条款。

仅在定期安排的软件发布日期提供升级。未经 Sybase, Inc. 事先书面许可, 本书的任何部分不得以任何形式、任何手段(电子的、机械的、手动、光学的或其它手段)进行复制、传播或翻译。

可在 <http://www.sybase.com/detail?id=1011207> 上的 Sybase 商标页中查看 Sybase 商标。Sybase 和列出的标记均是 Sybase, Inc. 的商标。® 表示已在美国注册。

SAP 和此处提及的其它 SAP 产品与服务及其各自的徽标是 SAP AG 在德国和世界各地其它几个国家/地区的商标或注册商标。

Java 和所有基于 Java 的标记都是 Oracle 和/或其在美国和其它国家/地区的附属机构的商标或注册商标。

Unicode 和 Unicode 徽标是 Unicode, Inc. 的注册商标。

本书中提到的所有其它公司和产品名均可能是与之相关的相应公司的商标。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

目录

- 用于 Python 的 Adaptive Server Enterprise 扩展模块1
 - 安装用于 Python 的扩展模块1
 - 用于 Python 的扩展模块的配置概述2
 - 开发 Python 应用程序3
 - 装载用于 Python 的扩展模块3
 - 使用 Python 建立和关闭与 Adaptive Server 的连接3
 - 使用 Python 访问和更新数据4
 - 将输入和输出参数传递到存储过程5
 - 计算行处理6
 - 动态语句和存储过程的参数支持6
 - 用于 Python 的扩展模块的 API 参考7
- 词汇表11
- 索引13

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>。

用于 Python 的扩展模块的数据流

下图描述数据如何在必需的组件之间移动。



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。
- DBCAPI – 是一个函数库，充当此扩展模块和 CT-Library 之间的中间转换层。
- 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。

版本要求

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

- Adaptive Server Enterprise – 15.7 版或更高版本。
- Python 安装 – 2.6 版内置线程化模式。
- Open Client SDK – 15.7 版或更高版本。

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 Software Developers Kit and Open Server Installation Guide（《软件开发工具包和 Open Server 安装指南》）。

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。

配置任务

配置任务包括：

- Python 模块搜索路径
- 目标服务器的名称和地址
- 安全性和目录服务
- 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（这些是安装不同版本 Adaptive Server Python 扩展模块的缺省目录）之一：

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

目标服务器名称和地址

目标服务器的名称是从以下源之一中获得的（按以下顺序）：

- 客户端应用程序（可以在调用 `sybpydb.connect()` 时提供服务器名）。

- DSQUERY 环境变量（如果应用程序不指定目标服务器）。
- 如果未设置 DSQUERY，则缺省名称是 SYBASE。

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 and Open Server Configuration Guide`（《Open Client 和 Open Server 配置指南》）。

安全性和目录服务

若要更改目录驱动程序或安全性驱动程序，请配置 `libtcl.cfg`。

- [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 and Open Server Configuration Guide`（《Open Client 和 Open Server 配置指南》）。

运行时配置

使用运行时配置文件 `ocs.cfg` 设置：

- 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“Using the Open Client and Open Server Runtime Configuration File”（使用 Open Client 和 Open Server 运行时配置文件）。

开发 Python 应用程序

使用 `sybpydb` 界面编写 Python 脚本。

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。

若要从 Python 脚本中使用 `sybpydb` 模块，必须先装载它，即在 `python` 脚本的顶部包含此行：

```
import sybpydb
```

使用 Python 建立和关闭与 Adaptive Server 的连接

打开和关闭与 Adaptive Server 的连接。

`connect` 方法用于打开数据库连接并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。如果不指定此参数，则 DSQUERY 环境变量会定义 Adaptive Server 名称。

connect 方法用于返回一个连接对象，该对象随后用于关闭连接。此示例显示应用程序如何打开和关闭连接：

```
import sybpydb

# Create a connection
conn = sybpydb.connect(user='john', password='sybase')

# Close the connection.
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。

游标对象可用于访问方法来准备和执行查询以及从结果集中读取行。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()** 会抛出例外，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 **OutParam** 构造方法以用于指定输出参数。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` 扩展界面 API。

模块界面方法

用于模块界面的 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')
```

模块界面常量

用于扩展模块界面的常量。

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

用以声明界面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`。界面需要问号样式参数格式设置，例如：

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

```
class sybpydb.paramstyle
```

threadsafety

用以声明界面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存储数据库过程。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

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

```
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。

Warning

针对警告引发的异常。Python StandardError 异常的子类。

```
sybpydb.Warning
```

Error

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
```

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。
- **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。
- **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

A

安装选项 1

B

版本要求 1

C

参数

- date 6
- datetime 6
- decimal 6
- float 6
- money 6
- time 6

参数支持

- 存储过程 6
- 动态语句 6

词汇表 11

存储过程 5

J

计算行处理 6

警告和错误消息 9

K

扩展模块

- Python API 参考 7
- 界面常量 7
- 界面方法 7
- 警告和错误消息 9
- 连接对象方法 7
- 游标对象方法 8

装载 3

L

连接

- 关闭 3
- 建立 3

P

配置

- 安全性和目录服务 2
- 目标服务器 2
- 任务 2
- 搜索路径 2
- 运行时 2

S

sybpydb 接口 3

输出参数 5

输入参数 5

数据

- 访问 4
- 更新 4

数据流框图 1

Y

游标对象 4

Z

组件

- 必需 1
- 说明 1

