都是多任务编程

一个进程包含多个线程

都是动态的占有资源的，线程共享进程的资源

进程比线程消耗资源更多

进程空间独立使用特定的IPC，线程使用全局变量

2服务器模型

循环模型：同一时刻只能处理一个请求

并发模型：IO并发，多进程、多线程并发：任何任务

基于fork的多进程并发程序

每当有一个客户连接

多线程并发

threading的多线程并发

对比多进程并发

\*消耗资源较少

\*线程应该 更注意共享资源的 操作

\*在python中应该注意GIL问题，网络延迟较高，线程并发也是一种可行的办法

实现步骤

1. 创建套接字，绑定监听
2. 接收客户端请求，创建新的线程
3. 主线程继续写接收其他客户端连接
4. 分支线程启动对应的函数处理客户端请求
5. 当客户端断开，则分支线程结束

cookie

import traceback

功能：更详细的打印异常信息

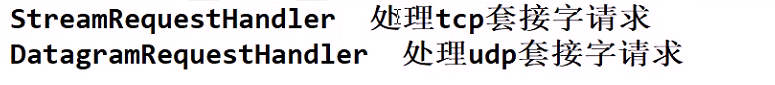
traceback.print\_exc()

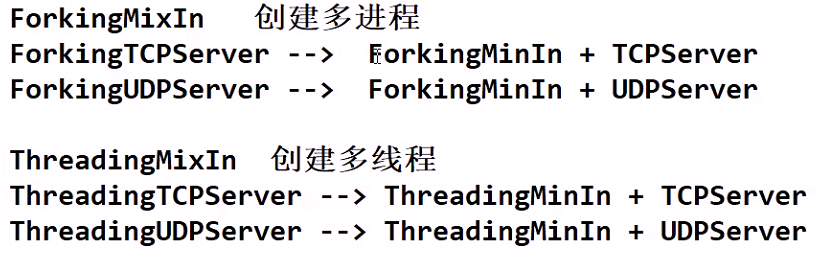
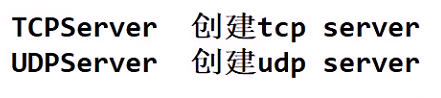
集成模块的使用

Python2 SocketServer

Python3 socketserver

功能：通过模块的不同类的组合完后才能多进程/多线程的TCP/udp的并发





httpserver v2.0

1. 接收客户端请求
2. 解析客户端请求
3. 组织数据，形成HTTP response
4. 将数据发送给客户端

升级

1. 采用多线程并发接收多个客户端请求
2. 基本的请求解析，根据请求返回相应的内容
3. 除了可以请求静态网页，也可以请求简单的数据
4. 将功能封装在一个类中

技术点：

1. socket TCP套接字
2. HTTP协议的请求响应格式
3. 线程并发的创建方式
4. 类的基本使用

协程基础：

定义：协程，微线程。协程的本质是一个单线程程序，所以协程不能够使用计算机多核资源

作用：

能够高效的完后才能并发任务，占用较少的资源，因此协程的并发量较高

原理：

通过记录应用层的的上下文栈区，显示在运行中进行上下文跳转，达到可以选择性的运行想要运行的部分，以此提高程序的运行效率。

优点： 消耗资源少

无需切换的开销

无需同步互斥

IO并发性好

缺点：无法利用计算机多核

yield--->协程实现的基本关键字

greenlet

pip安装

greenlet.greenlet()生成协程对象

gr.switch()选择要执行的协程事件

gevent

1. 将协程事件封装为函数
2. 生成协程对象

gevent.spawn(func，argv)

功能：生成协程对象

参数：func协程函数

argv给携程函数传参

返回值：返回协程对象

3回收协程

gevent.joinall()

功能：回收协程

参数：列表，将要回收的协程放入列表

gevent.sleep(n)

功能：设置协程阻塞，让携程跳转

参数： n阻塞事件

from gevent import monkey

monkey.patch\_all()

功能：修改套接字的IO阻塞行为

\*必须在socket导入之前使用