SYBASE®
An SAP® Company

プログラマーズガイド

Python 用 Adaptive Server® Enterprise 拡張モジュール バージョン 15.7 SP100

ドキュメント ID： DC01820-01-1570100-01
改訂： 2013 年 5 月

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

目次

Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用の拡張モジュール、sybpydb は、Adaptive Server® Enterprise データベースに対してクエリを実行するのに使用する Sybase® 固有の Python インタフェースです。

この拡張モジュールは、拡張機能の付いた Python データベース API 仕様バージョン 2.0 を実装します。API 仕様の詳細については、http://www.python.org/dev/peps/pep-0249 を参照してください。

必要なコンポーネント

Python プログラミング言語を使用して Adaptive Server データベースにアクセスするには、以下のコンポーネントが必要です。

- sybpydb - Python スクリプト言語用の拡張モジュール
- Open Client SDK - データソース、情報アプリケーションまたはシステムサービスにアクセスするためのアプリケーション開発ツール

バージョン要件

Python 用 Adaptive Server Enterprise 拡張モジュールには 3 つのバージョン要件があります。

- Adaptive Server Enterprise - バージョン 15.7 以上
- Python インストール - バージョン 2.6、2.7、または 3.1 組み込みスレッドモード
- Open Client SDK - バージョン 15.7 以上

注意： プラットフォームのサポートについては、使用しているプラットフォームの『Software Developer's Kit/Open Server インストールガイド』を参照してください。

Python 用拡張モジュールのインストール

Python 用の拡張モジュールは Sybase インストーラでインストールできるコンポーネントです。

インストールタイプとしてカスタムを選択した場合、Python 用の拡張モジュールはオプションでインストールします。 インストールタイプとして標準またはフル

を選択した場合、この拡張モジュールはデフォルトでインストールされます。 インストールと設定の詳細な手順については、使用しているプラットフォームの『Software Developer's Kit/Open Server インストールガイド』を参照してください。

Python 用拡張モジュールの設定の概要

Python アプリケーションが接続を行い、コマンドを実行できるようにするための基本的な設定作業を完了します。

- Python モジュールの検索パス
- ターゲットサーバの名前とアドレス
- セキュリティサービスとディレクトリサービス
- `ocs.cfg` ファイルでのランタイム設定

Python モジュールの検索パス

Python は、Python 変数 `sys.path` で指定されたディレクトリのリスト内で、インポートされたモジュールを検索します。

この変数はアプリケーションを含むディレクトリと環境変数 PYTHONPATH で指定されるディレクトリのリストから初期化されます。PYTHONPATH はシェル変数 PATH (つまりディレクトリ名のリスト) と同じ構文を使用します。
PYTHONPATH が設定されていない、またはそのファイルが見つからない場合は、インストールに依存するデフォルトのパス内で検索を続行します。

Python 用 Adaptive Server Enterprise 拡張モジュールをアプリケーションで使用するには、PYTHONPATH または Python 変数 `sys.path` を次のディレクトリパス (各バージョンの Adaptive Server Python 拡張モジュールがインストールされるデフォルトディレクトリ) のいずれか 1 つに設定する必要があります。

プラットフォーム	デフォルトインストールパス	Python のバージョン
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

プラットフォーム	デフォルトインストールパス	Python のバージョン
その他のプラットフォーム	`$SYBASE/$SYBASE_OCS/python/python26_64r/lib`	2.6、2.7
	`$SYBASE/$SYBASE_OCS/python/python31_64r/lib`	3.1

ターゲットサーバの名前とアドレス

Python アプリケーションが Adaptive Server に接続する際、ターゲットサーバの名前が次のソースのいずれかから取得されます。

1. サーバ名 (**connect** メソッドで指定されている場合)
2. DSQUERY 環境変数 (アプリケーションの **connect** メソッドでターゲットサーバが指定されていない場合)
3. SYBASE 環境変数 (DSQUERY が設定されていない場合)

ターゲットサーバのアドレスは、ディレクトリサービスまたはプラットフォームに依存する interfaces ファイルから取得されます。 interfaces ファイルまたは LDAP ディレクトリサービスにサーバエントリを作成します。 使用しているプラットフォームの『Open Client/Server 設定ガイド』を参照してください。

セキュリティサービスとディレクトリサービス

ディレクトリドライバまたはセキュリティドライバは、`libtcl.cfg` ファイル内の次のセクションで設定する必要があります。

- [DIRECTORY] セクション (ディレクトリドライバ)
- [SECURITY] セクション (セキュリティドライバ)

使用しているプラットフォームの『Open Client/Server 設定ガイド』を参照してください。

ランタイム設定

ランタイム設定ファイル `ocs.cfg` を使用して次の値を設定します。

- プロパティ値
- サーバオプション値
- サーバ機能
- デバッグオプション

ファイル構文と、ファイルで設定できるプロパティについては、『Open Client Client-Library/C リファレンスマニュアル』の「Open Client/Server ランタイム設定ファイルの使用方法」を参照してください。

Python アプリケーションの開発

sybpydb モジュールを使用して Python アプリケーションを開発する方法の概要を説明します。

スレッドセーフ

スレッドはモジュールを共有できます。 ただし、接続、その属性、および接続を使用して作成されたオブジェクトをロックメカニズムなしで共有することはできません。

接続から作成されるオブジェクトの例として、カーソルがあります。 複数のスレッド間で 1 つの接続を使用するには、その接続 (その属性および作成されたオブジェクト) を、リソースロックを実現するセマフォを使ってラップする必要があります。

パラメータスタイル

モジュールは、疑問符スタイルのパラメータマーカフォーマットをサポートします。

Python 用拡張モジュールのロード

Python 用拡張モジュールをロードするには、インポート文を使用します。
Python スクリプトで sybpydb モジュールを使用するには、まず、次の方法でインポートしてロードする必要があります。

```
import sybpydb
```

DSN スタイル接続文字列プロパティを使用した Adaptive Server との接続の確立と終了

connect メソッドを使用してデータベース接続を開きます。 このメソッドは、DSN スタイル接続プロパティをサポートします。

このメソッドは、次のキーワード引数を受け入れます。

- **user** - サーバへのログイン時に接続が使用するユーザログイン名。
- **password** - サーバへのログイン時に接続が使用するパスワード。

- **servername** - クライアントプログラムが接続する Adaptive Server の名前。 **servername** を指定しない場合、DSQUERY 環境変数で Adaptive Server 名前を定義します。
- **dsn** - データソース名。 データソース名は、セミコロンで区切られた名前=値の文字列です。
  - 名前 - 等号 (=) またはセミコロン (;) で区切ることができる値。大文字と小文字が区別されます。 1 つの属性に複数の同義語がある場合があります。 たとえば、**server** と **servername** は同じ属性を指します。
  - 等号 (=) - 名前に割り当てる値の開始を示します。 等号がない場合、名前は true 値のブール型とみなされます。
  - 値 - セミコロン (;) で終了する文字列。 値内にセミコロンまたは別の円記号 (¥) がある場合は、円記号を使用します。 値は、ブール型、整数型、または文字列型で指定できます。 ブール型の有効な値は、true、false、on、off、1、0 です。

    注意： 値がないブール名がある場合、ブール型は true に設定する必要があります。

  次に例を示します。

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

有効な属性名と属性値

dsn キーワード引数に対する有効な属性名とその値を以下の表に示します。

名前	説明	値
ANSINull	SQL の等号 (=) または不等号 (!=) の比較で NULL 値のオペランドの評価が ANSI 標準に準拠しているかどうかを判別する。 値が true の場合、Adaptive Server は、*= NULL* と *is NULL* は同義ではないとする ANSI の動作を適用する。 標準の Transact-SQL では、*= NULL* と *is NULL* は同義とみなされる。 このオプションは、*<> NULL* と *is not NULL* の動作にも同様に作用する。	ブール値。 デフォルトは false。
BulkLogin	接続でバルクコピーオペレーションを実行できるかどうかを判別する。	ブール値。 デフォルトは false。

名前	説明	値
ChainXacts	true の場合、Adaptive Server は連鎖トランザクション動作を使用する。つまり、各サーバコマンドが個別のトランザクションとみなされる。 Adaptive Server は次の各文の前に、begin transaction を暗黙的に実行する。 **delete**、**fetch**、**insert**、**open**、**select** および **update**。 ただし、トランザクションを明示的に終了またはロールバックする必要がある。 false の場合、アプリケーションは begin transaction 文を commit 文または rollback 文と対で明示的に指定する必要がある。	ブール値。 デフォルトは false。
Charset	この接続で使用する **charset** を指定する。	文字列値。
Confidentiality	接続でデータの暗号化サービスを実行するかどうか。	ブール値。 デフォルトは false。
Credential-Delegation	ユーザの委任クレデンシャルを使用してサーバを他のサーバに接続させるかどうかを判別する。	ブール値。 デフォルトは false。
DetectReplay	接続のセキュリティメカニズムがリプレイされた転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
DetectOutOfSequence	接続のセキュリティメカニズムが不正なシーケンスの転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
Integrity	接続のセキュリティメカニズムがデータ整合性チェックを実行するかどうかを判別する。	ブール値。 デフォルトは false。
Interfaces	interfaces ファイルのパスと名前。	文字列値。
Keytab	接続のセキュリティメカニズムが *username* 値と一緒に使用されるセキュリティキーを読み込むファイルのパスと名前。	文字列値。 デフォルトは NULL。つまり、接続前にユーザがクレデンシャルを確立しておく必要がある。
Locale	メッセージ、データ型変換、日時フォーマットで使用する言語と文字セットを判別する。	文字列値。
Language	メッセージ、データ型変換、日時フォーマットで使用する言語セットを判別する。	文字列値。

名前	説明	値
LoginTimeout	ログインタイムアウト値を指定する。	整数値。
MaxConnect	あるコンテキストで同時にオープン可能な接続の最大数を指定する。	整数値。 デフォルト値は 25。 負の値および 0 は使用できない。
MutualAuthentication	サーバがクライアントに対して自身を認証する必要があるかどうかを判別する。	ブール値。 デフォルトは false。
NetworkAuthentication	接続のセキュリティメカニズムがネットワークベースのユーザ認証を実行するかどうかを判別する。	ブール値。 デフォルトは false。
PacketSize	TDS パケットサイズを指定する。	整数値。
Password	サーバへのログインに使用するパスワードを指定する。	文字列値。
PasswordEncryption	接続が非対称のパスワード暗号化を使用するかどうかを判別する。	ブール値。 デフォルトは false。
SecurityMechanism	接続のためのセキュリティサービスを実行するネットワークセキュリティメカニズムの名前を指定する。	文字列値。 デフォルト値はセキュリティドライバ設定によって異なる。
Server **Servername**	接続先のサーバの名前を指定する。	文字列値。
ServerPrincipalName	接続がオープンされるサーバのネットワークセキュリティプリンシパル名を指定する。	文字列値。 デフォルトは NULL。 このとき接続は、サーバのプリンシパル名がその *ServerName* 値と同じであるとみなす。
Keepalive	KEEPALIVE オプションを使用するかどうかを判別する。	ブール値。 デフォルトは true。
Timeout	接続タイムアウト値を指定する。	整数値。

名前	説明	値
UID **User** **Username**	サーバへのログインに使用する名前を指定する。	文字列値。

バルクコピーのサポート

バルクコピーオペレーションのサポートは Python DBAPI の拡張機能です。 この機能を使用すると、ローのバルクコピーを実行できます。

バルクコピーオペレーションの例:

```
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()
```

Python を使用したデータへのアクセスと更新

接続が確立された後、cursor オブジェクトを使用してフェッチ操作のコンテキストを管理します。

cursor オブジェクトを使用すると、クエリを準備して実行し、結果セットからローをフェッチするメソッドにアクセスできるようになります。cursor オブジェクトは、**cursor** メソッドを使用して connection オブジェクトから取得されます。次の例は、アプリケーションがデータにアクセスして更新する方法を示します。

```
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()
```

ストアドプロシージャへの入力パラメータと出力パラメータの転送

15.7 ESD#3 から、Python 用 Adaptive Server Enterprise 拡張モジュールは、ストアドプロシージャへの入力パラメータと出力パラメータの転送をサポートしています。

Cursor オブジェクトの **callproc()** メソッドを使用して、ストアドプロシージャを呼び出します。ストアドプロシージャの実行中にエラーが発生した場合、**callproc()** は例外をスローするので、**proc_status** 属性を使用してこのステータス値を取得できます。このサポートは、Python DBAPI 仕様の拡張機能になります。

これは、ロー結果が複数になるサンプル Python アプリケーションです。

```
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)
```

拡張モジュールは、出力パラメータを指定するのに OutParam コンストラクタを用意しています。このサポートは、Python DBAPI 仕様の拡張機能です。**callproc()** メソッドは、このメソッドに渡されたすべてのパラメータのリストを返します。出力パラメータがあり、ストアドプロシージャから生成された結果セットがない場合、**callproc()** が終了するとすぐに、変更された出力値がリストに格納されます。ただし、結果セットがある場合は、ストアドプロシージャからの結果セットのすべてが **fetch*()** メソッドを使用して取得され、**nextset()** の呼び出しが実行され、結果セットがもうないか確認されるまで、変更された出力値はリストに格納されま

せん。**nextset()** メソッドは、予想される結果セットが 1 つのみの場合でも呼び出す必要があります。

これは、出力パラメータのあるサンプル Python アプリケーションです。

```
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()
```

さまざまな出力パラメータタイプのその他の例が、サンプルプログラム `callproc.py` で紹介されています。

ロー処理の計算

15.7 ESD#3 から、Python 用 Adaptive Server Enterprise 拡張モジュールは、ロー処理の計算をサポートしています。

この例はサンプルプログラム `compute.py` で紹介されています。

動的文とストアドプロシージャのパラメータサポート

15.7 ESD#4 から、Python 用 Adaptive Server Enterprise 拡張モジュールは、動的文とストアドプロシージャ用パラメータとして decimal、money、および LOB をサポートしています。

decimal 型パラメータと money 型パラメータ

次に、ストアドプロシージャのパラメータとしての decimal と money の使用例を示します。

```
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])
```

ストアドプロシージャに対してのdate パラメータ、time パラメータ、datetime パラメータ、およびfloat パラメータのサポート

Python 用 Adaptive Server Enterprise 拡張モジュールは、ストアドプロシージャのdate パラメータ、time パラメータ、datetime パラメータ、および float パラメータをサポートします。

date、time、datetime、float、integer を含むさまざまなデータ型のパラメータでストアドプロシージャを呼び出す方法については、`callproc.py` サンプルを参照してください。このサンプルでは、出力パラメータの処理方法も示しています。

Python API リファレンス用拡張モジュール

sybpydb 拡張インタフェース API。

モジュールインタフェースメソッド

モジュールインタフェースに使用する API。

connect

データベースへの接続を表す Connection オブジェクトを構成します。このメソッドは以下のキーワード引数を受け入れます。

- **user** - サーバへのログイン時に接続が使用するユーザログイン名。

- **password** - サーバへのログイン時に接続が使用するパスワード。
- **servername** - クライアントプログラムが接続する Adaptive Server の名前の定義。この引数が指定されない場合、DSQUERY 環境変数が Adaptive Server の名前を定義します。

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

モジュールインタフェース定数

拡張モジュールインタフェースに使用する定数。

apilevel

サポートされている DB API レベルを説明する文字列定数。デフォルト値は 2.0 です。

```
class sybpydb.apilevel
```

paramstyle

インタフェースが想定するフォーマットのパラメータマーカのタイプを説明する文字列定数。この定数の値は、**qmark** です。インタフェースは、次のような疑問符スタイルのパラメータフォーマットを想定しています。

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

```
class sybpydb.paramstyle
```

threadsafety

インタフェースがサポートしているスレッドセーフのレベルを説明する整数定数。このモジュールの **threadsafety** 定数値は 1 で、モジュールは共有はできるが、接続はできないことを示します。

```
class sybpydb.threadsafety
```

Connection オブジェクトメソッド

Connection オブジェクトに使用する API。

close()

サーバへの接続をクローズする。呼び出しの後は connection を使用できません。何らかの操作を行うと例外が発生します。cursor オブジェクトが connection にアクセスしようとする場合も同様です。

```
connection.close()
```

commit()

コマンド **commit** を実行します。

```
connection.commit()
```

rollback()

コマンド **rollback** を実行します。

```
connection.rollback()
```

cursor()

このメソッドは connection オブジェクトを使用して新しい cursor オブジェクトを構成します。

```
connection.cursor()
```

messages()

これは、この Connection オブジェクトに対してモジュールが受け取るすべてのメッセージに、モジュールがタプル (例外クラスと例外オブジェクト) を追加する Python リストオブジェクトです。同じ Connection オブジェクトから取得されるカーソル上のエラーが、そのカーソルの Connection オブジェクトのメッセージ属性に追加されます。

```
connection.messages()
```

使用例:

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

Cursor オブジェクトメソッド

cursor オブジェクトに使用する API。

close

```
cursor.close()
```

callproc

指定した名前のストアドデータベースプロシージャを呼び出します。結果セットとローをすべてフェッチした後、**proc_status** 属性を使用してストアドプロシージャのステータス結果をチェックします。

```
cursor.callproc()
```

execute

クエリを準備し、実行します。

```
cursor.execute()
```

executemany

データベースの操作の準備をし、シーケンス **seq_of_parameters** で見つかったすべてのパラメータシーケンスに対して実行します。

```
cursor.executemany()
```

fetch

クエリ結果のセットから次のローをフェッチし、シーケンスを 1 つ返すか、またはデータがない場合は None を返します。

```
cursor.fetch()
```

fetchmany

クエリ結果のローの次のセットをフェッチし、たとえば、タプルのリストなどのシーケンスを 1 つ返します。使用できるローがない場合は空のシーケンスを返します。

```
cursor.fetchmany()
```

fetchall

クエリ結果の残りのすべてのローをフェッチし、シーケンスを順番にして返します。

```
cursor.fetchall()
```

description

カラム情報を説明する読み込み専用の属性。

```
cursor.description()
```

nextset

使用できる次のセットにカーソルをスキップさせ、現在のセットから残りのローを破棄します。

```
cursor.nextset()
```

arraysize

この読み込み/書き込み属性で、**fetchmany()** で一度にフェッチするローの数を指定します。デフォルトは 1 で、一度に 1 つのローをフェッチします。

```
cursor.arraysize()
```

proc_status

ストアドプロシージャはステータス結果を返します。結果セットとローをすべてフェッチした後、このメンバを使用してストアドプロシージャのステータスをチェックします。

```
cursor.proc_status
```

警告メッセージとエラーメッセージ

すべてのエラーと警告の情報は、例外とサブクラスを通して取得できます。

警告

警告に対して発生する例外。Python `StandardError` 例外のサブクラス。

```
sybpydb.Warning
```

エラー

sybpydb により定義されるその他のすべての例外の基本クラスである例外。**Error** は Python `StandardError` 例外のサブクラスです。

```
sybpydb.Error
```

InterfaceError

データベースそのものではなく、データベースインタフェースに関連するエラーに対して発生する例外。`Error` のサブクラスです。

```
sybpydb.InterfaceError
```

DatabaseError

データベースに関連するエラーに対して発生する例外。`Error` のサブクラスです。

```
sybpydb.DatabaseError
```

DataError

処理されるデータの問題に関連するエラーに対して発生する例外。`DatabaseError` のサブクラスです。

```
sybpydb.DataError
```

OperationalError

データベースの操作の問題に関連するエラーに対して発生する例外。ただし、必ずしもプログラマがコントロールできる問題とは限りません。`DatabaseError` のサブクラスです。

```
sybpydb.OperationalError
```

IntegrityError

データベースの関係整合性が影響される場合に発生する例外。`DatabaseError` のサブクラスです。

```
sybpydb.IntegrityError
```

InternalError

データベースが内部エラーを起こした場合に発生する例外。DatabaseError のサブクラスです。

```
sybpydb.InternalError
```

ProgrammingError

プログラミングエラーに対して発生する例外。DatabaseError のサブクラスです。

```
sybpydb.ProgrammingError
```

NotSupportedError

サポートされていないメソッドやデータベース API が使用された場合に発生する例外。DatabaseError のサブクラスです。

```
sybpydb.NotSupportedError
```

BulkCursor オブジェクトコンストラクタ

データベースとの接続を確立する接続オブジェクトを提供する Python 拡張モジュールです。 接続オブジェクトには、バルクオペレーションのコンテキストを管理する新しい BulkCursor オブジェクトを作成するためのメソッドが含まれます。

バルクオペレーションで使用される接続をマーク付けするプロパティで構築された接続オブジェクトからのみ BulkCursor オブジェクトを作成できます。

使用法

```
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 オブジェクトの **close()** メソッドはバルクオペレーションを終了します。 このメソッドを呼び出すと、バルクカーソルオブジェクトは使用できなくなります。 **close()** に引数はありません。

使用法

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

copy()

BulkCursor オブジェクトの **copy()** メソッドはバルクオペレーションを開始します。

このメソッドは、次の引数を受け入れます。

- **tablename** - バルクオペレーション用のテーブルの名前を指定する文字列
- **direction** - *in* と *out* の値を持つキーワード引数

使用法

```
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` オブジェクトの **done()** メソッドはバルクオペレーションの完了をマーク付けします。 別のオペレーションを開始するには、**copy()** メソッドを呼び出します。

使用法

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

用語解説

スクリプト言語に特有の用語集

- **Client-Library –** Open Client の一部で、クライアントアプリケーションを記述するためのルーチンの集まり。Client-Library は、Sybase 製品ラインのカーソルや他の高度な機能を取り込むように設計されています。
- **CS-Library –** Client-Library と Server-Library のアプリケーションの両方で役立つユーティリティルーチンの集まり。Open Client および Open Server の両方に含まれています。
- **CT-Library –** (CT-Lib API) は Open Client スイートの一部であり、スクリプトアプリケーションで Adaptive Server に接続するために必要です。
- **拡張またはモジュール –** Python 言語は Python で記述されたモジュールで拡張できます。
- **Python –** インタプリタ型の、汎用で高レベルのプログラミング言語。詳細については、http://www.python.org を参照してください。
- **スレッド (thread) –** Open Server アプリケーションからライブラリコードまでの実行のパス。また、スタック領域、ステータス情報およびイベントハンドラに対応するパス。
- **Transact-SQL –** データベース言語 SQL の機能拡張バージョン。アプリケーションは、Transact-SQL を使用して、Adaptive Server Enterprise と通信できます。

索引