# 1 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

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

## 1.1 安装MySQLdb

安装MySQLdb，请访问 <https://pypi.python.org/pypi/MySQL-python#downloads，hozhe>,或者http://www.codegood.com/downloads(Linux平台可以访问：[https://pypi.python.org/pypi/MySQL-python](https://pypi.python.org/pypi/MySQL-python" \t "http://www.runoob.com/python/_blank))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

## 1.2 操纵数据库

### 1.2.1 查询

**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

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

import MySQLdb

# 打开数据库连接

db = MySQLdb.connect(host='127.0.0.1',user='root',passwd='root',db='secondkill')

# 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，必须先执行execute（）

data = cursor.fetchone()

#使用tetchall（）等价于select \*

data=cur.fetchall()

# 返回值result为影响的行数

result=cur.execute('select \* from seckill')

print "Database version : %s " % data

-------------------------------------------------------------------

# SQL 查询语句

sql = "SELECT \* FROM EMPLOYEE \

WHERE INCOME > '%d'" % (1000)

try:

# 执行SQL语句

cursor.execute(sql)

# 获取所有记录列表

results = cursor.fetchall()

for row in results:

fname = row[0]

lname = row[1]

age = row[2]

sex = row[3]

income = row[4]

# 打印结果

print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \

(fname, lname, age, sex, income )

except:

print "Error: unable to fecth data"

----------------------------------------------------------------------------

# 关闭数据库连接

db.close()

### 1.2.2 插入数据

插入操作必须要提交

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

import MySQLdb

# 打开数据库连接

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# SQL 插入语句，直接插入

result=cur.execute("insert into seckill values(1004,'344元秒杀小米平衡车',200,'2016-9-10','2016-9-16','2016-9-23')")

try:

#用占位符，防止sql注入，所有类型都用%s

sql="insert into seckill (name,number,start\_time," \  
 "end\_time,create\_time) values(%s,%s,%s,%s,%s)"  
 params=('600yuan sui yi mai',200,'2016-9-10','2016-9-16','2016-9-23')  
 result=cur.execute(sql,params)  
 conn.commit()

# 执行sql语句

cursor.execute(sql)

# 提交到数据库执行，必须要提交

conn.commit()

except:

# 发生错误时回滚

conn.rollback()

# 关闭数据库连接

db.close()

也可以使用以下代码使用变量向SQL语句中传递参数:

..................................

user\_id = "test123"

password = "password"

con.execute('insert into Login values("%s", "%s")' % \

(user\_id, password))

### 1.2.3 批量操作

|  |
| --- |
| lis=[  (**'358元秒杀小米平衡车'**,200,**'2016-9-10'**,**'2016-9-16'**,**'2016-9-23'**),  (**'700元秒杀小米平衡车'**,200,**'2016-9-10'**,**'2016-9-16'**,**'2016-9-23'**),  (**'800元秒杀小米平衡车'**,200,**'2016-9-10'**,**'2016-9-16'**,**'2016-9-23'**), ] sql = **'insert into seckill (name,number,start\_time,'** \  **'end\_time,create\_time) values(%s,%s,%s,%s,%s)'** cur.executemany(sql,lis) |

### 1.2.4 用字典展示输出

|  |
| --- |
| cur=conn.cursor(cursorclass=MySQLdb.cursors.DictCursor)  输出格式：  ({'name': '1000???iphone6', 'start\_time': datetime.datetime(2016, 11, 23, 16, 56, 19), 'number': 189L, 'seckill\_id': 1000L, 'create\_time': datetime.datetime(2016, 11, 20, 15, 36, 39),...... |

### 1.2.5 游标定位

#绝对定位

cur.scroll(-1,mode=**'absolute'**)

#相对定位，相对于当前位置  
cur.scroll(0,mode=**'relative'**)

### 1.2.6 获取自增id

执行插入和提交后用：

New\_id=cur.lastrowid

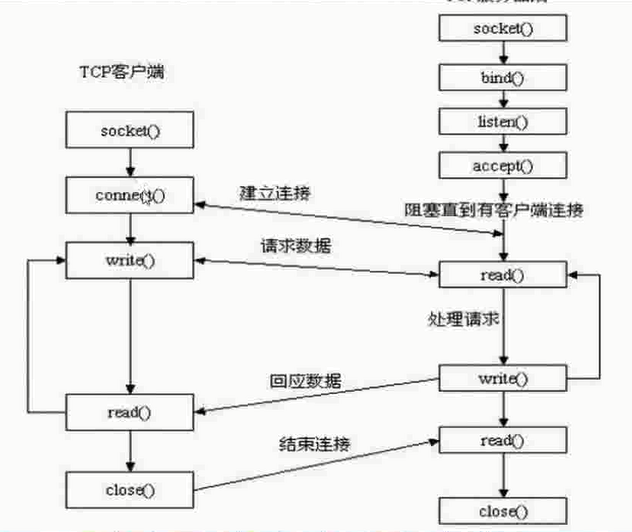
# 2 Python 网络编程

Python 提供了两个级别访问的网络服务。：

* 低级别的网络服务支持基本的 Socket，它提供了标准的 BSD Sockets API，可以访问底层操作系统Socket接口的全部方法。
* 高级别的网络服务模块 SocketServer， 它提供了服务器中心类，可以简化网络服务器的开发。

## 什么是 Socket?

Socket又称"套接字"，应用程序通常通过"套接字"向网络发出请求或者应答网络请求，使主机间或者一台计算机上的进程间可以通讯。



## socket()函数

Python 中，我们用 socket（）函数来创建套接字，语法格式如下：

socket.socket([family[, type[, proto]]])

### **参数**

* family: 套接字家族可以使AF\_UNIX或者AF\_INET
* type: 套接字类型可以根据是面向连接的还是非连接分为SOCK\_STREAM或SOCK\_DGRAM
* protocol: 一般不填默认为0.

### **Socket 对象(内建)方法**

| **函数** | | | **描述** |
| --- | --- | --- | --- |
| 服务器端套接字 | | | |
| s.bind() | 绑定地址（host,port）到套接字， 在AF\_INET下,以元组  （host,port）的形式表示地址。 | | |
| s.listen() | 开始TCP监听。backlog指定在拒绝连接之前，操作系统可以挂起的  最大连接数量。该值至少为1，大部分应用程序设为5就可以了。 | | |
| s.accept() | 被动接受TCP客户端连接,(阻塞式)等待连接的到来 | | |
| 客户端套接字 | | | |
| s.connect() | 主动初始化TCP服务器连接，一般address的格式为元组（hostname,port），  如果连接出错，返回socket.error错误。 | | |
| s.connect\_ex() | connect()函数的扩展版本,出错时返回出错码,而不是抛出异常 | | |
| 公共用途的套接字函数 | | | |
| s.recv() | | 接收TCP数据，数据以字符串形式返回，bufsize指定要接收的最大数据量。  flag提供有关消息的其他信息，通常可以忽略。 | |
| s.send() | | 发送TCP数据，将string中的数据发送到连接的套接字。  返回值是要发送的字节数量，该数量可能小于string的字节大小。 | |
| s.sendall() | | 完整发送TCP数据，完整发送TCP数据。将string中的数据发送到连接的  套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。 | |
| s.recvform() | | 接收UDP数据，与recv()类似，但返回值是（data,address）。  其中data是包含接收数据的字符串，address是发送数据的套接字地址。 | |
| s.sendto() | | 发送UDP数据，将数据发送到套接字，address是形式为（ipaddr，port）  的元组，指定远程地址。返回值是发送的字节数。 | |
| s.close() | | 关闭套接字 | |
| s.getpeername() | | 返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。 | |
| s.getsockname() | | 返回套接字自己的地址。通常是一个元组(ipaddr,port) | |
| s.setsockopt(level,optname,value) | | 设置给定套接字选项的值。 | |
| s.getsockopt(level,optname[.buflen]) | | 返回套接字选项的值。 | |
| s.settimeout(timeout) | | 设置套接字操作的超时期，timeout是一个浮点数，单位是秒。  值为None表示没有超时期。一般，超时期应该在刚创建套接字时设置，  因为它们可能用于连接的操作（如connect()） | |
| s.gettimeout() | | 返回当前超时期的值，单位是秒，如果没有设置超时期，则返回None。 | |
| s.fileno() | | 返回套接字的文件描述符。 | |
| s.setblocking(flag) | | 如果flag为0，则将套接字设为非阻塞模式，否则将套接字设为阻塞模式  （默认值）。非阻塞模式下，如果调用recv()没有发现任何数据，  或send()调用无法立即发送数据，那么将引起socket.error异常。 | |
| s.makefile() | | 创建一个与该套接字相关连的文件 | |

## 简单实例

### **服务端**

我们使用 socket 模块的 **socket** 函数来创建一个 socket 对象。socket 对象可以通过调用其他函数来设置一个 socket 服务。

现在我们可以通过调用 **bind(hostname, port)** 函数来指定服务的 *port(端口)*。

接着，我们调用 socket 对象的 *accept* 方法。该方法等待客户端的连接，并返回 *connection* 对象，表示已连接到客户端。

完整代码如下：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-# 文件名：server.py

import socket # 导入 socket 模块

s = socket.socket() # 创建 socket 对象

host = socket.gethostname() # 获取本地主机名

port = 12345 # 设置端口

s.bind((host, port)) # 绑定端口

s.listen(5) # 等待客户端连接

while True:

c, addr = s.accept() # 建立客户端连接。

print '连接地址：', addr

c.send('欢迎访问菜鸟教程！')

c.close() # 关闭连接

### **客户端**

接下来我们写一个简单的客户端实例连接到以上创建的服务。端口号为 12345。

**socket.connect(hosname, port )** 方法打开一个 TCP 连接到主机为 *hostname* 端口为 *port* 的服务商。连接后我们就可以从服务端后期数据，记住，操作完成后需要关闭连接。

完整代码如下：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-# 文件名：client.py

import socket # 导入 socket 模块

s = socket.socket() # 创建 socket 对象

host = socket.gethostname() # 获取本地主机名

port = 12345 # 设置端口好

s.connect((host, port))

print s.recv(1024)

s.close()

现在我们打开连个终端，第一个终端执行 server.py 文件：

$ python server.py

第二个终端执行 client.py 文件：

$ python client.py 欢迎访问菜鸟教程！

这是我们再打开第一个终端，就会看到有以下信息输出：

连接地址： ('192.168.0.118', 62461)

## SocketServer

socker同时只能处理一个请求，当有多个请求来时只能等待。

|  |
| --- |
| **import** SocketServer **class** MyServer(SocketServer.BaseRequestHandler):  **def** handle(self):  *#request为客户端socket对象，client\_address为客户端（host,port）元祖，最后为thtreadingTCPServer对象  #print self.request,self.client\_address,self.server* conn = self.request  flag=True  **while** flag:  rec = conn.recv(1024)  **print '客户端：'** + rec  **if** rec == **'exit'**:  flag=False  inp = raw\_input(**'服务端：'**)  conn.send(inp)  **def** finish(self):  **pass   def** setup(self):  **pass  if** \_\_name\_\_==**'\_\_main\_\_'**:  server=SocketServer.ThreadingTCPServer((**'127.0.0.1'**,8090),MyServer)  server.serve\_forever() |

## Python Internet 模块

以下列出了 Python 网络编程的一些重要模块：

|  |  |  |  |
| --- | --- | --- | --- |
| **协议** | **功能用处** | **端口号** | **Python 模块** |
| HTTP | 网页访问 | 80 | httplib, urllib, xmlrpclib |
| NNTP | 阅读和张贴新闻文章，俗称为"帖子" | 119 | nntplib |
| FTP | 文件传输 | 20 | ftplib, urllib |
| SMTP | 发送邮件 | 25 | smtplib |
| POP3 | 接收邮件 | 110 | poplib |
| IMAP4 | 获取邮件 | 143 | imaplib |
| Telnet | 命令行 | 23 | telnetlib |
| Gopher | 信息查找 | 70 | gopherlib, urllib |