### 什么是socket？

 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：它能实现不同主机间的进程间通信，socket属于全双工。

关于创建套接字的参数说明：

socket.socket(AddressFamily, Type)

Address Family：可以选择 AF\_INET（用于 Internet 进程间通信） 或者 AF\_UNIX（用于同一台机器进程间通信）,实际工作中常用AF\_INET

Type：套接字类型，可以是 SOCK\_STREAM（流式套接字，主要用于 TCP 协议）或者 SOCK\_DGRAM（数据报套接字，主要用于 UDP 协议）

### 网络udp概述

UDP 是User Datagram Protocol的简称， 中文名是用户数据报协议。一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。类似于日常生活中写信的场景，不需要建立链接，不关心发送结果。

### 网络udp通信实现步骤：

# 1、创建套接字

Udp\_socket = Socket.socket(socket. AF\_INET, socket.SOCK\_DGRAM)

# 2、绑定端口 作为客户端udp一般不绑定端口号

Udp\_socket.bind((“”, 7788)) # ip地址不写，默认本机任何一个ip

# 3、发送数据

Socket.sendto(‘hello python’.encode(‘utf-8’), (‘192.168.21.66’, 7788))

#4、接收数据

Recv\_data = Socket.recvfrom(1024) # 返回元组信息包含数据和发送方的ip以及端口

# 5、关闭套接字

Socket.close()

### Encode()和decode()函数的用法?

Str.Encode(‘utf-8’),把字符串编码成字节码，utf-8代表的是国际码，gbk代表的是汉字码

Bytes.decode(‘utf-8’),把字节码解码成字符串。

### 网络TCP概述

TCP协议，传输控制协议（英语：Transmission Control Protocol，缩写为 TCP）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。类似于日常生活中打电话的场景，需要建立链接，确认发送结果，安全传输。

TCP严格区分客户端和服务端，服务端一般都需要绑定端口，否则客户端找不到该服务器，客户端一般不绑定端口，如果客户端绑定端口就会出现端口冲突导致无法多开的问题。

### TCP相比UDP有哪些优势？（不同）

面向连接（确认有创建三方交握，连接已创建才作传输。）；有序数据传输；重发丢失的数据包；舍弃重复的数据包；差错的数据传输；阻塞/流量控制。

### 网络TCP通信实现步骤

#### TCP客户端：

# 创建套接字

Tcp\_client\_socket = socket.socket(socket.SOCK\_AF\_INET, socket.SOCK\_STREAM)

# 链接服务器

Tcp\_client\_socket.connect((‘server\_ip’, server\_port))

# 向服务器发送数据

tcp\_client\_socket.send(‘hello’.encode(‘utf-8’))

# 接收服务器回送数据

Recv\_data = Tcp\_cilent\_socket.recv(1024) # 返回的是字节类型的数据

# 关闭套接字

Tcp\_client\_socket.close()

#### TCP服务端：

# 创建套接字

Tcp\_sever\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# 绑定端口信息，一定要做

Tcp\_server\_socket.bind((“”, 7788))

# 把监听由主动变为被动

Tcp\_server\_socket.listen(128)

# 等待客户端链接，在链接之前处于堵塞状态，一直到客户端链接才会解堵塞

# 返回一个新的套接字专为客户端服务、以及客户端的ip、port

New\_client\_socket, client\_addr = Tcp\_server\_socket.accept()

# 等待客户端发送数据

Recv\_data = new\_client\_socket.recv(1024).decode(‘utf-8’)

# 可以给客户端回送数据

New\_client\_socket.send(‘hello’.encode(‘utf-8’))

# 为该客户端服务完毕之后关闭套接字

New\_client\_socket.close()

# 关闭服务器端套接字

Tcp\_server\_socket.close()

#### Listen(backlog)数值代表的含义？

监听变主动为被动，backlog指定最多允许多少个客户连接到服务器。它的值至少为1。收到连接请求后，这些请求需要排队，如果队列满，就拒绝请求。请注意，最大限制是取决于操作系统，Ubuntu上默认是128

#### TCP服务端recv解堵塞的两种方式？

1、收到客户端发送的数据

2、客户端关闭了套接字

可以通过判断收到数据的长度来判断客户端是否下线，长度为0代表已下线。

### 什么是多任务？

简单地说，就是操作系统可以同时运行多个任务。实现多任务有多种方式，线程、进程、协程。

### 并行和并发的区别？

并发：指的是任务数多余cpu核数，通过操作系统的各种任务调度算法，实现用多个任务“一起”执行（实际上总有一些任务不在执行，因为切换任务的速度相当快，看上去一起执行而已）

并行：指的是任务数小于等于cpu核数，即任务真的是一起执行的

真的多任务叫并行，假的多任务叫并发。

### 什么是线程？

可以简单理解为同一进程中有多个计数器，每个线程的执行时间不确定，而每个进程的时间片相等，线程是操作系统调度执行的最小单位

### 线程的创建步骤

Import threading

# 创建一个线程对象

t1 = threading.Thread(target=func\_name, args=(num,), name=”子线程名字”)

# 创建一个线程并启动

t1.start()

# 等待子线程执行完毕之后再继续向下执行主线程

t1.join()

备注：**主线程会等待子线程结束之后才会结束，主线程一死，子线程也会死。线程的调度是随机的，并没有先后顺序**。

### 枚举函数enumerate的用法

enumerate()是python的内置函数，在字典上是枚举、列举的意思。用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。通过threading.enumerate()就可以获取线程列表。

### 互斥锁的运用

由于多线程之间共享全局变量就会导致出现资源竞争的问题，为了避免这种竞争出现，利用互斥锁可以实现线程同步。

# 创建锁

Mutex = threading.Lock()

# 加锁

Mutex.acquire()

# 释放锁

Mutex.release()

### 什么是死锁？

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并且同时等待对方的资源，就会造成死锁，尽管死锁很少发生，但一旦发生就会造成应用的停止响应。

### 什么是进程？

程序是静态的，当程序运行起来就叫做进程。是操作系统分配资源的基本单元。

### 进程、线程的区别与优缺点

#### 1. 定义的不同：

进程是系统进行资源分配的最小单位.

线程是进程的一个实体,是CPU进行调度的基本单位，它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

#### 2. 区别：

一个程序至少有一个进程,一个进程至少有一个线程.线程的划分尺度小于进程(资源比进程少)，使得多线程程序的并发性高。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，从而极大地提高了程序的运行效率。线程不能够独立执行，必须依存在进程中

#### 3. 优缺点：

线程执行开销小，但不利于资源的管理和保护；而进程正相反。

### 进程的创建步骤

# 导入进程模块

Import multiprocessing

# 创建一个进程的实力对象

P = Multiprocessing.Process(target=func\_name[,args=(元组), kwargs={字典}])

# 创建并启动进程

p.start()

p.join(5)

### 通过Queue实现进程间通信

# 创建一个队列

Q = multiprocess.Queue(3) # 3代表队列中最多可以接收三条消息

# 向队列中添加数据

q.put(data)

# 从队列中取数据

q.get()

# 判断队列中是否为空

q.empty()

# 判断队列中是否已满

q.full()

注意：**如果是通过进程池创建的进程，那么队列的使用要用multiprocessing.Manager().Queue()的方式，否则会报错**。

### 进程池Pool

当需要创建的子进程数量不多时，可以直接利用multiprocessing中的Process动态成生多个进程，但如果是上百甚至上千个目标，手动的去创建进程的工作量巨大，此时就可以用到multiprocessing模块提供的Pool方法。

初始化Pool时，可以指定一个最大进程数，当有新的请求提交到Pool中时，如果池还没有满，那么就会创建一个新的进程用来执行该请求；但如果池中的进程数已经达到指定的最大值，那么该请求就会等待，直到池中有进程结束，才会用之前的进程来执行新的任务

### 进程池的实现步骤

# 导入进程池模块

From multiprocessing import Pool

# 定义进程池，最大进程池最大数

Po = Pool(3)

# 通过进程池调用目标 apply\_async非阻塞，不会等待子进程结束；apply阻塞，会等待子进程结束才结束

po.apply\_async(要调用的目标,(传递给目标的参数元祖,))

# 关闭进程池

Po.close()

# 等待进程池执行完毕

Po.join()

### 什么是协程？

协程，又称微线程，纤程。英文名Coroutine。协程，利用线程在等待某个资源的期间执行其他函数，切换资源消耗非常小，协程效率相当快。

### 进程、线程、协程的区别？

进程、线程、协程的区别

进程是资源分配的单位，真正执行代码的是线程，操作系统真正调度的是线程。

进程没有线程效率高，进程占用资源多，线程占用资源少，比线程更少的是协程。

协程依赖于线程、线程依赖于进程，进程一死线程必挂，线程一挂协程必死

一般不用多进程，可以考虑使用多线程，如果多线程里面有很多网络请求，网络可能会有堵塞，此时用协程比较合适。

### 什么是GIL？

GIL并不是Python的特性，Python完全可以不依赖于GIL。GIL全称Global Interpreter Lock。它是在实现Python解析器(CPython)时所引入的一个概念。GIL无疑就是一把全局排他锁。

Python GIL其实是功能和性能之间权衡后的产物，它尤其存在的合理性，也有较难改变的客观因素。因为GIL的存在，只有IO Bound场景下得多线程会得到较好的性能，如果对并行计算性能较高的程序可以考虑把核心部分也成C模块，或者索性用其他语言实现

GIL在较长一段时间内将会继续存在，但是会不断对其进行改进。