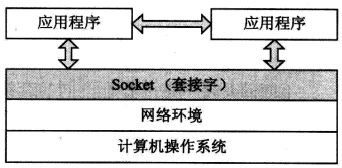
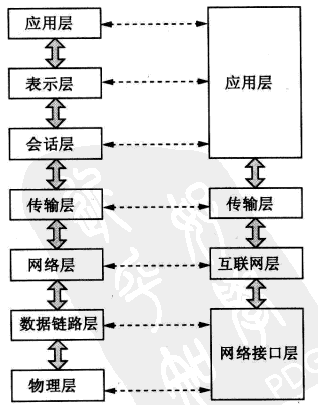
Python编程基础之网络编程

**一、简介**

       网络中绝大部分网络协议都是使用socket开发的， Python提供了访问底层操作系统Socket接口的全部方法，需要的时候这些接口可以提供灵活而强有力的功能，使用TCP/IP和UDP/IP可以很容易的创建客户端和服务器。

**二、详解**

**1、网络编程简介**

（1）客户/服务器  
       服务器是一个软件或硬件，用于提供客户需要的“服务”。服务器存在的唯一目的就是等待客户的请求，给这些客户服务，然后再等待其它的请求。客户连上一个(预先已知的)服务器，提出自己的请求，发送必要的数据，然后就等待服务器的完成请求或说明失败原因的反馈。如今最常见的客户/服务器结构，一个用户或客户电脑通过 Internet 从服务器上取数据。  
（2）套接字  
        套接字是一种具有之前所说的“通讯端点”概念的计算机网络[数据结构](http://lib.csdn.net/base/datastructure)。**网络化的应用程序在开始任何通讯之前都必需要创建套接字**。一开始，套接字被设计用在同一台主机上多个应用程序之间的通讯，这也被称进程间通讯或IPC。套接字有两种，分别是基于文件型的和基于网络型的。Python 只支持AF\_UNIX、AF\_NETLINK和AF\_INET家族。  
  
（3）主机与端口  
        一个 Internet 地址由网络通讯所必需的主机与端口组成，合法的端口号范围为0到65535。其中，小于 1024 的端口号为系统保留端口。如果你所使用的是Unix操作系统，保留的端口号（及其对应的服务/协议和套接字类型）可以通过/etc/services文件获得。  
（4）面向连接与无连接  
        面向连接，这种通讯方式也被称为“虚电路”或“流套接字”。面向连接的通讯方式提供了顺序的、可靠的、不会重复的数据传输，而且也不会被加上数据边界。这也意味着，每一个要发送的信息，可能会被拆分成多份，每一份都会不多不少地正确到达目的地。然后被重新按顺序拼装起来，传给正在等待的应用程序。实现这种连接的主要协议就是传输控制协议(即 TCP)，指定套接字类型为 SOCK\_STREAM，表达了它做为流套接字的特点。由于这些套接字使用 Internet 协议(IP)来查找网络中的主机，这样形成的整个系统，一般会由这两个协议(TCP 和 IP)来提及即TCP/IP。  
        无连接，与虚电路完全相反的是数据报型的无连接套接字。这意味着，无需建立连接就可以进行通讯。但这时数据到达的顺序，可靠性及数据不重复性就无法保证了。数据报会保留数据边界，这就表示数据不会像面向连接的协议那样被拆分成小块。既然数据报有这么多缺点，为什么还要使用它呢？由于面向连接套接字要提供一些保证以及要维持虚电路连接，这都是很重的额外负担。数据报没有这些负担，所以它更“便宜”。通常能提供更好的性能，更适合某些应用场合。实现这种连接的主要协议就是用户数据报协议(即 UDP) 要创建UDP套接字就得在创建的时候，指定套接字类型为SOCK\_DGRAM。一般会由这两个协议(UDP和IP)来提及，即UDP/IP。  
         TCP/IP和OSI模型对应关系：  


**2、Python中的网络编程**

        如下分别为服务器和客户端的通讯流程：

|  |  |
| --- | --- |
| http://img.blog.csdn.net/20150330145444759?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdGFpeWFuZzE5ODc5MTI=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center | http://img.blog.csdn.net/20150330145502497?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdGFpeWFuZzE5ODc5MTI=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center |

网络编程主要使用 socket 模块。模块中的 socket()函数被用来创建套接字。套接字也有自己的一套函数来提供基于套接字的网络通讯。  
（1）socket()模块函数:socket.socket()函数来创建套接字，

语法为：

socket(socket\_family, socket\_type, protocol=0)，

socket\_family可以是AF\_UNIX或AF\_INET。

socket\_type可以是SOCK\_STREAM或SOCK\_DGRAM。

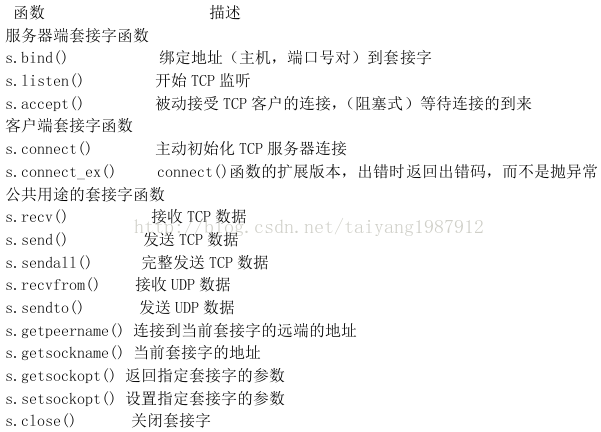
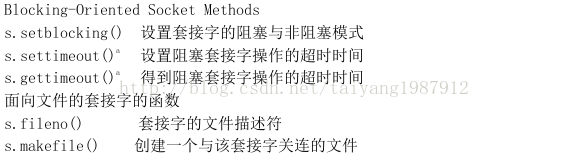
protocol一般不填,默认值为0。

创建一个TCP/IP的套接字：

tcpSock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)；

同样地创建一个UDP/IP的套接字：

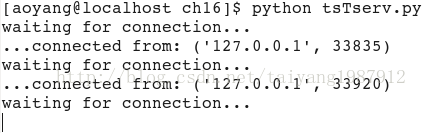
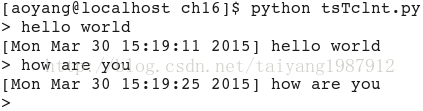
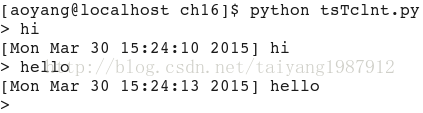
udpSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)。

（2）套接字对象(内建)方法:套接字对象的常用函数：  
  
（3）创建一个TCP服务器  
       创建一个能接收客户的消息，在消息前加一个时间戳后返回的TCP服务器。

1. #!/usr/bin/env python
3. from socket import \*
4. from time import ctime
6. HOST = ''
7. PORT = 21567
8. BUFSIZ = 1024
9. ADDR = (HOST, PORT)
11. tcpSerSock = socket(AF\_INET, SOCK\_STREAM)
12. tcpSerSock.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, True)
13. tcpSerSock.setsockopt(SOL\_SOCKET, SO\_REUSEPORT, True)
14. tcpSerSock.bind(ADDR)
15. tcpSerSock.listen(5)
17. while True:
18. print 'waiting for connection...'
19. tcpCliSock, addr = tcpSerSock.accept()
20. print '...connected from:', addr
22. while True:
23. data = tcpCliSock.recv(BUFSIZ)
24. if not data:
25. break
26. tcpCliSock.send('[%s] %s' % (ctime(), data))
28. tcpCliSock.close()
29. tcpSerSock.close()

       所有的套接字都用socket.socket()函数来创建，它必需要“绑定”到一个本地的地址上，TCP是一个面向连接的通讯系统，开始工作之前，要先完成一些设置，然后就进入无限循环了。一个简单的（单线程的）服务器会调用accept()函数等待连接的到来。默认情况下accept()函数是阻塞式的，即程序在连接到来之前会处于挂起状态，套接字也支持非阻塞模式。一旦接收到一个连接，accept()函数就会返回一个单独的客户的套接字用于后续的通讯。在临时套接字创建好之后，通讯就可以开始了，客户与服务器都使用这个新创建的套接字进行数据的发送与接收，直到通讯的某一方关闭了连接或发送了一个空字符串之后，通讯就结束了。客户连接关闭后，服务器继续等待下一个客户的连接。程序最后都会把服务器的套接字关闭。  
（4）创建TCP客户端  
        在客户有了套接字之后，马上就可以调用connect()函数去连接服务器。连接建立后，就可以与服务器开始对话了。在对话结束后，客户就可以关闭套接字，结束连接。

1. #!/usr/bin/env python
3. from socket import \*
5. HOST = 'localhost'
6. PORT = 21567
7. BUFSIZ = 1024
8. ADDR = (HOST, PORT)
10. tcpCliSock = socket(AF\_INET, SOCK\_STREAM)
11. tcpCliSock.connect(ADDR)
13. while True:
14. data = raw\_input('**>** ')
15. if not data:
16. break
17. tcpCliSock.send(data)
18. data = tcpCliSock.recv(BUFSIZ)
19. if not data:
20. break
21. print data
23. tcpCliSock.close()

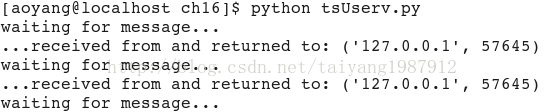
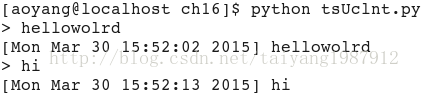
       TCP 客户端连接到服务器，提示用户输入要传输的数据，然后显示服务器返回的加了时间戳的结果。  
（5）运行TCP客户端和服务器  
       先运行服务器，主动地建立一个连接，等待客户端。有客户连接上来的时候，会显示一个“... connected from ...”信息。  
  
       开客户，在客户端的输入，接着显示客户端的输出，不输入数据直接按回车键就可以退出程序。  
          
      “友好地”退出的一个方法就是把服务器的无限循环放在一个try-except语句的try子句当中，并捕获EOFError和KeyboardInterrupt异常。在异常处理子句中，调用 close()函数关闭服务器的套接字。  
（6）创建一个UDP服务器  
       UDP服务器不是面向连接的，所以不用像TCP服务器那样做那么多设置工作。事实上，并不用设置什么东西,直接等待进来的连接就好了。

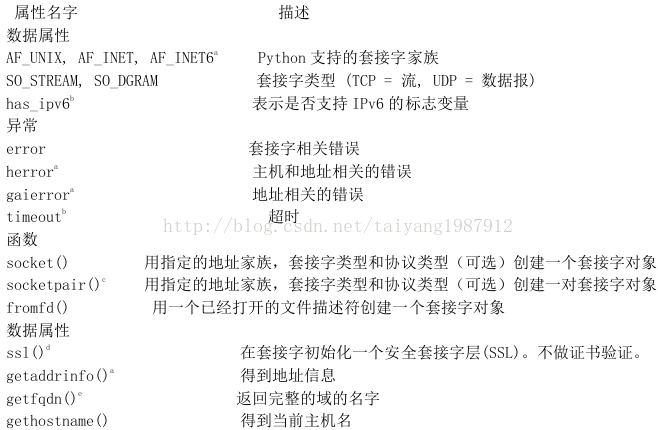
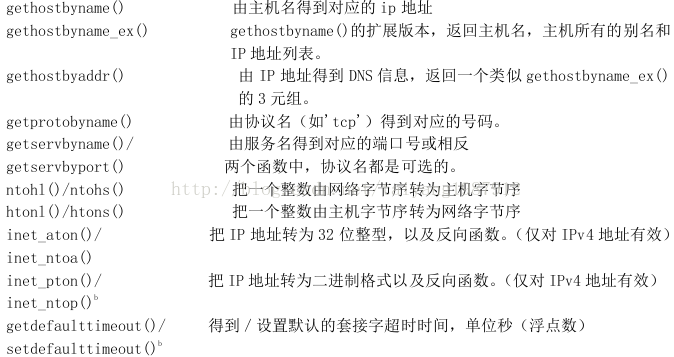
1. #!/usr/bin/env python
3. from socket import \*
4. from time import ctime
6. HOST = ''
7. PORT = 21567
8. BUFSIZ = 1024
9. ADDR = (HOST, PORT)
11. udpSerSock = socket(AF\_INET, SOCK\_DGRAM)
12. udpSerSock.bind(ADDR)
14. while True:
15. print 'waiting for message...'
16. data, addr = udpSerSock.recvfrom(BUFSIZ)
17. udpSerSock.sendto('[%s] %s' % (ctime(), data), addr)
18. print '...received from and returned to:', addr
20. udpSerSock.close()

        进入服务无限循环后，(被动地)等待(数据报)消息的到来。当有消息进来时,就处理它(在前面加时间戳) 把结果返回回去，然后再去等等下一个消息。  
（7）创建一个UDP客户端  
       程序会提示用户输入要传给服务器的信息，显示服务器返回的加了时间戳的结果。

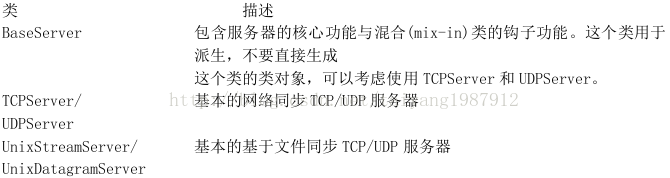
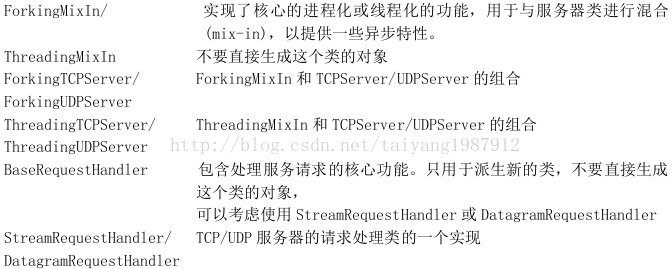
1. #!/usr/bin/env python
3. from socket import \*
5. HOST = 'localhost'
6. PORT = 21567
7. BUFSIZ = 1024
8. ADDR = (HOST, PORT)
10. udpCliSock = socket(AF\_INET, SOCK\_DGRAM)
12. while True:
13. data = raw\_input('**>** ')
14. if not data:
15. break
16. udpCliSock.sendto(data, ADDR)
17. data, ADDR = udpCliSock.recvfrom(BUFSIZ)
18. if not data:
19. break
20. print data
22. udpCliSock.close()

（8）运行UDP服务器和客户端  
UDP服务器：

  
UDP客户器：  


（9）套接字模块属性  
       socket.socket()函数之外，socket 模块还有很多属性可供网络应用程序使用。  


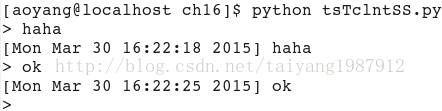
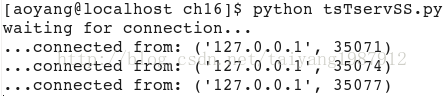
**3、SocketServer模块**

       SocketServer是标准库中一个高级别的模块，用于简化网络客户与服务器的实现。模块中，已经实现了一些可供使用的类。  
       SocketServer 模块的类：  
  
  
       新实现与之前有很多相似之处，现在很多繁杂的事情已经被封装好了，不用再去关心那个样板代码了。用面向对象的方法可以帮助我们更好的组织数据与逻辑功能。程序现在是“事件驱动”了，这就意味着，只有在事件出现的时候，程序才有“反应”，事件包含发送与接收数据两种。  
（1）创建一个SocketServerTCP服务器  
       在代码中，先导入服务器类，然后像之前一样定义主机常量，主机常量后就是请求处理器类，最后是启动代码。SocketServer里的TCPServer和StreamRequestHandler类创建一个时间戳TCP服务器。

1. #!/usr/bin/env python
3. from SocketServer import (TCPServer as TCP,
4. StreamRequestHandler as SRH)
5. from time import ctime
7. HOST = ''
8. PORT = 21567
9. ADDR = (HOST, PORT)
11. class MyRequestHandler(SRH):
12. def handle(self):
13. print '...connected from:', self.client\_address
14. self.wfile.write('[%s] %s' % (ctime(),
15. self.rfile.readline()))
17. tcpServ = TCP(ADDR, MyRequestHandler)
18. tcpServ.allow\_reuse\_address = True
19. print 'waiting for connection...'
20. tcpServ.serve\_forever()

       从SocketServer 的StreamRequestHandler类中派生出一个子类，并重写handle()函数，在BaseRequest类中，这个函数什么也不做，def handle(self):pass。StreamRequestHandler类支持像操作文件对象那样操作输入输出套接字，可以用readline()函数得到客户消息，用write()函数把字符串发给客户。  
（2）创建SocketServerTCP客户端  
        客户端要做一些相应地调整以适应新的服务器。

1. #!/usr/bin/env python
3. from socket import \*
5. HOST = 'localhost'
6. PORT = 21567
7. BUFSIZ = 1024
8. ADDR = (HOST, PORT)
10. while True:
11. tcpCliSock = socket(AF\_INET, SOCK\_STREAM)
12. tcpCliSock.connect(ADDR)
13. data = raw\_input('**>** ')
14. if not data:
15. break
16. tcpCliSock.send('%s\r\n' % data)
17. data = tcpCliSock.recv(BUFSIZ)
18. if not data:
19. break
20. print data.strip()
21. tcpCliSock.close()

       SocketServer的请求处理器的默认行为是接受连接、得到请求、然后就关闭连接。这使得不能在程序的运行时，一直保持连接状态，要每次发送数据到服务器的时候都要创建一个新的套接字。不过，这种行为也可以通过重写请求处理器中相应的函数来改变。  
（3）执行SocketServerTCP服务器和客户端  
  
       输出与之前的TCP客户与服务器相似，但连了服务器三次，是一种短连接。

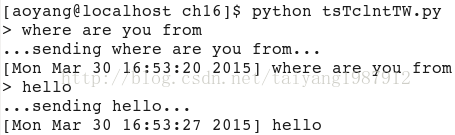
**4、Twisted 框架介绍**

       Twisted是一个完全事件驱动的网络框架，它允许使用和开发完全异步的网络应用程序和协议。它为创建一个完整系统提供了很大的帮助，系统中可以有：网络协议、线程、安全和认证、聊天/即时通讯、数据库管理、关系数据库集成、网页/互联网、电子邮件、命令行参数、图形界面集成等。Twisted  
        Twisted需要另外下载安装，或直接使用yum进行安装，#yum install python-twisted。  
        像SocketServer一样，Twisted的大部分功能都在它的类里面。例子中将使用Twisted的Internet组件中reactor和protocol包类。  
（1）创建一个Twisted Reactor TCP服务器  
        代码与SocketServer例子有些相似，先创建一个协议类并像安装回调函数那样重写几个函数，而不是写一个处理器类。同样的，它是异步的。

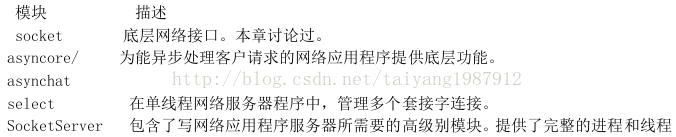
1. #!/usr/bin/env python
3. from twisted.internet import protocol, reactor
4. from time import ctime
6. PORT = 21567
8. class TSServProtocol(protocol.Protocol):
9. def connectionMade(self):
10. clnt = self.clnt = self.transport.getPeer().host
11. print '...connected from:', clnt
12. def dataReceived(self, data):
13. self.transport.write('[%s] %s' % (ctime(), data))
15. factory = protocol.Factory()
16. factory.protocol = TSServProtocol
17. print 'waiting for connection...'
18. reactor.listenTCP(PORT, factory)
19. reactor.run()

        从Protocol类中派生出TSServProtocol类做为时间戳服务器。然后重写connectionMade()函数，这个函数在有客户连接的时候被调用；以及重写dataReceived()函数，这个函数在客户通过网络发送数据过来时被调用。reactor把数据当成参数传到这个函数中，这样我们就不用自己去解析数据了。创建一个protocol工厂，每次有连接进来的时候，它都会“生产”一个protocol 对象；然后在reactor中安装一个TCP监听器以等待服务请求；当有请求进来时，创建一个TSServProtocol实例来服务那个客户。  
（2）创建一个Twisted Reactor TCP客户端  
       与SocketServer TCP客户不一样的是，这个例子与之前的客户端看上去不大一样，它是完全Twisted的。

1. #!/usr/bin/env python
3. from twisted.internet import protocol, reactor
5. HOST = 'localhost'
6. PORT = 21567
8. class TSClntProtocol(protocol.Protocol):
9. def sendData(self):
10. data = raw\_input('**>** ')
11. if data:
12. print '...sending %s...' % data
13. self.transport.write(data)
14. else:
15. self.transport.loseConnection()
17. def connectionMade(self):
18. self.sendData()
20. def dataReceived(self, data):
21. print data
22. self.sendData()
24. class TSClntFactory(protocol.ClientFactory):
25. protocol = TSClntProtocol
26. clientConnectionLost = clientConnectionFailed = \
27. lambda self, connector, reason: reactor.stop()
29. reactor.connectTCP(HOST, PORT, TSClntFactory())
30. reactor.run()

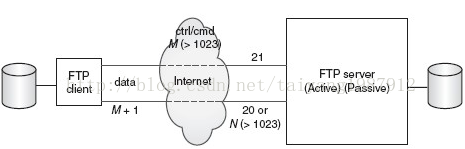
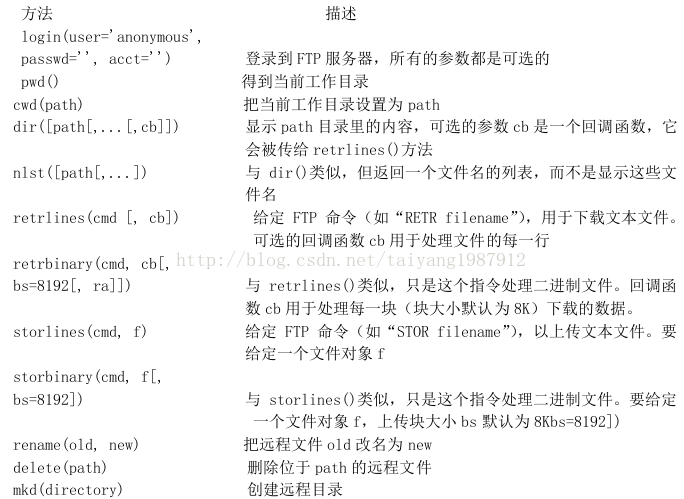
       由于我们现在是客户，所以我们要主动初始化跟服务器的对话。一旦连接建立好之后，我们先发送一个消息，服务器回复这个消息，我们把收到的回复显示在屏幕上，然后再发送其它消息给服务器。创建了一个客户工厂连接到服务器，然后运行reactor，在这里实例化了客户端工厂，而不是像在服务器里那样把它传到reactor中，因为，我们不是等待客户连接的服务器，服务器在有连接时要为每个连接创建一个新的protocol对象。我们只是一个客户，所以我们只要创建一个protocol对象，连接到服务器，服务器的工厂会创建一个 protocol 对象来与我们对话。  
（3）执行Twisted Reactor TCP服务器和客户端  
       服务器只有一个连接的情况，Twisted维护连接，不会在每个消息后都关闭。  
http://img.blog.csdn.net/20150330165342833?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdGFpeWFuZzE5ODc5MTI=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center   

**5、socket相关模块**

        网络/套接字编程相关模块：  
  
       select模块通常在底层套接字程序中与socket模块联合使用，它提供的 select()函数可以同时管理多个套接字对象，它最有用的功能就是同时监听多个套接字的连接。select()函数会阻塞，直到有至少一个套接字准备好要进行通讯的时候才退出，它提供了哪些套接字已经准备好可以开始读取的集合。  
        async\*和SocketServer模块在创建服务器方面都提供了高层次的功能。由于是基于socket和select 模块，封装了所有的底层的代码，它们使得你可以快速开发客户/服务器的系统。SocketServer甚至提供了把线程或进程集成到服务器中的功能，以实现更好的对客户请求的并行处理的能力。  
        虽然async\*是标准库提供的唯一的异步开发支持库。但我们也可选择如Twisted 这样的，相对标准库更现代、更强大的第三方库。Twisted提供了更为强大，更具弹性的框架，它已经实现了很多协议。

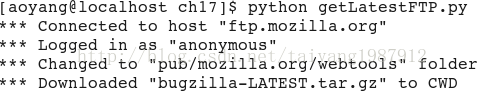
**6、文件传输协议FTP**

（1）FTP协议

       FTP要求输入用户名和密码才能访问远程的FTP服务器，但它也允许没有帐号的用户以匿名用户登录。匿名用户的用户名是  
“anonymous”，密码一般是用户的e-mail地址，匿名用户通过FTP协议可以使用的命令与一般的用户相比来说限制更多。一般,在客户端超过15分钟(900 秒)不活动之后，连接就会被关闭。  
        在底层上，FTP只使用TCP，它不使用UDP。而且FTP是客户端/服务器编程中很“与众不同”的例子。客户端和服务器都使用两个套接字来通讯：一个是控制和命令端口(21号端口)，另一个是数据端口(有时是20号端口)。  
  
        FTP有两种模式：主动和被动。只有在主动模式服务器才使用数据端口。在服务器把20号端口设置为数据端口后，它“主动”连接客户端的数据端口。而被动模式中，服务器只是告诉客户端它的随机端口的号码，客户端必须主动建立数据连接。在这种模式下，会看到FTP 服务器在建立数据连接时是“被动”的。  
（2）Python和FTP  
       使用Python的FTP支持时，需要导入ftplib模块，并实例化一个ftplib.FTP类对象，所有的 FTP 操作(如登录、传输文件和登出等)都要使用这个对象来完成。  
       ftplib.FTP的类方法：  
  
http://img.blog.csdn.net/20150330210624861?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdGFpeWFuZzE5ODc5MTI=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center  
       在一般的FTP通讯中，要使用到的指令有login()、cwd()、dir()、pwd()、stor\*()、retr\*()和quit()。  
（3）客户端FTP程序  
       FTP下载示例用于下载网站中最新版本的文件。

1. #!/usr/bin/env python
3. import ftplib
4. import os
5. import socket
7. HOST = 'ftp.mozilla.org'
8. DIRN = 'pub/mozilla.org/webtools'
9. FILE = 'bugzilla-LATEST.tar.gz'
11. def main():
12. try:
13. f = ftplib.FTP(HOST)
14. except (socket.error, socket.gaierror), e:
15. print 'ERROR: cannot reach "%s"' % HOST
16. return
17. print '\*\*\* Connected to host "%s"' % HOST
19. try:
20. f.login()
21. except ftplib.error\_perm:
22. print 'ERROR: cannot login anonymously'
23. f.quit()
24. return
25. print '\*\*\* Logged in as "anonymous"'
27. try:
28. f.cwd(DIRN)
29. except ftplib.error\_perm:
30. print 'ERROR: cannot CD to "%s" folder' % DIRN
31. f.quit()
32. return
33. print '\*\*\* Changed to "%s" folder' % DIRN
35. try:
36. f.retrbinary('RETR %s' % FILE,
37. open(FILE, 'wb').write)
38. except ftplib.error\_perm:
39. print 'ERROR: cannot read file "%s"' % FILE
40. if os.path.exists(FILE): os.unlink(FILE)
41. else:
42. print '\*\*\* Downloaded "%s" to CWD' % FILE
43. f.quit()
44. return
46. if \_\_name\_\_ == '\_\_main\_\_':
47. main()

      创建一个FTP对象，尝试连接到FTP服务器然后返回，在有任何错误发生的时候退出。尝试用“anonymous”登录，如果不行就结束，下一步就是转到发布目录，最后下载文件。



（4）FTP应用  
       Python同时支持主动和被动模式，Python2.1后被动模式支持默认是打开的。典型的 FTP 客户端类型：a、命令行客户端程序，使用一些FTP文件传输工具如/bin/ftp或NcFTP，它们允许用户在命令行交互式的参与到 FTP 通讯中来。b、GUI客户端程序，与命令行客户端程序相似，只是它是一个GUI程序， WsFTP和Fetch等。c、网页浏览器，在使用 HTTP 之外，大多数网页浏览器(也是一个客户端)可以进行 FTP 通讯。d、定制程序，自己写的用于FTP文件传输的程序，由于程序用于特殊目的，一般这种程序都不允许用户与服务器接触。  
        这四种客户端类型都可以用Python来写。用ftplib来创建了一个自己的定制程序，也可以自己做一个命令行的应用程序。在命令行的基础上，可以使用一些界面工具包如Tk、wxWidgets、GTK+、Qt、MFC、甚至Swing(要导入相应的 Python的接口模块)来创建一个完整的 GUI 程序。最后，还可以使用Python的urllib模块来解析FTP的URL并进行FTP传输，在urllib的内部也导入并使用的ftplib、urllib也是ftplib的客户端。  
（5）FTP服务器        
       由于FTP服务器的实现比较复杂，在标准库中没有包含此实现，如果需要开发FTP服务器端，可以使用Twisted框架。也可以使用FTP的服务器第三方开发包pyftplib（PythonFTPserverlibrary），它提供了高级的便携式的编程接口用来实现异步的FTP服务器的功能。

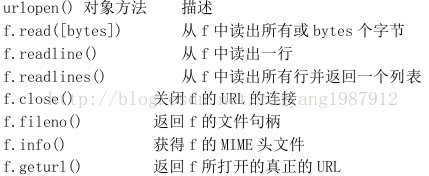
**7、Python的web应用**

        Web 客户端和服务器端交互使用的“语言” Web交互的标准协议是HTTP(超文本传输协议)，HTTP协议是TCP/IP协议的上层协议，这意味着HTTP协议依靠TCP/IP协议来进行低层的交流工作。HTTP的职责不是路由或者传递消息(TCP/IP 协议处理这些)，而是通过发送、接受 HTTP 消息来处理客户端的请求。简单的 Web应用包扩使用被称为URL(统一资源定位器)的 Web 地址。这个地址用来在Web上定位一个文档，或者调用一个CGI程序来为你的客户端产生一个文档。  
        Python支持两种不同的模块，分别以不同的功能和兼容性来处理URL。一种是urlparse，一种是urllib。  
（1）urlparse 模块  
       urlpasrse模块提供了操作URL字符串的基本功能，这些功能包括urlparse()、urlunparse()和urljoin()。语法结构：urlparse(urlstr, defProtSch=None, allowFrag=None)，urlparse()将urlstr解析成一个6元 组 (prot\_sch, net\_loc,path, params, query,frag)；如果urlstr中没有提供默认的网络协议或下载规划时可以使用defProtSch；allowFrag标识一个URL是否允许使用零部件；一个给定URL经urlparse()后的输出：

1. **>>>** import urlparse
2. **>>>** urlparse.urlparse('http://www.python.org/doc/FAQ.html')
3. ParseResult(scheme='http', netloc='www.python.org', path='/doc/FAQ.html', params='', query='', fragment='')
4. **>>>**

       urlparse.urlunparse()，它的功能与urlpase()完全相反，它拼合一个6元组(prot\_sch, net\_loc, path,params, query, frag)- urltup，它可能是一个URL经urlparse()后的输出返回值，语法结构：urlunparse(urltup)。  
       urlparse.urljoin()，在需要多个相关的URL时就需要使用urljoin()的功能了，如在一个Web页中生成的一系列页面的URL，语法：urljoin(baseurl, newurl, allowFrag=None)。

1. **>>>** import urlparse
2. **>>>** urlparse.urljoin('http://www.python.org/doc/FAQ.html','current/lib/lib.htm')
3. 'http://www.python.org/doc/current/lib/lib.htm'
4. **>>>**

（2）urllib 模块  
       urllib提供了一个高级的Web交流库，支持Web协议、HTTP、FTP和Gopher协议，同时也支持对本地文件的访问。urllib模块的特殊功能是利用上述协议下载数据(从因特网、局域网、主机上下载)。使用这个模块可以避免使用 httplib、ftplib和gopherlib这些模块。urllib模块提供了在给定的URL地址下载数据的功能，同时也可以通过字符串的编码、解码来确保它们是有效URL字符串的一部分。urllib包括urlopen()、urlretrieve()、quote()、unquote()、quote\_plus()、unquote\_plus()和urlencode() 。 可以使用urlopen()方法返回文件类型对象。  
       urllib.urlopen()，打开一个给定URL字符串与 Web 连接，并返回了文件类的对象。语法结构：rlopen(urlstr, postQueryData=None)，urlopen()打开urlstr所指向的URL，如果没有给定协议或者下载规划或者文件规划早已传入，urlopen()则会打开一个本地的文件。  
        连接成功urlopen()将会返回一个文件类型对象，就像在目标路径下打开了一个可读文件。例如文件对象是f，那么“句柄”将会支持可读方法如：f.read()、f.readline()、f.readlines()、f.close()和f.fileno()。  
  
      若打算访问更加复杂的URL或者想要处理更复杂的情况如基于数字的权限验证、重定位、coockie等问题，建议使用urllib2模块。  
      urllib.urlretrieve()语法：urlretrieve(urlstr, localfile=None, downloadStatusHook=None)，除了像urlopen()这样从URL中读取内容，urlretrieve()可以方便地将urlstr定位到的整个HTML文件下载到你本地的硬盘上。可以将下载后的数据存成一个本地文件或者一个临时文件，如果该文件已经被复制到本地或者已经是一个本地文件，后续的下载动作将不会发生。urlretrieve()返回一个2元组(filename, mime\_hdrs)，filename是包含下载数据的本地文件名，mime\_hdrs是对Web服务器响应后返回的一系列MIME文件头。  
         urllib.quote()和urllib.quote\_plus()，quote\*()函数获取URL数据，并将其编码从而适用于 URL 字符串中。尤其是一些不能被打印  
的或者不被Web服务器作为有效URL接收的特殊字符串必须被转换，这就是quote\*()函数的功能。quote\*()函数的语法如下：:quote(urldata, safe='/')，其中逗号、下划线、句号、斜线和字母数字这类符号是不需要转化，其他的则均需要转换；另外那些不被允许的字符前边会被加上百分号(%)同时转换成 16 进制。当调用quote\*()时，urldata字符串被转换成了一个可在 URL 字符串中  
使用的等价值，safe字符串可以包含一系列的不能被转换的字符。默认的是斜线(/)。quote\_plus()与quote()很像，另外quote\_plus()还可以将空格编码成+号。  
        urllib.unquote()和urllib.unquote\_plus()，unquote\*()函数与quote\*()函数的功能安全相反，它将所有编码为“%xx”式的字母都转换成它们的ASCII码值。unquote\*()的语法如下：unquote\*(urldata)，调用unquote()函 数将会把urldata中所有的URL-编码字母都解码，并返回字符串 ，unquote\_plus()函数会将加号转换成空格符。  
        urllib.urlencode()函数接收字典的键-值对，并将其编译成CGI请求的URL字符串的一部分。键值对的格式是“键=值”，以连接符(&)划分。更进一步，键和它们的值被传到quote\_plus()函数中进行适当的编码。

1. **>>>** import urllib
2. **>>>** name = 'joe mama'
3. **>>>** number = 6
4. **>>>** base = 'http://www/~foo/cgi-bin/s.py'
5. **>>>** final = '%s?name=%s&num=%d' % (base, name, number)
6. **>>>** final
7. 'http://www/~foo/cgi-bin/s.py?name=joe mama&num=6'
8. **>>>** urllib.quote(final)
9. 'http%3A//www/%7Efoo/cgi-bin/s.py%3Fname%3Djoe%20mama%26num%3D6'
10. **>>>** urllib.quote\_plus(final)
11. 'http%3A%2F%2Fwww%2F%7Efoo%2Fcgi-bin%2Fs.py%3Fname%3Djoe+mama%26num%3D6'
12. **>>>** aDict = { 'name': 'Georgina Garcia', 'hmdir': '~ggarcia' }
13. **>>>** urllib.urlencode(aDict)
14. 'name=Georgina+Garcia&hmdir=%7Eggarcia'
15. **>>>**

（3）urllib2模块  
        urllib2可以处理更复杂URL的打开问题。建立基本认证(登录名和密码)需求的 Web 站点，urllib2可以通过两种不同的方式来解决这个问题：一是，可以建立一个基础认证处理器(urllib2.HTTPBasicAuthHandler)，同时在基本URL或域上注册一个登录密码，这就意味着在Web站点上定义了个安全区域，一旦完成这些，可以安装URL打开，通过这个处理器打开所有的 URL；二是，当浏览器提示的时候,输入用户名和密码，这样就发送了一个带有适当用户请求的认证头。  
（4）网络爬虫

        高级Web客户端的一个例子就是网络爬虫(aka 蜘蛛和机器人)，这些程序可以基于不同目的在因特网上探索和下载页面，其中包括：a、为Google和Yahoo这类大型的搜索引擎建索引；b、脱机浏览，将文档下载到本地，重新设定超链接，为本地浏览器创建镜像；c、下载并保存历史记录或框架；d、Web页的缓存，节省再次访问Web站点的下载时间。  
       该程序包括两个类：一个管理整个crawling进程（Crawler类），和一个检索并解析每一个下载的Web页面（Retriever类）。

1. #!/usr/bin/env python
3. from sys import argv
4. from os import makedirs, unlink, sep
5. from os.path import isdir, exists, dirname, splitext
6. from string import replace, find, lower
7. from htmllib import HTMLParser
8. from urllib import urlretrieve
9. from urlparse import urlparse, urljoin
10. from formatter import DumbWriter, AbstractFormatter
11. from cStringIO import StringIO
13. class Retriever(object):    # download Web pages
15. def \_\_init\_\_(self, url):
16. self.url = url
17. self.file = self.filename(url)
19. def filename(self, url, deffile='index.htm'):
20. parsedurl = urlparse(url, 'http:', 0)  # parse path
21. path = parsedurl[1] + parsedurl[2]
22. ext = splitext(path)
23. if ext[1] == '':
24. if path[-1] == '/':
25. path += deffile
26. else:
27. path += '/' + deffile
28. ldir = dirname(path)    # local directory
29. if sep != '/':      # os-indep. path separator
30. ldir = replace(ldir, ',', sep)
31. if not isdir(ldir):      # create archive dir if nec.
32. if exists(ldir): unlink(ldir)
33. makedirs(ldir)
34. return path
36. def download(self):     # download Web page
37. try:
38. retval = urllib.urlretrieve(self.url, self.file)
39. except IOError:
40. retval = ('\*\*\* ERROR: invalid URL "%s"' % \
41. self.url, )
42. return retval
44. def parseAndGetLinks(self): # pars HTML, save links
45. self.parser = HTMLParser(AbstractFormatter( \
46. DumbWriter(StringIO())))
47. self.parser.feed(open(self.file).read())
48. self.parser.close()
49. return self.parse.anchorlist
51. class Crawler(object):      # manage entire crawling process
53. count = 0           # static downloaded page counter
55. def \_\_init\_\_(self, url):
56. self.q = [url]
57. self.seen = []
58. self.dom = urlparse(url)[1]
60. def getPage(self, url):
61. r = Retriever(url)
62. retval = r.download()
63. if retval[0] == '\*':     # error situation, do not parse
64. print retval, '... skipping parse'
65. return
66. Crawler.count = Crawler.count + 1
67. print '\n(', Crawler.count, ')'
68. print 'URL:', url
69. print 'FILE:', retval[0]
70. self.seen.append(url)
72. links = r.parseAndGetLinks()  # get and process links
73. for eachLink in links:
74. if eachLink[:4] != 'http' and \
75. find(eachLink, '://') == -1:
76. eachLink = urljoin(url, eachLink)
77. print '\* ', eachLink,
79. if find(lower(eachLink), 'mailto:') != -1:
80. print '... discarded, mailto link'
81. continue
83. if eachLink not in self.seen:
84. if find(eachLink, self.dom) == -1:
85. print '... discarded, not in domain'
86. else:
87. if eachLink not in self.q:
88. self.q.append(eachLink)
89. print '... new, added to Q'
90. else:
91. print '... discarded, already in Q'
92. else:
93. print '... discarded, already processed'
95. def go(self):                # process links in queue
96. while self.q:
97. url = self.q.pop()
98. self.getPage(url)
100. def main():
101. if len(argv) **>** 1:
102. url = argv[1]
103. else:
104. try:
105. url = raw\_input('Enter starting URL: ')
106. except (KeyboardInterrupt, EOFError):
107. url = ''
109. if not url: return
110. robot = Crawler(url)
111. robot.go()
113. if \_\_name\_\_ == '\_\_main\_\_':
114. main()

        网络爬虫抓取Web的开始页面地址(URL)，下载该页面和其它后续链接页面，但仅限于那些与开始页面有着相同域名的页面，如果没有这个限制的话，硬盘将会被耗尽!  
 （5）下载博客文章的简单网络爬虫  
        获取到HTML地址的单个页面：

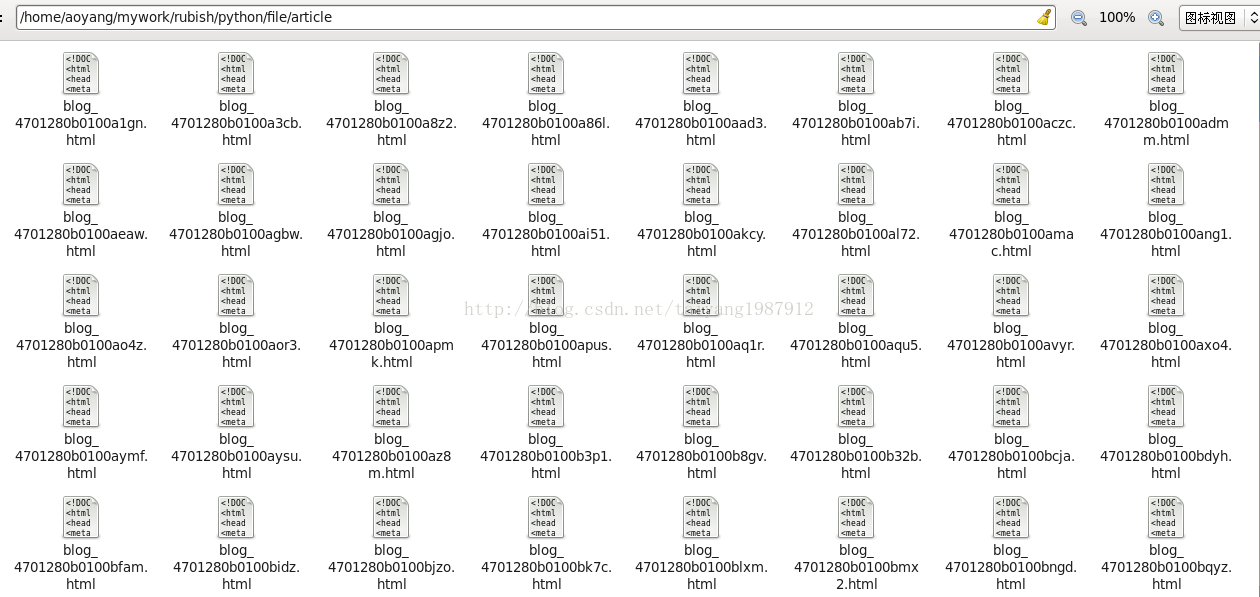
1. #coding:utf-8
2. import urllib
3. strHtml='<a title="地震思考录" target="\_blank" href="http://blog.sina.com.cn/s/blog\_4701280b0102egl0.html">地震思考录</a>'
4. title = strHtml.find(r'**<a** title')
5. href = strHtml.find(r'href=')
6. html = strHtml.find(r'.html')
7. url = strHtml[href + 6:html + 5]
9. content = urllib.urlopen(url).read()
11. filename = url[-26:]
12. #print filename
13. open(filename, 'w').write(content)

          下载单页HTML列表下的文章：

1. #coding:utf-8
2. import urllib
4. url=['']\*100
5. i = 0
6. address='http://blog.sina.com.cn/s/articlelist\_1191258123\_0\_1.html'
7. con = urllib.urlopen(address).read()
8. title = con.find(r'**<a** title=')
9. href = con.find(r'href=', title)
10. html = con.find(r'.html', href)
12. while title != -1 and href != -1 and html != -1 and i **<** **100:**
13. url[i] = con[href + 6:html + 5]
14. content = urllib.urlopen(url[i]).read()
15. print "downloading:", url[i]
16. open(r'article/' + url[i][-26:], "w+").write(content)
18. title = con.find(r'**<a** title=', html)
19. href = con.find(r'href=', title)
20. html = con.find(r'.html', href)
21. i = i + 1
22. else:
23. print 'download article finished!'

        下载多页HTML列表的所有文章：

1. #coding:utf-8
2. import urllib
4. url = ['']\*350
5. page = 1
6. number = 1;
7. address = 'http://blog.sina.com.cn/s/articlelist\_1191258123\_0\_%d.html'
8. while page **<**= 7:
9. print address % page
10. con = urllib.urlopen(address % page).read()
11. title = con.find(r'**<a** title=')
12. href = con.find(r'href=', title)
13. html = con.find(r'.html', href)
15. i = 0
16. while title != -1 and href != -1 and html != -1 and i **<** **50:**
17. url[i] = con[href + 6:html + 5]
18. content = urllib.urlopen(url[i]).read()
19. open(r'article/' + url[i][-26:], "w+").write(content)
20. print "%d page, %d number:downloading %s" % (page, number, url[i])
21. #time.sleep(1)
22. i = i + 1
23. title = con.find(r'**<a** title=', html)
24. href = con.find(r'href=', title)
25. html = con.find(r'.html', href)
26. number = number + 1
27. else:
28. print 'download %d page article finished!' % page
29. page = page + 1
30. else:
31. print "download all files!"
32. print "files number:", number - 1

运行结果：  
  
         article目录下的下载文件：  


**三、总结**

（1）Twisted框架功能强大，可以非常方便的实现很多复杂的功能，可参看其他的相关文档。  
（2）Python的网络编程可以异步通信，包括Fork方式、线程方式、异步I/O方式（select和poll）、asyncore模板。  
（3）Python在其他web和图形化界面及数据库方面的应用暂不涉及，可根据以后的需求更新相应的代码。

顶