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

ドキュメント ID： DC01820-01-1570-02
改訂： 2012 年 6 月

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

目次

Python 用 Adaptive Server Enterprise 拡張モジュール 1
Python 用拡張モジュールのインストール 2
Python 用拡張モジュールの設定の概要 2
Python アプリケーションの開発 4
Python 用拡張モジュールのロード 4
Python を使用した Adaptive Server との接続の確立と終了 4
Python を使用したデータへのアクセスと更新 5
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 5
ロー処理の計算 7
動的文とストアド・プロシージャのパラメータ・サポート 7
Python API リファレンス用拡張モジュール 8
用語解説 13
索引 15

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。

Python データ・フロー用の拡張モジュール
次の図は必要なコンポーネント中でのデータの流れ方を示します。

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。
- DBCAPI – 拡張モジュールと CT-Library 間の中間の変換レイヤとして動作する関数ライブラリ。
- CT-Library – (CT-Lib API) は Open Client スイートの一部です。CT-Library は Adaptive Server にコマンドを送信し、結果を処理します。

バージョン要件
プラットフォームのサポートの詳細は、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。

- 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 組み込みスレッド・モード
- Open Client SDK – バージョン 15.7 以上

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。

Python 用の拡張モジュールは、インストール・タイプとしてカスタムを選択した場合はオプションでインストールします。選択したインストール・タイプが標準またはフルの場合は、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完全な手順については、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。

設定作業
設定作業には以下のものがあります。

- 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` で与えられる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` のいずれかを以下のディレクトリ・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 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
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
ターゲット・サーバの名前は、以下のソースのいずれかからこの順序で取得されます。

- **sybpydb.connect()** に対する呼び出しでサーバ名を提供できるクライアント・アプリケーション。
- アプリケーションがターゲット・サーバを指定していない場合、DSQUERY 環境変数。
- DSQUERY が設定されていない場合は、デフォルト名の SYBASE。

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。詳細については、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・ドライバを変更するには、`libtcl.cfg` を設定します。

- ディレクトリ・ドライバは [DIRECTORY] セクションを参照。
- セキュリティ・ドライバは [SECURITY] セクションを参照。

詳細については、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、以下を設定します。

- 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「ランタイム設定ファイルの使い方」を参照してください。

Python アプリケーションの開発

sybpydb インタフェースを使用して Python スクリプトを記述します。

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、以下の行をスクリプトの最上部にインクルードしてロードする必要があります。

```
import sybpydb
```

Python を使用した Adaptive Server との接続の確立と終了

Adaptive Server への接続のオープンとクローズ。

connect メソッドはデータベース接続を開き、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。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。

connect メソッドは connection オブジェクトを返し、このオブジェクトを使用して接続を終了します。次の例は、アプリケーションが接続を開く方法と終了する方法を示します。

```
import sybpydb

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

```
    # Close the connection.
    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

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.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。

Warning

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

```
sybpydb.Warning
```

Error

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らない。`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
```

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。
- **DBD –** ベンダ固有のデータベース・ドライバで、DBI データベース API 呼び出しをターゲット・データベース SDK が理解できる形式に変換します。
- **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

C

cursor オブジェクト 5

S

sybpydb インタフェース 4

い

インストール・オプション 2

か

拡張モジュール
- connection オブジェクト・メソッド 9
- cursor オブジェクト・メソッド 10
- Python API リファレンス 8
- 警告メッセージとエラー・メッセージ 11

こ

コンポーネント
- 説明 1
- 必要な 1

し

出力パラメータ 5

す

ストアド・プロシージャ 5

て

データ
- アクセス 5
- 更新 5

データ・フロー図 1

に

入力パラメータ 5

は

バージョン要件 1
パラメータ
- date 7
- datetime 7
- decimal 7
- float 7
- money 7
- time 7

パラメータのサポート
- ストアド・プロシージャ 7
- 動的文 7

よ

用語解説 13

ろ

ロー処理の計算 7