python操作mysql数据库

Python 标准数据库接口为 Python DB-API，Python DB-API为开发人员提供了数据库应用编程接口。

Python 数据库接口支持非常多的数据库，你可以选择适合你项目的数据库：

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

你可以访问Python数据库接口及API查看详细的支持数据库列表。

不同的数据库你需要下载不同的DB API模块，例如你需要访问Oracle数据库和Mysql数据，你需要下载Oracle和MySQL数据库模块。

DB-API 是一个规范. 它定义了一系列必须的对象和数据库存取方式, 以便为各种各样的底层数据库系统和多种多样的数据库接口程序提供一致的访问接口 。

Python的DB-API，为大多数的数据库实现了接口，使用它连接各数据库后，就可以用相同的方式操作各数据库。

Python DB-API使用流程：

- 引入 API 模块。
- 获取与数据库的连接。
- 执行SQL语句和存储过程。
- 关闭数据库连接。

什么是MySQLdb?

MySQLdb 是用于Python链接Mysql数据库的接口，它实现了 Python 数据库 API 规范 V2.0，基于 MySQL C API 上建立的。

如何安装MySQLdb?

安装MySQLdb，请访问 http://sourceforge.net/projects/mysql-python ，(Linux平台可以访问：https://pypi.python.org/pypi/MySQL-python)从这里可选择适合您的平台的安装包，分为预编译的二进制文件和源代码安装包。
如果您选择二进制文件发行版本的话，安装过程基本安装提示即可完成。如果从源代码进行安装的话，则需要切换到MySQLdb发行版本的顶级目录，并键入下列命令:

```
$ gunzip MySQL-python-1.2.2.tar.gz
$ tar -xvf MySQL-python-1.2.2.tar
$ cd MySQL-python-1.2.2
$ python setup.py build
$ python setup.py install
```

下载安装包

MySQL-python

```
[root@localhost ~]# ls
MySQL-python-1.2.5.zip
[root@localhost ~]# unzip MySQL-python-1.2.5.zip
[root@localhost ~]# cd MySQL-python-1.2.5/
[root@localhost MySQL-python-1.2.5]# ls
oc      MANIFEST.in   _mysql_exceptions.py   README.md         setup_posix.py    site.cfg
GPL-2.0  metadata.cfg  MySQL_python.egg-info  setup.cfg         setup_posix.pyc   tests
HISTORY  _mysql.c      PKG-INFO               setup_common.py   setup.py
INSTALL  MySQLdb       pymemcompat.h          setup_common.pyc  setup_windows.py
[root@localhost MySQL-python-1.2.5]# python setup.py build
sh: mysql_config: command not found
Traceback (most recent call last):
  File "setup.py", line 17, in <module>
    metadata, options = get_config()
  File "/root/MySQL-python-1.2.5/setup_posix.py", line 43, in get_config
    libs = mysql_config("libs_r")
  File "/root/MySQL-python-1.2.5/setup_posix.py", line 25, in mysql_config
    raise EnvironmentError("%s not found" % (mysql_config.path,))
EnvironmentError: mysql_config not found
```

缺少**mysql_config**命令

该命令是由mysql-devel或者mariadb-devel安装生成，因此先检查是否安装，在检车site.cfg中mysql_config的路径是否正确

```
[root@localhost MySQL-python-1.2.5]# which mysql_config
/usr/bin/which: no mysql_config in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin)

[root@localhost MySQL-python-1.2.5]# yum install -y mariadb-devel

[root@localhost MySQL-python-1.2.5]# which mysql_config
/usr/bin/mysql_config

[root@localhost MySQL-python-1.2.5]# vim site.cfg
mysql_config=/usr/bin/mysql_config
```

开始编译安装

```
[root@localhost MySQL-python-1.2.5]# python setup.py build
[root@localhost MySQL-python-1.2.5]# python setup.py install
```

加载模块

```
In [2]: import MySQLdb

In [3]: dir(MySQLdb)
Out[3]:
['BINARY',
 'Binary',
 'Connect',
 'Connection',
 'DATE',
 'DATETIME',
 'DBAPISet',
 'DataError',
 'DatabaseError',
 'Date',
 'DateFromTicks',
 'Error',
 'FIELD_TYPE',
 'IntegrityError',
 'InterfaceError',
 'InternalError',
 'MySQLError',
 'NULL',
 'NUMBER',
 'NotSupportedError',
 'OperationalError',
 'ProgrammingError',
 'ROWID',
 'STRING',
 'TIME',
 'TIMESTAMP',
 'Time',
 'TimeFromTicks',
 'Timestamp',
 'TimestampFromTicks',
 'Warning',
 '__all__',
 '__author__',
 '__builtins__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__revision__',
 '__version__',
 '_mysql',
 'apilevel',
 'connect',
 'connection',
 'constants',
 'debug',
 'escape',
```

```
 'escape_dict',
 'escape_sequence',
 'escape_string',
 'get_client_info',
 'paramstyle',
 'release',
 'result',
 'server_end',
 'server_init',
 'string_literal',
 'test_DBAPISet_set_equality',
 'test_DBAPISet_set_equality_membership',
 'test_DBAPISet_set_inequality',
 'test_DBAPISet_set_inequality_membership',
 'thread_safe',
 'threadsafety',
 'times',
 'version_info'
```

数据库连接

连接数据库前，请先确认以下事项：

1. 数据库服务已经启动，并已创建了数据库 test1
2. 在test1数据库中您已经创建了表 db1
3. t1表字段为 id,first_name,last_name,age,sex,income
4. 连接数据库t1使用的用户名为 "root" ，密码为 "uplooking"

```
[root@localhost ~]# systemctl start mariadb
[root@localhost ~]# mysqladmin -uroot password 'uplooking'
[root@localhost ~]# mysql -uroot -puplooking
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 5.5.44-MariaDB-log MariaDB Server

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database db1;
Query OK, 1 row affected (0.01 sec)

MariaDB [(none)]> create table db1.t1 (id int auto_increment primary key,first_name
varchar(50),last_name varchar(50),age int,sex char(1),income float);
Query OK, 0 rows affected (0.09 sec)
```

实例**1**：连接数据库获取数据库版本信息

```
[root@localhost python-mysql]# vim getversion.py
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost","root","uplooking","db1" )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# 使用execute方法执行SQL语句
cursor.execute("SELECT VERSION()")

# 使用 fetchone() 方法获取一条数据库。
data = cursor.fetchone()

print "Database version : %s " % data

# 关闭数据库连接
db.close()


# 执行结果
[root@localhost python-mysql]# python getversion.py
Database version : 5.5.44-MariaDB-log
```

实例**2**：查询**mysql**中的**db1.t1**表的结构（数据字典）信息

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb
import sys

sql=sys.argv[1]

# 打开数据库连接
db = MySQLdb.connect("localhost","root","uplooking","db1" )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# 使用execute方法执行SQL语句
cursor.execute(sql)

# 使用 fetchone() 方法获取一条数据库。
data = cursor.fetchall()

for i in data:
        for j in i:
                sys.stdout.write(str(j)+'\t')
        sys.stdout.write('\n')


# 关闭数据库连接
db.close()
```

脚本带参数执行：

```
[root@localhost python-mysql]# python select.py "desc db1.t1"
id  int(11) NO  PRI None    auto_increment
first_name  varchar(50) YES     None
last_name   varchar(50) YES     None
age int(11) YES     None
sex char(1) YES     None
income  float   YES     None
```

数据库查询操作

- Python查询Mysql使用 fetchone() 方法获取单条数据, 使用fetchall() 方法获取多条数据。
- fetchone(): 该方法获取下一个查询结果集。结果集是一个对象
- fetchall():接收全部的返回结果行.
- rowcount: 这是一个只读属性，并返回执行execute()方法后影响的行数。

实例3：获取mysql数据库的变量值

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb
import sys

key=sys.argv[1]
sql="select @@"+key

# 打开数据库连接
db = MySQLdb.connect("localhost","root","uplooking","db1" )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# 使用execute方法执行SQL语句
cursor.execute(sql)

# 使用 fetchone() 方法获取一条数据库。
data = cursor.fetchall()

print "{} : {}".format(sys.argv[1],data[0])


# 关闭数据库连接
db.close()
```

获取innodb_version\innodb_buffer_pool_size

```
[root@localhost python-mysql]# python Get_variables.py innodb_version
innodb_version : 5.5.43-MariaDB-37.2
[root@localhost python-mysql]# vim Get_variables.py
[root@localhost python-mysql]# python Get_variables.py innodb_buffer_pool_size
innodb_buffer_pool_size : 134217728
```

实例4：创建数据库表

如果数据库连接存在我们可以使用execute()方法来为数据库创建表，如下所示创建表t2：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost","root","uplooking","db1" )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# 如果数据表已经存在使用 execute() 方法删除表。
cursor.execute("DROP TABLE IF EXISTS t2")

# 创建数据表SQL语句
sql = """CREATE TABLE t2 (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""

cursor.execute(sql)

# 关闭数据库连接
db.close()
```

执行该脚本

```
[root@localhost python-mysql]# python createdb.py
createdb.py:13: Warning: Unknown table 't2'
  cursor.execute("DROP TABLE IF EXISTS t2")

[root@localhost python-mysql]# ls
createdb.py  Get_variables.py  getversion.py  select.py

#查看表的结构

[root@localhost python-mysql]# python select.py "desc db1.t2"
FIRST_NAME   char(20)     NO       None
LAST_NAME    char(20)     YES      None
AGE int(11) YES      None
SEX char(1) YES      None
INCOME   float    YES      None
```

实例**5**：数据库插入操作

以下实例使用执行 SQL INSERT 语句向表 db1.t1 插入记录：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost","root","uplooking","db1" )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 插入语句
sql = """INSERT INTO t1(FIRST_NAME,
         LAST_NAME, AGE, SEX, INCOME)
         VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
   # 执行sql语句
   cursor.execute(sql)
   # 提交到数据库执行
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# 关闭数据库连接
db.close()
```

执行该脚本，并查看db1.t1表中的所有内容

```
[root@localhost python-mysql]# python insert_table.py
[root@localhost python-mysql]# python select.py "select * from db1.t1"
1   Mac Mohan   20  M   2000.0
```

实例6：删除操作

删除操作用于删除数据表中的数据，以下实例演示了删除数据表 db1 中 AGE 大于 20 的所有数据：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost","root","uplooking","db1" ))

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 删除语句
sql = "DELETE FROM db1.t1 WHERE age > 20"
try:
   # 执行SQL语句
   cursor.execute(sql)
   # 提交修改
   db.commit()
except:
   # 发生错误时回滚
   db.rollback()

# 关闭连接
db.close()
```

实例7：执行事务

事务机制可以确保数据一致性。

事务应该具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。

- 原子性（atomicity）。一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
- 一致性（consistency）。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
- 隔离性（isolation）。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- 持久性（durability）。持续性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

Python DB API 2.0 的事务提供了两个方法 commit 或 rollback。

实例：

```
# SQL删除记录语句
sql = "DELETE FROM db1.t1 WHERE age > 20"
try:
   # 执行SQL语句
   cursor.execute(sql)
   # 向数据库提交
   db.commit()
except:
   # 发生错误时回滚
   db.rollback()
```

对于支持事务的数据库， 在Python数据库编程中，当游标建立之时，就自动开始了一个隐形的数据库事务。

commit()方法游标的所有更新操作，rollback（）方法回滚当前游标的所有操作。每一个方法都开始了一个新的事务。

错误处理

DB API中定义了一些数据库操作的错误及异常，下表列出了这些错误和异常:

异常	描述
Warning	当有严重警告时触发，例如插入数据是被截断等等。必须是 StandardError 的子类。
Error	警告以外所有其他错误类。必须是 StandardError 的子类。
InterfaceError	当有数据库接口模块本身的错误（而不是数据库的错误）发生时触发。 必须是Error的子类。
DatabaseError	和数据库有关的错误发生时触发。 必须是Error的子类。
DataError	当有数据处理时的错误发生时触发，例如：除零错误，数据超范围等等。 必须是DatabaseError的子类。
OperationalError	指非用户控制的，而是操作数据库时发生的错误。例如：连接意外断开、 数据库名未找到、事务处理失败、内存分配错误等等操作数据库是发生的错误。 必须是DatabaseError的子类。
IntegrityError	完整性相关的错误，例如外键检查失败等。必须是DatabaseError子类。
InternalError	数据库的内部错误，例如游标（cursor）失效了、事务同步失败等等。 必须是DatabaseError子类。
ProgrammingError	程序错误，例如数据表（table）没找到或已存在、SQL语句语法错误、 参数数量错误等等。必须是DatabaseError的子类。
NotSupportedError	不支持错误，指使用了数据库不支持的函数或API等。例如在连接对象上 使用.rollback()函数，然而数据库并不支持事务或者事务已关闭。 必须是DatabaseError的子类。

总结

- 安装模块 mysql-python
- 加载模块 import MySQLdb
- 打开数据库连接 db = MySQLdb.connect(host,user,password,database))
- 使用cursor()方法获取操作游标事务开始 cursor = db.cursor()
- 执行SQL语句 cursor.execute(sql)
- 向数据库提交 db.commit()
- 发生错误时回滚 db.rollback()
- 关闭连接 db.close()