**网络通信**

**IP地址**：用来在网络中标记一台电脑，在本地局域网上时唯一的，不可重复。

**IP地址的分类：**每一个IP地址包括两部分，**网络地址和主机地址**。

**端口：**与程序之间进行沟通的通道。

**端口号：**端口是通过端口号来标记的，端口号只有整数，范围是0~65535

**端口：**

**知名端口：**知名端口是众所周知的端口号，范围从**0到1023。**

88端口分配给HTTP服务

21端口分配给FTP服务

**动态端口：**一般不固定分配某种服务，而是动态分配，范围是**1024到65535。**

python3 hello.py：用python3运行hello.py文件

python3:打开python3的交互模式

ipython3：比python3高端的一种交互模式。

**socket简介**

**socket（简称：套接字）是进程间通信的一种方式**，它与其他进程间通信的一个主要不同是：它**能实现不同主机进程间通信**，网络上大多数的服务都是基于socket来完成通信的。

**创建socket**

在Python中使用socket模块的函数socket就可以完成：

**import socket**

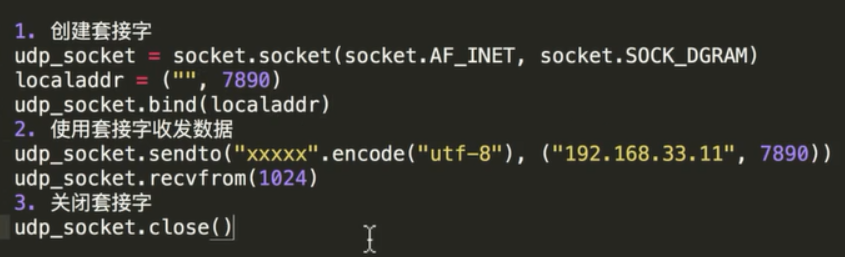
**socket.socket(AddressFamily, Type)**

AddressFamily:协议族（IPV4/IPV6）

Type：类型（TCP/UDP等）

**发送的数据是字符串时，在引号前加上一个小写的b**

**或者在发送的数据后面加上.encode(“utf-8”)**



如果自己给自己的电脑发送消息，可以在目标IP中输入**回环地址**。

**单工：**只能单向传递。例如：收音机

**半双工：**同一时刻只能单向传递。例如：对讲机

**全双工：**双向传递。例如：手机。  **socket套接字属于全双工。**

**发送数据的流程：**

1.创建套接字

2.发送数据

3.关闭套接字

**接收数据的流程：**  
1.创建套接字

2.绑定本地信息

3.接收数据

4.关闭套接字

**TCP**

**TCP协议**：传输控制协议是一种**面向连接的、可靠的、基于字节流**的传输层通信协议。

TCP通信需要经过**创建连接、数据传送、终止连接**三个步骤。

可靠传输：  
1.TCP采用发送应答机制

2.超时重传

3.错误校验

4.流量控制和阻塞管理

**TCP与UDP的不同点**

1.面向连接

2.有序数据传输

3.重发丢失的数据包

4.舍弃重复的数据包

5.无差错的数据传输

6.阻塞/流量控制

**TCP客户端流程**

1.创建套接字

2.连接服务器

3.发送/接收数据

4.关闭套接字



**TCP服务器流程**

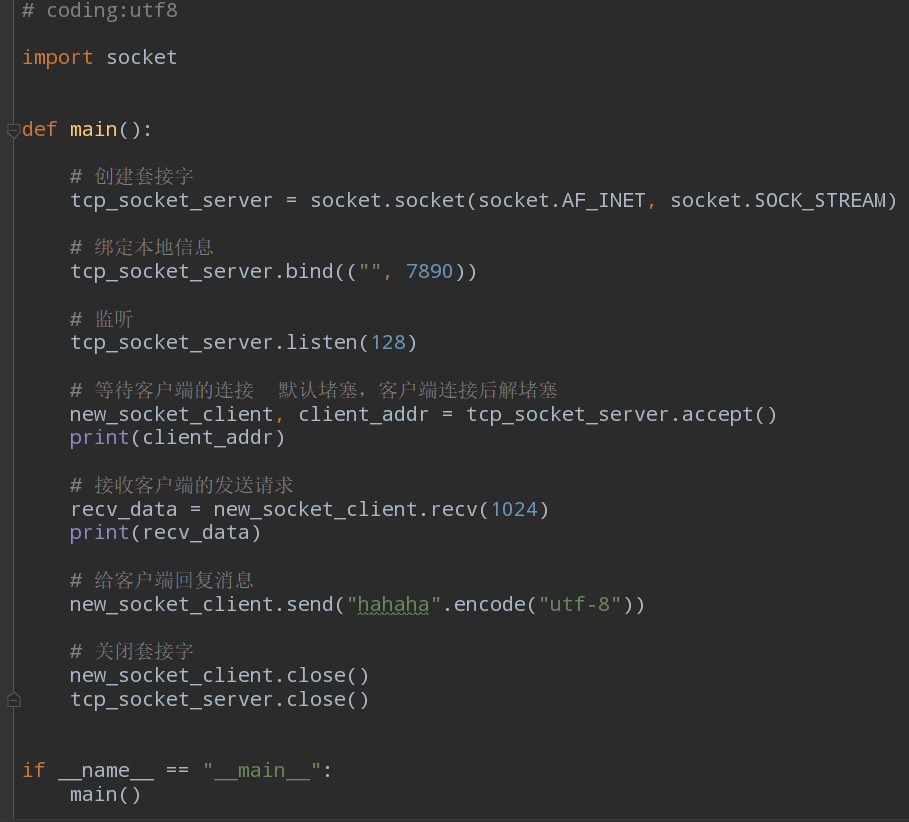
1.创建套接字

2.bind绑定ip和端口

3.listen使套接字变成可以被动连接

4.accept等待客户端连接

5.recv/send接收发送数据



监听套接字负责等待新的客户端进行连接

accept产生的新的套接字用来为已连接的客户端服务

**TCP注意点**

1.tcp服务器一般情况下都需要绑定，否则客户端找不到这个服务器。

2.tcp客户端一般不绑定，因为是主动链接服务器的，所以只要确定好服务器的ip/port等信息就可以了，本地客户端可以随机。

3.tcp服务器通过listen可以将socket创建出来的主动套接字变成被动的，这是做tcp服务器必须要做的。

4.当客户端需要链接服务器时，就需要使用connect进行连接，udp是不需要连接的而是直接发送，但是tcp必须先链接，只有链接成功才能通信。

5.当一个tcp客户端链接服务器时，服务器端会有一个新的套接字，这个套接字用来标记客户端，单独为这个客户端服务。

6.listen后的套接字是被动套接字，用来接收新的客户端的链接请求，而accept返回的新的套接字是标记这个新的客户端的。

7.关闭listen后的套接字意味着被动套接字关闭了，会导致新的客户端不能连接服务器，但是之前已经连接成功的客户端能正常通信。

8.关闭accept返回的套接字意味着这个客户端已经服务完毕。

9.当客户端的套接字调用close后，服务器端会recv解堵塞，并且返回的长度为0，因此服务器可以通过返回数据的长度来区别这个客户端是否已经下线。

**多任务**

**线程与进程**

多任务：多个任务同时进行。

并行：真的多任务。

并发：假的多任务。

查看线程数量 len(threading.enumerate())

**线程**

调用start时，线程开始执行，此时线程才被创建。

**调用函数时**，t = threading.Thread(target=函数名)

**调用类时**，t = 类名()，而且类要继承thread，即class 类名(threading.Thread)，而且在类中必须定义run方法。

class MyTheard(threading.Thread):

def run():

pass

在threading中，**target**指定将来这个线程**去哪个函数**执行代码，**args**指定将来调用函数的时候**传递什么数据**进去，而且args中指定的是**元祖。**

**同步**

同步就是协同步调，按预定的先后顺序运行。

**互斥锁**

当多个线程几乎同时修改某一个共享数据的时候，需要进行同步控制。

线程同步能够保证多个线程安全访问竞争资源，最简单的同步机制就是互斥锁。

互斥锁为资源引入一个状态：锁定/非锁定。

创建锁：mutex = threading.Lock()

锁定：mutex.acquire()

释放：mutex.release()

**死锁**

在线程间共享多个资源的时候，如果两个线程分别占有一部分资源而且同时等待对方的资源，就会造成死锁。

**避免死锁**

1.程序设计时尽量避免（银行家算法）

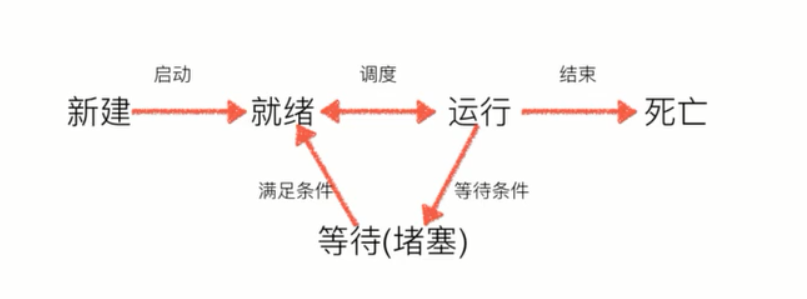
2.添加超时时间等。

**进程**

**程序：**程序是一个静态的。

**进程：**一个程序运行起来后，**代码+用到的资源**称之为进程，它是操作系统分配资源的**基本单位**。

**进程状态**



就绪态：运行的条件都已经就绪，正在等待cpu执行。

执行态：cpu正在执行其功能。

等待态：等待某些条件满足，例如一个程序sleep了，此时就处于等待态。

**进程、线程的关系**

**先有进程，再有线程。**

线程不能够独立执行，必须依存在进程中。

线程和进程在使用上各有优缺点，线程执行开销小，但不利于资源的保护和管理，而进程刚好相反。

进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大的提高了程序的运行效率。

**队列**

队列：先进先出

使用queue的一个目的是为了解耦合，另外也是为了完成进程间的数据共享。

存数据:q.put()

取数据:q.get() q.get\_nowait()

判断是否是空队列:q.empty()

判断是否是满队列:q.full()

**栈**

栈：先进后出

**进程池**

要使用if \_\_name\_\_ == “\_\_main\_\_”:

pool = threading.Pool(3)

pool.close()

pool.join()

**迭代器**

迭代是访问集合元素的一种方式，迭代器是一个可以记住自己位置的对象，迭代器对象从集合的第一个元素开始访问，到最后一个元素结束。迭代器只能往前不能后退。

迭代器存储的是生成数据的方式，而不是数据结果。

判断是否可迭代 isintance(对象, Iterrable)

具有iter和next方法的对象叫做迭代器。

range和xrange的区别

range返回的是生成数据的结果，xrange返回的是生成数据的方式。

并不是只有for循环能接收可迭代对象，还有list、tuple等也能接收。

**生成器**

生成器是一种特殊的迭代器。

含有yield语句的就是生成器

yield关键字有两点作用：

1.保存当前运行状态（断点），然后暂停执行，即将生成器（函数）挂起

2.将yield关键字后面表达式的值作为返回值返回，此时可以理解为起到了return的作用

可以使用next()函数让生成器从断点处继续执行，即唤醒生成器（函数）

send传递的数据是yield的结果。

一般send不用在第一次启动生成器过程中，一般放在next后面。

如果非要第一次启动就用send，可以使用send（None ）

monkey.patch\_all()将程序中的耗时操作的代码内部转化为gevent.sleep()

gevent.joinall()把创建的协程对象放在这个括号中，以列表的格式存放。

简单总结：  
1.进程是资源分配的单位。

2.线程是操作系统调度的单位。

3.进程切换需要的资源很大，效率很低。

4.线程切换需要的资源一般，效率一般（不考虑GIL的情况下）

5.协程切换需要的资源很小，效率高

6.多进程、多线程根据CPU核数不一样可能是并行的，但是协程是在一个线程中，所以是并发。

**Python高级语法**

**GIL锁**

GIL（全局解释器锁）：每个线程在执行的过程都需要先获取GIL，保证同一时刻只有一个线程可以执行代码。只对线程有影响，对进程、协程没有影响。

计算密集型：没有延时。进程

IO（读写）密集型：线程、协程。

怎么解决GIL的问题：

1.更换解释器，避免使用C语言解释器。

2.用其他的语言来替代要做的事情。

**深拷贝、浅拷贝**

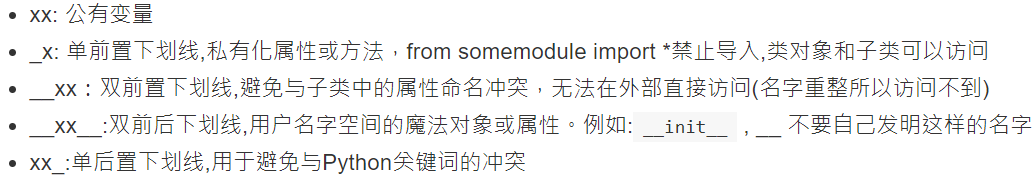
浅拷贝：浅拷贝是对于一个对象的顶层拷贝。即：拷贝了引用，并没有拷贝内容。

浅拷贝对于不可变类型，不会拷贝，仅仅是指向。

深拷贝：深拷贝是一个整个独立的对象拷贝，拷贝了所有属性，并拷贝属性指向的动态分配的内存。

浅拷贝:copy.copy 深拷贝：copy.deepcopy

**私有化**



重新导入模块： from imp import reload

为什么要使用继承？

1.能够提升代码的重用率，即开发一个类，可以在多个子功能中直接使用

2.继承能够有效的进行代码的管理，当某个类有问题只要修改这个类就行，而其继承这个类的子类往往不需要就修改

重写：子类与父类中有同名的函数，子类调用时会覆盖父类的方法，叫做重写。

**继承注意事项：**

1.super().\_\_init\_\_相对于类名.\_\_init\_\_，在单继承上用法基本无差

2.但在多继承上有区别，super方法能保证每个父类的方法只会执行一次，而使用类名的方法会导致方法被执行多次，具体看前面的输出结果

3.多继承时，使用super方法，对父类的传参数，应该是由于python中super的算法导致的原因，必须把参数全部传递，否则会报错

4.单继承时，使用super方法，则不能全部传递，只能传父类方法所需的参数，否则会报错

5.多继承时，相对于使用类名.\_\_init\_\_方法，要把每个父类全部写一遍, 而使用super方法，只需写一句话便执行了全部父类的方法，这也是为何多继承需要全部传参的一个原因。

上下文管理器

任何实现了\_\_enter\_\_方法和\_\_exit\_\_方法的对象都可以成为上下文管理器。上下文管理器对象可以使用with关键字，文件对象也实现了上下文管理器。

**Web服务器**

**正则表达式**

**需要导入re模块**

**import re**

**匹配单个字符**



\w也匹配中文

**匹配多个字符**



**匹配开头和结尾**



**括号代表分组，第一个分组是group(1)，以此类推。**

**如果跟group(1)中的内容是一样的话，那么可以直接在正则表达式中该位置写上数字1.**

**匹配分组**



**re模块的高级用法**

search：不用从头开始匹配，没有group分组的功能。

如果想要search达到match的功能，可以在正则表达式开头加上^符。

findall：不需要使用group，返回的是一个列表。

sub：将匹配到的数据进行替换。

split：根据匹配进行切割字符串，并返回一个列表。

**HTTP协议**

HTML是一种用来定义网页的文本，可以编写网页。

HTTP是网络上传输HTML的协议，用于浏览器和服务器之间的通信，基于TCP。

http:超文本传输协议。

TCP三次握手、四次挥手：建立连接三次握手，断开连接四次挥手。

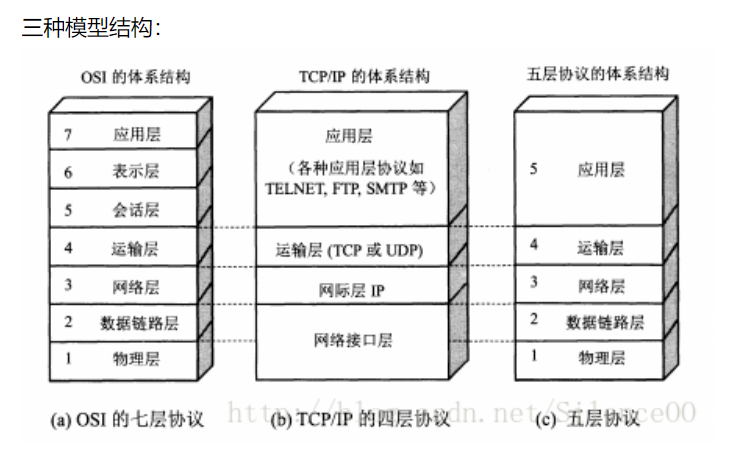
request header类型是GET时，一般没有request body，类型是POST时，一般都有request body。

response header和response body之间有一个空行。

**tcp\_socket\_server.setblocking(False)**  设置套接字为非堵塞的方式

epoll：共享内存，事件通知的方式。

TCP/IP协议：是一种用于浏览器与服务器之间进行数据传输的一种通信方式。



集线器（hub）：只支持多播。

交换机（switch）：支持多波及单播。

通用mac地址：FF:FF:FF:FF:FF:FF

路由器（router）：链接两个及以上的网络，让他们之间可以通信。

网关：具有数据转发功能的设备。

IP、网络掩码：可以确定ip是哪种类型的。使用按位与操作。

按位与操作：都为1才是1，只要有一个0，就为0。

DNS服务器：专门用来解析域名对应的IP地址的服务器。

子网掩码与交换机有关，网关与路由器有关。

**Mini\_web框架**

**WSGI**

文件中都需要有一个application对象，server端会找到这个文件，然后调用这个对象。

application对象是唯一的入口，所以不管客户端请求的路径和数据是什么，server都是调用这个application对象，具体的客户端请求的处理有application对象完成。

def application(environ, start\_response):

pass

environ参数是一个Python的字典，里面存放了所有和客户端相关的信息，这样application对象就能知道客户端请求的资源是什么，请求中带了什么数据等。

start\_response是一个可调用对象，接收两个必选参数和一个可选参数。

必选参数：

status:一个字符串，表示HTTP响应状态字符串。

response\_header:一个列表，包含如下形式的元祖(header\_name,header\_value)

用来表示HTTP响应的headers。

可选参数：

exc\_info:用于出错时，server需要返回给浏览器的信息。

**WSGI流程**

1.定义application方法

start\_response(服务器的配置信息)

def application(environ, start\_response):

pass

2.在服务器脚本中调用application方法，定义set\_response\_header方法（保存response\_header信息）

body = mini\_frame.application(environ, self.set\_response\_header)

def set\_response\_header(self, status, response\_header):

3.根据浏览器的请求返回相应的页面数据

pass--->file\_name = environ[“PATH\_INFO”]

if file\_name == “/index.py”:

return index()

.....

**闭包**

闭包的定义：函数中嵌套着另外一个函数，里面的函数需要使用到外部的函数的变量，这样的函数称之为闭包。

思考：函数、匿名函数、闭包、对象 当做实参时 有什么区别？  
1. 匿名函数能够完成基本的简单功能，传递是这个函数的引用 只有功能。  
2. 普通函数能够完成较为复杂的功能，传递是这个函数的引用 只有功能。  
3. 闭包能够将较为复杂的功能，传递是这个闭包中的函数以及数据，因此传递是功能+数据。  
4. 对象能够完成最为复杂的功能，传递是很多数据+很多功能，因此传递是功能+数据。