这是作者的系列网络安全自学教程，主要是关于网安工具和实践操作的在线笔记，特分享出来与博友共勉，希望您们喜欢，一起进步。前文分享了Python网络攻防相关基础知识，包括正则表达式、Web编程和套接字通信，本文将继续分析Python攻防之多线程、C段扫描和数据库编程。本文参考了爱春秋ADO老师的课程内容，这里也推荐大家观看他Bilibili和ichunqiu的课程，同时也结合了作者之前的经验进行讲解。

作者作为网络安全的小白，分享一些自学基础教程给大家，希望你们喜欢。同时，更希望你能与我一起操作深入进步，后续也将深入学习网络安全和系统安全知识并分享相关实验。总之，希望该系列文章对博友有所帮助，写文不容易，大神请飘过，不喜勿喷，谢谢！

下载地址：https://github.com/eastmountyxz/NetworkSecuritySelf-study
百度网盘：https://pan.baidu.com/s/1dsunH8EmOB_tlHYXXguOeA 提取码：izeb

文章目录

前文学习：

[网络安全自学篇] 一.入门笔记之看雪Web安全学习及异或解密示例
[网络安全自学篇] 二.Chrome浏览器保留密码功能渗透解析及登录加密入门笔记
[网络安全自学篇] 三.Burp Suite工具安装配置、Proxy基础用法及暴库示例
[网络安全自学篇] 四.实验吧CTF实战之WEB渗透和隐写术解密
[网络安全自学篇] 五.IDA Pro反汇编工具初识及逆向工程解密实战
[网络安全自学篇] 六.OllyDbg动态分析工具基础用法及Crakeme逆向破解
[网络安全自学篇] 七.快手视频下载之Chrome浏览器Network分析及Python爬虫探讨
[网络安全自学篇] 八.Web漏洞及端口扫描之Nmap、ThreatScan和DirBuster工具
[网络安全自学篇] 九.社会工程学之基础概念、IP获取、IP物理定位、文件属性
[网络安全自学篇] 十.论文之基于机器学习算法的主机恶意代码
[网络安全自学篇] 十一.虚拟机VMware+Kali安装入门及Sqlmap基本用法

[网络安全自学篇] 十二.Wireshark安装入门及抓取网站用户名密码（一）
[网络安全自学篇] 十三.Wireshark抓包原理（ARP劫持、MAC泛洪）及数据流追踪和图像抓取（二）
[网络安全自学篇] 十四.Python攻防之基础常识、正则表达式、Web编程和套接字通信（一）

前文欣赏：
[渗透&攻防] 一.从数据库原理学习网络攻防及防止SQL注入
[渗透&攻防] 二.SQL MAP工具从零解读数据库及基础用法
[渗透&攻防] 三.数据库之差异备份及Caidao利器
[渗透&攻防] 四.详解MySQL数据库攻防及Fiddler神器分析数据包

[python] 专题七.网络编程之套接字Socket、TCP和UDP通信实例
[python] 专题八.多线程编程之thread和threading
[python] 专题九.Mysql数据库编程基础知识

参考文献：
《安全之路Web渗透技术及实战案例解析》陈小兵老师
《Wireshark数据包分析实战》第二版 Chris Sanders
《TCP/IP协议栈详解卷一》 W.Richard Stevens

《Wireshark协议分析从入门到精通》-51cto老师
https://www.bilibili.com/video/av29479068
2019 Python黑客编程：安全工具开发 - bilibili 白帽黑客教程
http://www.heibanke.com/lesson/crawler_ex00/
python subprocess模块使用总结

> 声明：本人坚决反对利用社会工程学方法进行犯罪的行为，一切犯罪行为必将受到严惩，绿色网络需要我们共同维护，更推荐大家了解它们背后的原理，更好地进行防护。

一.Python多线程

1.进程和线程

进程： 是程序的一次执行，每个进程都有自己的地址空间、内存、数据栈及其他记录运行轨迹的辅助数据。
线程： 所有的线程都运行在同一个进程当中，共享相同的运行环境。线程有开始、顺序执行和结束三个部分。

由于单线程效率低，这里引入了多线程编程。
计算机的核心是CPU，它承担了所有的计算任务，它就像一座工厂，时刻运行着。假定工厂的电力有限，一次只能供给一个车间使用。也就是说，一个车间开工的时候，其他车间都必须停工。背后的含义就是，单个CPU一次只能运行一个任务。
进程就好比工厂的车间，它代表CPU所能处理的单个任务。任一时刻，CPU总是运行一个进程，其他进程处于非运行状态。一个车间里，可以有很多工人。他们协同完成一个任务。线程就好比车间里的工人，一个进程可以包括多个线程。

2.thread模块

Python thread模块可以调用下述函数实现多线程开启。它将产生一个新线程，在新的线程中用指定的参数和可选的kwargs来调用这个函数。
start_new_thread(function, args kwargs=None)

注意：使用这个方法时，一定要加time.sleep()函数，否则每个线程都可能不执行。此方法还有一个缺点，遇到较复杂的问题时，线程数不易控制。

```
# -*- coding: utf-8 -*-
# By:CSDN Eastmount 2019-10-05
import thread
import time

def fun1():
    print "hello world %s" % time.ctime()

#多线程
def main():
    thread.start_new_thread(fun1, ()) #无参数
    thread.start_new_thread(fun1, ())
    time.sleep(2)
    print "over"

#程序成功在同一时刻运行两个函数
if __name__ == '__main__':
    main()
```

输出结果如下图所示：

```
>>>
hello world Sat Oct 05 12:02:59 2019hello world Sat Oct 05 12:02:59 2019

over
>>>
```

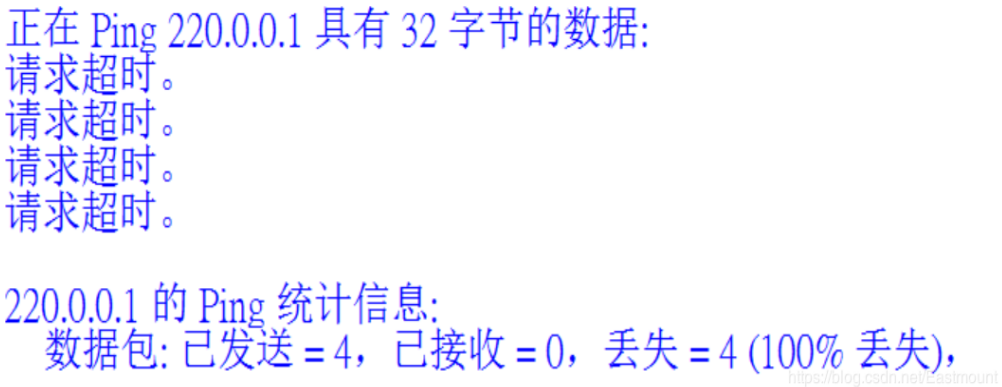
上面的代码简单讲述了thread模块的多线程使用。但实际应用中这种例子遇到比较少，而哪一种情况比较多呢？假如说所有C段的地址ping探测其是否存活，代码如下。

```
# -*- coding: utf-8 -*-
# By:CSDN Eastmount 2019-10-05
import thread
import time
from subprocess import Popen, PIPE

def ping_check():
    #check = Popen(['/bin/bash', '-c', 'ping -c 2 ' + '127.0.0.1'], stdi
    ip = '127.0.0.1'
    #ping指定次数后停止ping 但报错访问被拒绝，选项 -c 需要具有管理权限。
    #check = Popen("ping -c 3 {0} \n".format(ip), stdin=PIPE, stdout=PIPI
    check = Popen("ping {0} \n".format(ip), stdin=PIPE, stdout=PIPE,  she
    data = check.stdout.read() #数据
    print data

#程序成功在同一时刻运行两个函数
if __name__ == '__main__':
    ping_check()
```

如果输入的ip地址为本机127.0.0.1，则输出正常连通结果，如下所示。

```
正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms，最长 = 0ms，平均 = 0ms
```


如果输入的ip地址为本机220.0.0.1，则提示超时，如下图所示。



接着思考：如何对一个C段网址进行ping探测呢?
设计一个循环，如果主机不存在，返回的是timeout；如果主机存在，则包含TTL字样，这里以TTL为判断标准，从而判断存活的数据。

```
# -*- coding: utf-8 -*-
# By:CSDN Eastmount 2019-10-05
import thread
import time
from subprocess import Popen, PIPE

def ping_check():
    ip = '127.0.0.1'
    check = Popen("ping {0} \n".format(ip), stdin=PIPE, stdout=PIPE,  she
    data = check.stdout.read() #数据
    if 'TTL' in data: #存活
        print 'UP'

#程序成功在同一时刻运行两个函数
if __name__ == '__main__':
    ping_check()
```

输出结果为“UP”。

3.检测ichunqiu地址

接下来我们尝试检测ichunqiu网站的ip地址存活情况。首先，调用ping命令检测该网站的ip地址，即：1.31.128.240。

这里将ping_check()函数设置一个传递参数，对应ip地址，对它进行探测；通过thread线程实现ip地址存活性探测，能探测到很多存活的主机。

```
# -*- coding: utf-8 -*-
# By:CSDN Eastmount 2019-10-05
import thread
import time
from subprocess import Popen, PIPE

def ping_check(ip):
    check = Popen("ping {0} \n".format(ip), stdin=PIPE, stdout=PIPE, shel
    data = check.stdout.read() #数据
    if 'TTL' in data: #存活
        print '%s is UP' % ip

#主函数
if __name__ == '__main__':
    #寻找目标 ichunqiu  1.31.128.240
    for i in range(1,255):
        ip = '1.31.128.' + str(i)
        #多线程方法
        thread.start_new_thread(ping_check, (ip, ))
        time.sleep(0.1)
```

输出结果如下图所示：

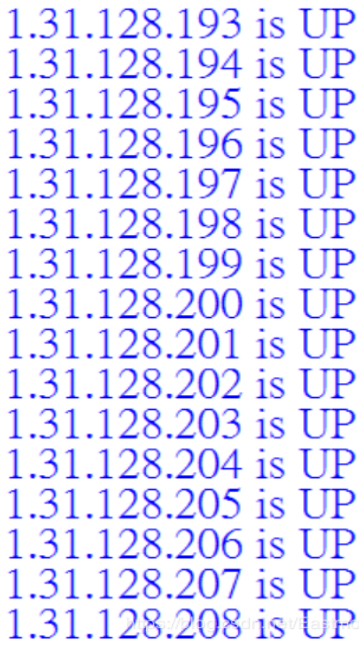
```
1.31.128.243 is UP
1.31.128.244 is UP
1.31.128.245 is UP1.31.128.246 is UP

1.31.128.247 is UP
1.31.128.248 is UP
1.31.128.249 is UP
1.31.128.250 is UP
1.31.128.215 is UP
 1.31.128.214 is UP
1.31.128.252 is UP
1.31.128.254 is UP
1.31.128.221 is UP1.31.128.220 is UP

1.31.128.193 is UP1.31.128.235 is UP
```


问题：

在多线程编程中，几个线程是同时启动，所以输出也是输出在一行，那怎么才能换行输出呢？这里使用系统输出。

```
# -*- coding: utf-8 -*-
# By:CSDN Eastmount 2019-10-05
import thread
import time
from subprocess import Popen, PIPE
import sys

def ping_check(ip):
    check = Popen("ping {0} \n".format(ip), stdin=PIPE, stdout=PIPE, she
    data = check.stdout.read() #数据
    if 'TTL' in data: #存活
        sys.stdout.write('%s is UP \n' % ip)

#主函数
if __name__ == '__main__':
    #寻找目标 ichunqiu  1.31.128.240
    for i in range(1,255):
        ip = '1.31.128.' + str(i)
        #多线程方法
        thread.start_new_thread(ping_check, (ip, ))
        time.sleep(0.1)
```

按行输出结果，如下图所示：



4.threading模块

thread模块存在一些缺点，尤其是线程数不能被控制。下面使用threading解决线程数可控制的问题。

- 使用threading模块
- 子类化Thread类

```
# -*- coding: utf-8 -*-
import threading
import time
import sys

def fun1(key):
    sys.stdout.write('hello %s:%s \n'%(key, time.ctime()))

def main():
    threads = []
    keys = ['a', 'b', 'c']

    #线程数
    threads_count = len(keys)

    for i in range(threads_count):
        t = threading.Thread(target=fun1,  args=(keys[i],))
        threads.append(t)

    for i in range(threads_count):
        threads[i].start()
```

```
    for i in range(threads_count):
        threads[i].join()

if __name__ == '__main__':
    main()
```

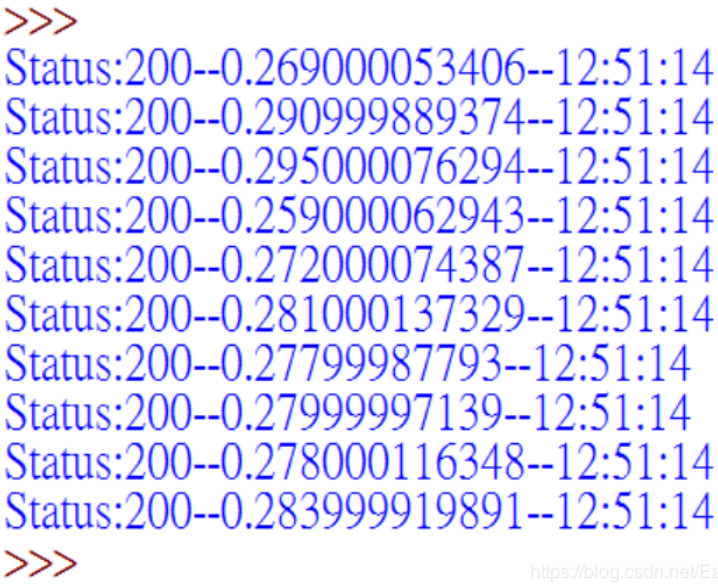
输出结果如下图所示，三个线程同时发生。

```
>>>
hello a:Sat Oct 05 12:45:07 2019
hello b:Sat Oct 05 12:45:07 2019
hello c:Sat Oct 05 12:45:07 2019
>>>
```

多线程threading方法实现了我们想要的功能，能够控制线程数，例如想写成requests模块，获取网站的status_code状态码。

```
# -*- coding: utf-8 -*-
import threading
import time
import requests
import sys

def fun1():
    time_start = time.time()
    r = requests.get(url="http://www.eastmountyxz.com/")
    times = time.time() - time_start
    #print('Status:%s--%s--%s'%(r.status_code, times, time.strftime('%H:
    sys.stdout.write('Status:%s--%s--%s\n'%(r.status_code, times, time.s

def main():
    threads = []
    #线程数 网页访问10次
    threads_count = 10

    for i in range(threads_count):
        t = threading.Thread(target=fun1,  args=())
        threads.append(t)

    for i in range(threads_count):
        threads[i].start()

    for i in range(threads_count):
        threads[i].join()

if __name__ == '__main__':
    main()
```

输出结果如下图所示：



二.多线程+Queue实现C段扫描

生产者-消费者问题和Queue模块：

- Queue模块：qsize()、empty()、full()、put()、get()
- 完美搭配，Queue+Thread

虽然threading解决了线程数可控问题，但是面对复杂问题的时候，比如生产者和消费者问题，仍然不能很好地解决。

生产者和消费者问题
生产者生产货物，将货物放到队列数据中，生产者在生产这些货物时，它的时间是不确定的；当生存着将货物交给消费者，消耗的时间也是不确定的；由于两个时间都不确定，多线程编程存在一定问题。

这里引入Queue模块解决该问题。

```
import Queue

queue = Queue.Queue()

for i in range(10):
    queue.put(i)
print(queue.empty())
print(queue.qsize())
```

```
#取数据 get依次取出里面的数据
print(queue.get())
print(queue.get())
```

输出结果如下所示：

```
>>>
False
10
0
1
>>>
```

生产者利用Queue将所有数据货物按顺序放入Queue，接着消费者依次取出Queue中的数据。接着实现C段扫描。

```
# -*- coding: utf-8 -*-
# By:CSDN Eastmount 2019-10-05
import threading
import Queue
import sys
from subprocess import Popen, PIPE

#定义一个类 传入参数queue
class DoRun(threading.Thread):
    def __init__(self, queue):
        threading.Thread.__init__(self)
        self._queue = queue

    def run(self):
        #非空取数据
        while not self._queue.empty():
            ip = self._queue.get()
            #print ip
            check_ping = Popen("ping {0} \n".format(ip), stdin=PIPE, std
            data = check_ping.stdout.read()
            if 'TTL' in data:
                sys.stdout.write(ip+' is UP.\n')

def main():
    threads = []
    threads_count = 100
```

```
    queue = Queue.Queue()

    #放入ip地址
    for i in range(1, 255):
        queue.put('1.31.128.' + str(i))

    for i in range(threads_count):
        threads.append(DoRun(queue))

    for i in threads:
        i.start()

    for i in threads:
        i.join()

if __name__ == '__main__':
    main()
```

最终输出结果如下图所示，通过该代码可以实现检测某网站ip段的存活情况。

```
>>>
1.31.128.129 is UP.
1.31.128.139 is UP.
1.31.128.138 is UP.
1.31.128.136 is UP.
1.31.128.140 is UP.
1.31.128.137 is UP.
1.31.128.141 is UP.
1.31.128.142 is UP.
1.31.128.130 is UP.
1.31.128.143 is UP.
1.31.128.145 is UP.
1.31.128.147 is UP.
1.31.128.153 is UP.
1.31.128.152 is UP.
1.31.128.149 is UP.
1.31.128.151 is UP.
1.31.128.144 is UP.
1.31.128.162 is UP.
1.31.128.155 is UP.
```

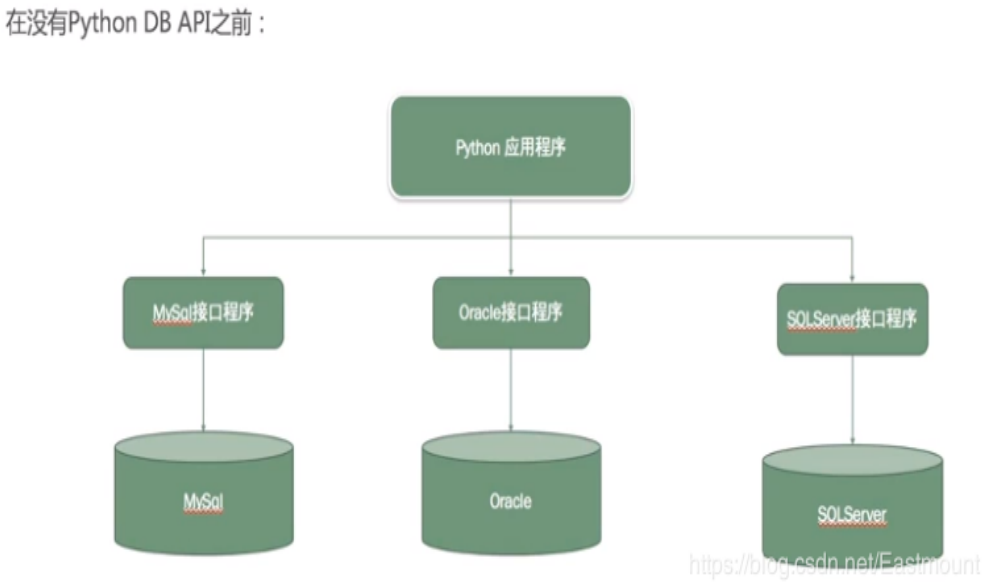
三.数据库编程

1.基础概念

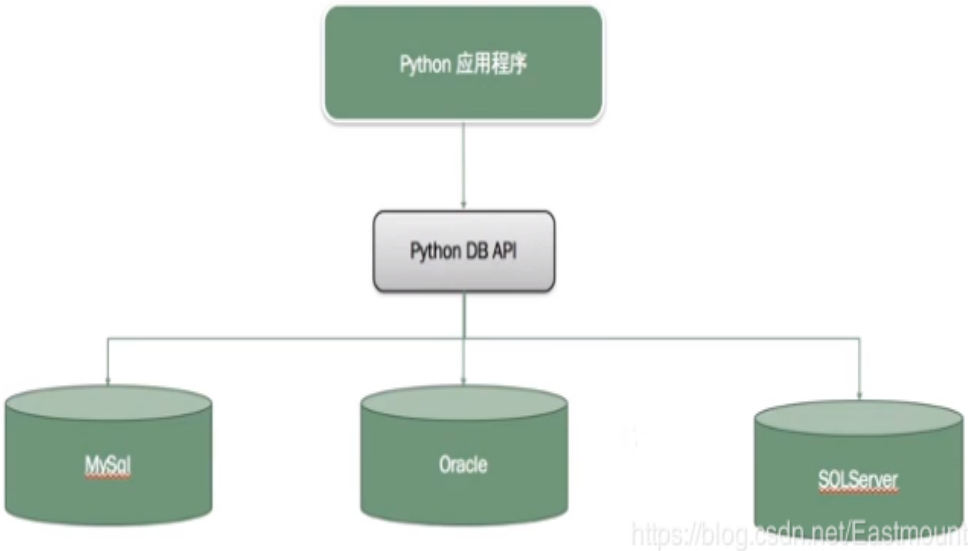
Python数据库编程，这里直接引入核心代码即可，更多参考前文：[python] 专题九.Mysql数据库编程基础知识

- Python DB API
- Python MySQL 开发环境
- Python数据库编程实例

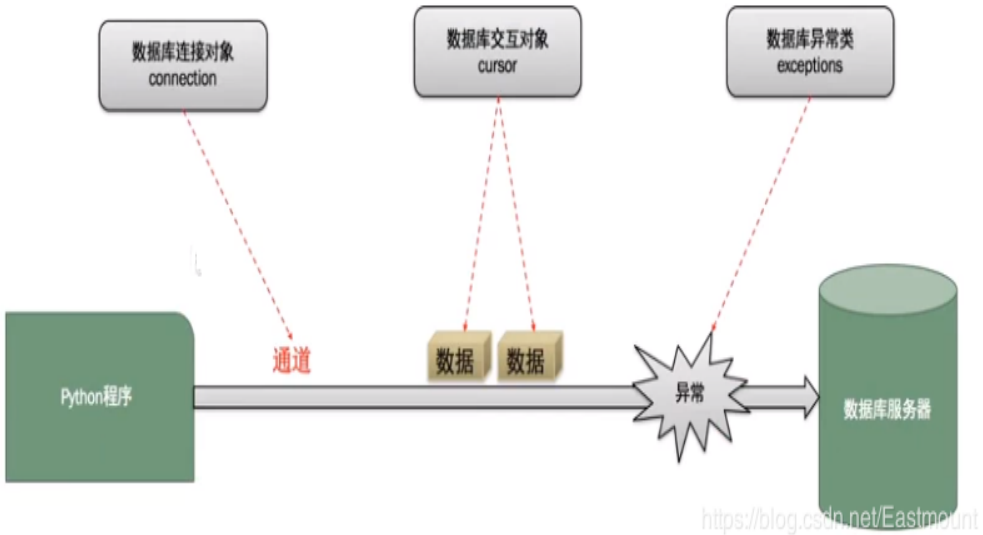
早期每个数据库和Python建立连接时，都需要有对应的接口程序，这样导致数据库比较混乱，需要学习很多新的接口程序并修改大量代码。

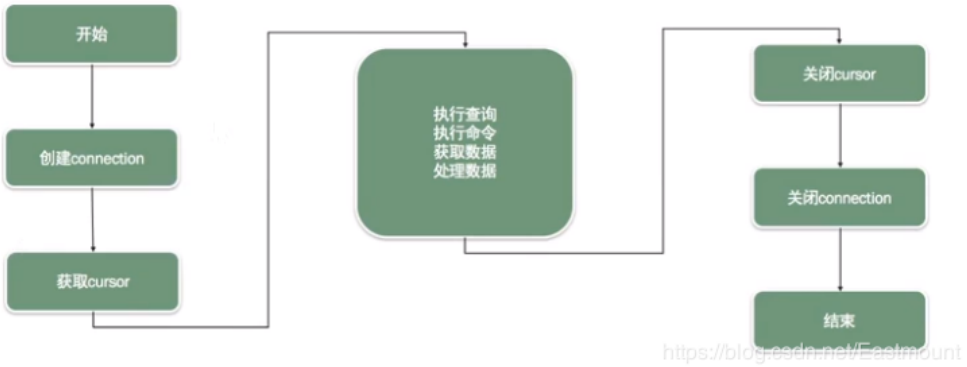


后来为了解决该问题，我们定义了相关的接口规范，所有数据库接入其中即可编程。我们只要学会Python DB API，就能实现对不同版本数据库的操作，这样更加方便快捷，更改的代码较少。



Python访问数据库的基本流程如下图所示：





2.MySQLdb

Python调用MsSQL需要导入MySQLdb库，如下：

```
import MySQLdb
```

connect()函数
主要使用的方法是connect对象。connect()方法生成一个connect对象，用于访问数据库，其参数如下：

```
user:Username
password:Password
host:Hostname
database:DatabaseName
dsn:Data source name
```

注意并非所有的接口程序都严格按照这种格式，如MySQLdb。

```
import MySQLdb
conn = MySQLdb.connect(host='localhost', db='test01', user='root', passw
```

connect()对象方法如下：

> lose():关闭数据库连接，或者关闭游标对象
> commit():提交当前事务
> rollback()：取消当前事务
> cursor()：创建游标或类游标对象
> errorhandler(cxn,errcls,errval)：作为已给游标的句柄

注意，执行close()方法则上述的连接对象方法不能再使用，否则发生异常。commit()、rollback()、cursor()或许更对于支持事务的数据库更有意义。数据库事务(Database Transaction)，是指作为单个逻辑工作单元执行的一系列操作，要么完整地执行，要么完全地不执行。 一旦你完成了数据库连接，关闭了游标对象，然后在执行commit()提交你的操作，然后关闭连接。

游标对象
上面说了connect()方法用于提供连接数据库的接口，如果要对数据库操作那么还需要使用游标对象。游标对象的属性和方法：

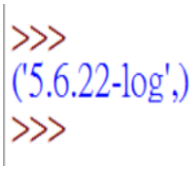
> fetchone()：可以看作fetch(取出) one(一个)，也就是得到结果集的下一行(一行)。
> fetchmany(size):可以看作fetch(取出)many(多个)，这里的参数是界限，得到结果集的下几行(几行)
> fetchall()：顾名思义，取得所有。
> execute(sql)：执行数据库操作，参数为sql语句。
> close():不需要游标时尽可能的关闭

下面是一个获取MySQL数据库版本的代码，它覆盖了Python链接数据库的基本过程。

```
# -*- coding: utf-8 -*-
import MySQLdb

#建立连接
conn = MySQLdb.connect(
    host = '127.0.0.1',
    port = 3306,
    user = 'root',
    passwd = '123456'
    )

#执行最简单的语句
```

```
cus = conn.cursor()

#查看MySQL的版本
sql = 'select version()'

cus.execute(sql)

#查看返回结果
print(cus.fetchone())

#关闭连接和对象
cus.close()
conn.close()
```

输出结果如下图所示：

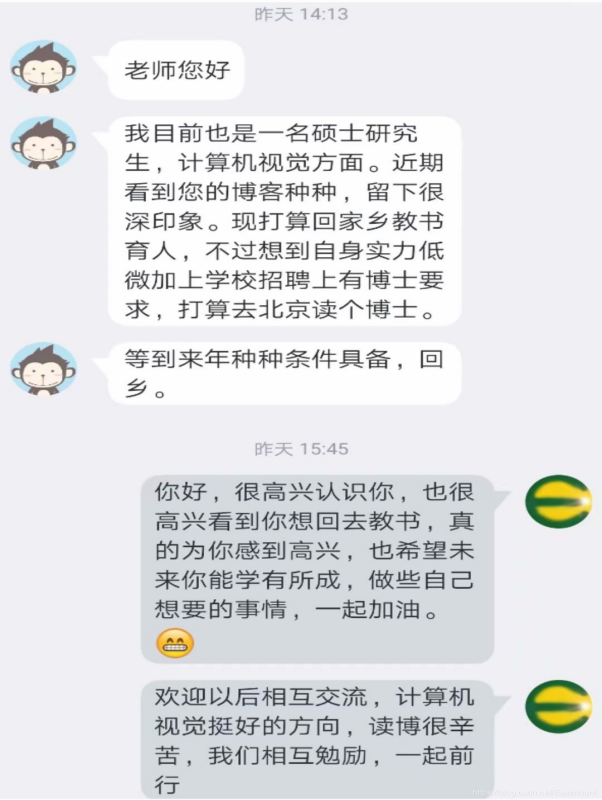


Python网络爬虫和MySQL数据库结合的文章参考作者前文：
[python爬虫] 招聘信息定时系统 (一).BeautifulSoup爬取信息并存储MySQL
[python爬虫] Selenium爬取内容并存储至MySQL数据库

四.总结

希望这篇文章对你有所帮助，这是Python网络攻防非常基础的一篇博客，后续作者也将继续深入学习，制作一些常用的小工具供大家交流。

下午通过CSDN认识了一位计算机视觉方向的博友，我们相约博士毕业回各自的家乡教书，虽天各一方，并且不认识，但已是朋友，祝好。



晚上终于看懂了第一篇恶意代码检测的论文，溯源和扫描还需要多学习，多实验。同时，看了几篇讲梵高的文章，挺不错的。分享一段：

> 梵高的生活较为落魄和不堪，但他的作品里永远是明亮的，美好的，纯真的，积极的。1889年，他在精神病院里画出了《星空》，那大概是梵高内心最纯洁的颜色，罗纳河上的星空，让处于压迫的内心仍闪烁着点点星光。善良淳朴的人性之美，和坚持纯粹的艺术之美交合着。这大概就是为什么他的作品特别能打动人的原因！
> 晚安，女神，我也准备学学油画

(By:Eastmount 2019-10-05 晚上11点 http://blog.csdn.net/eastmount/)