# 目录

[目录 1](#_Toc517884289)

[一、网络编程 4](#_Toc517884290)

[网络目的：数据的传输 4](#_Toc517884291)

[ISO：国际标准化组织 4](#_Toc517884292)

[OSI七层模型🡪网络通信的标准化流程 4](#_Toc517884293)

[四层模型 4](#_Toc517884294)

[五层模型 4](#_Toc517884295)

[网络相关概念 5](#_Toc517884296)

[socket套接字编程 8](#_Toc517884297)

[套接字对象 12](#_Toc517884298)

[udp套接字应用之广播 13](#_Toc517884299)

[tcp应用之http传输 14](#_Toc517884300)

[基础的http服务器 17](#_Toc517884301)

[IO input output 18](#_Toc517884302)

[extra：位运算 24](#_Toc517884303)

[本地套接字 24](#_Toc517884304)

[extra 25](#_Toc517884305)

[二、多任务编程（进程线程） 26](#_Toc517884306)

[一、进程 26](#_Toc517884307)

[二、线程 50](#_Toc517884308)

[三、服务器开发训练 56](#_Toc517884309)

[服务器模型 56](#_Toc517884310)

[extra 文件处理函数 58](#_Toc517884311)

[Thread并发服务器 58](#_Toc517884312)

[socket套接字服务器集成模块 58](#_Toc517884313)

[基于多线程并发的HTTPServer 59](#_Toc517884314)

[协程 61](#_Toc517884315)

[四、mongodb 62](#_Toc517884316)

[关系型数据库 63](#_Toc517884317)

[非关系型数据库（nosql）🡪not only sql 63](#_Toc517884318)

[关系型数据库和非关系型数据库的比较 63](#_Toc517884319)

[NoSql的使用情况 64](#_Toc517884320)

[MongoDB安装 64](#_Toc517884321)

[MongoDB命令 65](#_Toc517884322)

[文档 67](#_Toc517884323)

[值：即文档存储的数据，支持bson类型 68](#_Toc517884324)

[集合的设计 69](#_Toc517884325)

[插入文档：db.collectionName.insert() 69](#_Toc517884326)

[文档查找：db.collection.find(query,field) 70](#_Toc517884327)

[查找结果处理函数 73](#_Toc517884328)

[删除文档 74](#_Toc517884329)

[练习 75](#_Toc517884330)

[修改数据（修改器） 76](#_Toc517884331)

[时间类型 79](#_Toc517884332)

[null 79](#_Toc517884333)

[Object 内部文档 80](#_Toc517884334)

[数组的下标引用 80](#_Toc517884335)

[查找结果的有序性 80](#_Toc517884336)

[索引 80](#_Toc517884337)

[聚合操作 82](#_Toc517884338)

[固定集合 84](#_Toc517884339)

[文件存储 84](#_Toc517884340)

[游标 85](#_Toc517884341)

[五、正则表达式 89](#_Toc517884342)

[动机： 89](#_Toc517884343)

[特点 89](#_Toc517884344)

[元字符 90](#_Toc517884345)

[正则表达式转义 95](#_Toc517884346)

[raw字符串 95](#_Toc517884347)

[正则表达式分组 96](#_Toc517884348)

[捕获组和非捕获组（命名组和非命名组） 97](#_Toc517884349)

[re模块 97](#_Toc517884350)

[match对象属性变量 100](#_Toc517884351)

[match对象属性方法 100](#_Toc517884352)

[六、第二阶段项目训练 103](#_Toc517884353)

[项目 103](#_Toc517884354)

[\*电子词典 104](#_Toc517884355)

[extra 106](#_Toc517884356)

[pip工具的使用 106](#_Toc517884357)

[pdb调试 107](#_Toc517884358)

[\*httpserver带框架版本 107](#_Toc517884359)

# 一、网络编程

### 网络目的：数据的传输

### ISO：国际标准化组织

### OSI七层模型🡪网络通信的标准化流程

1. 应用层：提供用户服务，具体的内容由特定程序规定
2. 表示层：数据的压缩、优化、加密
3. 会话层：建立应用连接，选择传输层服务
4. 传输层：提供数据传输服务，流量控制
5. 网络层：路由选择，网络互连
6. 链路层：提供链路交换，具体消息的发送
7. 物理层：物理硬件，接口 网卡的规定

osi模型优点：将功能分开，降低网络传输中的耦合度，每一部分完成自己的功能，开发更加清晰专一。

### 四层模型

应用层：应用层 表示层 会话层

传输层：传输层

网络层：网络层

物理链路层：链路层 物理层

### 五层模型

应用层：应用层 表示层 会话层

传输层：传输层

网络层：网络层

链路层：链路层

物理层：物理层

day1要求：能够按照顺序说出七层模型，五层和四层模型，知道每一层是什么作用，对网络消息的传输流程有基本了解

协议：网络通信中，各方必须遵守的协议，包括建立什么样的连接，消息结构等。

### 网络相关概念

1. 主机：host 表示一台计算机

本地使用：‘localhost’ 或‘’ 或‘127.0.0.1’

网络使用：‘0.0.0.0’或‘本机IP’

获取计算机名：

import socket

socket.gethostname()

通过计算机名获取地址

socket.gethostbyname(‘tedu’)

1. 在网络上确定一台主机的地址

IPv4：点分十进制 比如：192.168.1.17 0—255

32位二进制表示

IPv6：128位二进制

windows下：cmd.exe窗口下：ipconfig

linux下：查看本机地址ifconfig

1. 网络连接测试

ping + ip

1. 特殊IP
2. 127.0.0.1 本地测试IP
3. 0.0.0.0 使用局域网内可用的IP
4. 192.168.1.0 表示当前网段，一般不用
5. 192.168.1.1 通常是网络节点设备的IP（网关），一般不用
6. 192.168.1.255 广播地址，一般不用
7. 通过域名获取服务器信息

socket.gethostbyaddr('www.baidu.com')

（‘127.0.0.1’,[], [‘117.75.213.61’]）

主机名 主机别名 主机IP

1. 将点分十进制IP转换为二进制

In [5]: socket.inet\_aton('192.168.1.2')

Out[5]: b'\xc0\xa8\x01\x02'

1. 将二进制IP转换为点分十进制

In [7]: socket.inet\_ntoa(b'\xc0\xa8\x01\x02')

Out[7]: '192.168.1.2'

1. 以下两个函数功能同上面两个完全相同，只是可以选择地址类型

inet\_pton(socket.AF\_INET,’192.168.1.2’)

inet\_ntop()

1. 域名：网络服务器地址的网络名称
2. 端口号：是网络地址的一部分，在一个系统中，每个网络应用都会有一个端口号相对应，用来在接收消息时确定由谁来接收

范围：1—65535

1—255 一些通用的众所周知的程序占用

256--1023 系统应用端口

1024—65535 自用 建议使用大于10000的端口

获取应用端口：In [13]: socket.getservbyname('mysql')

Out[13]: 3306

1. 字节序

数据在内存中的存储解析方式，不同操作系统可能不同

网络字节序：不同的操作系统在消息收发时均按照该标准操作

1. 传输层服务
   1. 面向连接的传输服务🡪 tcp协议
2. 传输特征：
   1. 可靠的数据传输：可靠性是指数据传输中无失序 无差错 无丢失 无重复
   2. 所有的消息传输前一定会建立连接，传输后一定会断开连接
3. 三次握手：在进行面向连接的数据传输前进行传输连接的过程
   1. 客户端向服务器发起连接请求（问是否可以连接）
   2. 服务器收到连接请求进行确认，返回报文
   3. 客户端收到回复，进行连接建立
4. 四次挥手：在进行面向连接的数据传输时，断开连接的过程
   1. 主动方发送报文，告知被动方要断开连接
   2. 被动方返回报文，告知收到请求，准备断开
   3. 被动方再次发送报文给主动方，告知准备完毕，可以断开
   4. 主动方发送报文进行断开
5. 应用情况：适用于传输较大的内容或文件，网络情况良好，需要保证传输可靠性的情况，比如：聊天信息、文件的上传下载、邮件传输、网页获取
   1. 面向无连接的传输服务🡪 udp协议
6. 传输特点
   1. 不保证可靠的数据传输
   2. 没有连接过程
   3. 数据的收发比较自由，不会受另一端制约
7. 应用情况：网络情况较差，对传输可靠性要求不高，需要提升传输效率的情况，比如：网络视屏、群聊、发送广播

13. 问题总结：

1. osi模型问题
2. 三次握手和四次挥手问题
3. tcp和udp的区别

### socket套接字编程

1. 目的：通过编程语言提供的函数接口进行组合，更简单的完成基于tcp或者udp通信的网络编程
2. 套接字：完成上述目标的编程方法
3. 套接字分类：
   1. 流式套接字（SOCK\_STREAM）：传输层基于tcp协议进行通信
   2. 数据报套接字（SOCK\_DGRAM） :传输层基于udp协议进行通信
   3. 底层套接字（SOCK\_RAM）：访问底层协议的套接字

#### TCP服务端

import socket

1. 创建套接字

socket.socket(socket\_family = AF\_INET, socket\_type = SOCK\_STREAM, proto = 0)

* + - 1. 功能：创建一个套接字
      2. 参数：socket\_family：选择地址族类型

socket\_type：套接字类型SOCK\_STREAM流式套接字，SOCK\_DGRAM数据报套接字

proto：子协议类型，没有子协议选择0

* + - 1. 返回值：套接字对象

1. 绑定地址（IP和端口号）

sockfd.bind()

功能：绑定地址

参数：一个二元元祖（第一项为IP，第二项为端口号），如（‘172.60.50.50’,8888）

1. 将套接字设置可监听

sockfd.listen(n)

功能：将套接字设置为监听套接字

参数：n为一个正整数，表示监听等待队列的大小

1. 等待接收连接请求

connfd,addr = sockfd.accept()

功能：阻塞等待处理客户端连接

返回值：1.返回一个新的套接字，用来和客户端通信；2.连接的客户端的地址

\*阻塞函数：当程序运行到阻塞函数的位置，如果某种期待条件没有达成则暂停继续运行，当条件达成后结束阻塞继续运行

1. 收发消息

data = connfd.recv(buffersize)

功能：流式套接字接收消息

参数：一次最多接收多大的消息，单位为字节

返回值：返回接收到的内容

recv()特性：

1. 如果连接双方断开连接，则recv会立即结束阻塞返回空字符串
2. 当发送缓冲区满的时候会阻塞
3. 如果一次接收不完缓冲区内容，下一次会继续接收

n = connfd.send(data)

功能：流式套接字发送内容

参数：要发送的内容，要求bytes格式

返回实际发送的字节数

send()特性：

1. 如果一端不存在，另一端还在试图send操作时，会产生BrokenPipeError异常
2. 关闭套接字

close()

功能：关闭套接字，tcp套接字连接断开

#### TCP客户端

1. 创建套接字

socket()

1. 请求连接

connect()

功能：发起连接请求

参数：是一个元组，表示服务器的地址

1. 消息收发

send() recv()

1. 关闭套接字

close()

#### 网络收发缓冲区

1. 发送和接收消息均先放到缓冲区中，再进行处理，即recv和send实际是从缓冲区接收内容，向缓冲区发送消息
2. 当接收缓冲区为空的时候会阻塞

#### TCP粘包

1. 产生原因：tcp传输中以字节流的方式发送内容，发送的消息之间没有明显的边界，此时如果发送和接收的速度不匹配就会产生粘包
2. 影响：如果每次发送的内容表示一个独立的意思，此时可能需要处理粘包，如果发送的内容本身就是一个连续的整体，就不需要处理粘包
3. 如何处理
   1. 每次发送后加一个结尾标志
   2. 发送一个数据结构
   3. 每次发送中间有一个短暂的延迟

#### udp服务端

1. 创建套接字🡪数据报套接字

sockfd = socket(AF\_INET,SCOK\_DGRAM)

1. 绑定服务端地址

sockfd.bind()

1. 消息的收发

data, addr= recvfrom(buffersize)

功能：接收udp消息

参数：每次最多接收消息的大小

返回值：接收到的内容，接收到的消息的发送者

recvfrom一次接收一个数据包，如果数据包超出了一次能够接受的大小则会丢失没有收到的内容

sendto(data, addr)

功能：udp发送消息

参数：data 要发送的内容， bytes格式，addr 消息要发送给谁

返回值：发送的字节

1. 关闭套接字

sockfd.close()

#### extra：

sys.argv

功能：获取来自命令行的参数，形成一个列表

示例：🡪python3 argv.py hello

🡪['argv.py', 'hello']

argv[0]是程序本身

总结：tcp流式套接字和udp数据报套接字使用区别

1. 流式套接字采用字节流的方式传输，而数据报套接字以数据报形式传输

2. tcp会产生粘包现象，udp不会

3. tcp编程保证传输的可靠性，udp则不保证

4. tcp需要listen，accept，udp不需要

5. tcp使用recv，send进行消息收发，udp使用recvfrom和sendto

补充：

sendall（）

功能：同send

返回值：发送成功返回none，否则返回异常

### 套接字对象

1. s.fileno() s表示一个套接字对象
   1. 获取套接字的文件描述符
      1. 文件描述符：每一个IO事件，操作系统都会分配一个不同的正数与之匹配，该整数即此IO操作的文件描述符
      2. sys.stdin.fileno() 0
      3. sys.stdout.fileno() 1
      4. sys.stderr.fileno() 2
2. s.type：获取套接字的类型
3. s.getsockname()
   1. 功能：获取套接字绑定的地址
4. s.getpeername()
   1. 功能：用户
5. s.setsockopt(level, optname, value)
   1. 功能：设置套接字选项，丰富修改原有套接字功能
   2. 参数：level：设置选项的类型（SOL\_SOCKET常用） optname：选项类型中的子选项 value：为选定的选项设置值
6. s. getsockopt(level, optname)
   1. 功能：获取套接字选项
   2. 参数：level：获取选项的类型 optname：

### udp套接字应用之广播

1. 一点发送多点接收
2. 目标地址：广播地址：一个网段内最大的地址，如172.60.50.255
3. 发送广播

from socket import \*

from time import sleep

# 广播地址

dest = ('176.209.101.255',9999)

s = socket(AF\_INET,SOCK\_DGRAM)

# 设置套接字可以发送接收广播

s.setsockopt(SOL\_SOCKET,SO\_BROADCAST,1)

while True:

sleep(2)

s.sendto('fighting!'.encode(),dest)

s.close()

1. 接收广播

from socket import \*

# 创建套接字

s = socket(AF\_INET,SOCK\_DGRAM)

# 设置可以接收广播

s.setsockopt(SOL\_SOCKET,SO\_BROADCAST,1)

# 固定端口号

s.bind(('',9999))

while True:

try:

msg,addr = s.recvfrom(1024)

print('从{}获取信息{}'.format(addr,msg.decode()))

except (KeyboardInterrupt,SyntaxError):

raise

except Exception:

print(e)

s.close()

1. 广播风暴：占用大量的带宽造成网络拥塞

### tcp应用之http传输

http协议：超文本传输协议

用途：编写基于http协议的数据传输程序，网络中浏览器端获取网页的过程

作用：将想要获取的内容以http协议的格式发送给服务端，服务端根据格式进行解析，获取到真实需求，将结果以http协议的格式回复给客户端

特点：

1. 应用层协议，传输层使用tcp服务
2. 简单、灵活和多种语言对接方便
3. 无状态的协议，即不记录用户使用过程中传递的内容
4. http1.1 支持持久连接

1）http请求（request）

请求格式：

请求行：具体的请求类别

GET / HTTP/1.1

请求种类 请求内容 协议版本

请求头：对请求信息的具体描述

Accept：text/html

以键值对的格式对信息进行具体描述

空行

请求体：具体的请求参数或者提交内容

请求类型：

GET 获取网络资源

POST 提交一定的附加数据，得到返回结果

HEAD 获取响应头

PUT 更新服务器资源

DELETE 删除服务器资源

CONNECT

TRACE 用于测试

OPTIONS 获取服务器性能

2）http响应（response）

格式：

响应行 反馈具体的响应情况

HTTP/1.1 200 OK

版本 响应码 附加信息

响应头 对响应的具体描述

Accept-Ranges:bytes

以键值对方式给出响应信息的具体描述

空行

响应体 将客户想要的内容进行返回

响应码种类：

1xx 提示信息，表示请求已经接受

2xx 响应成功

3xx 响应需要重定向

4xx 客户端错误

5xx 服务端错误

200 成功

404 请求内容不存在

401 没有访问权限

500 服务器发生未知错误

503 服务器暂时不能执行

day2要求：

什么是http协议

请求格式和每部分的功能

响应格式和每部分功能

http请求和响应的流程

#### extra：

format：

示例：s = ‘hello,{}’

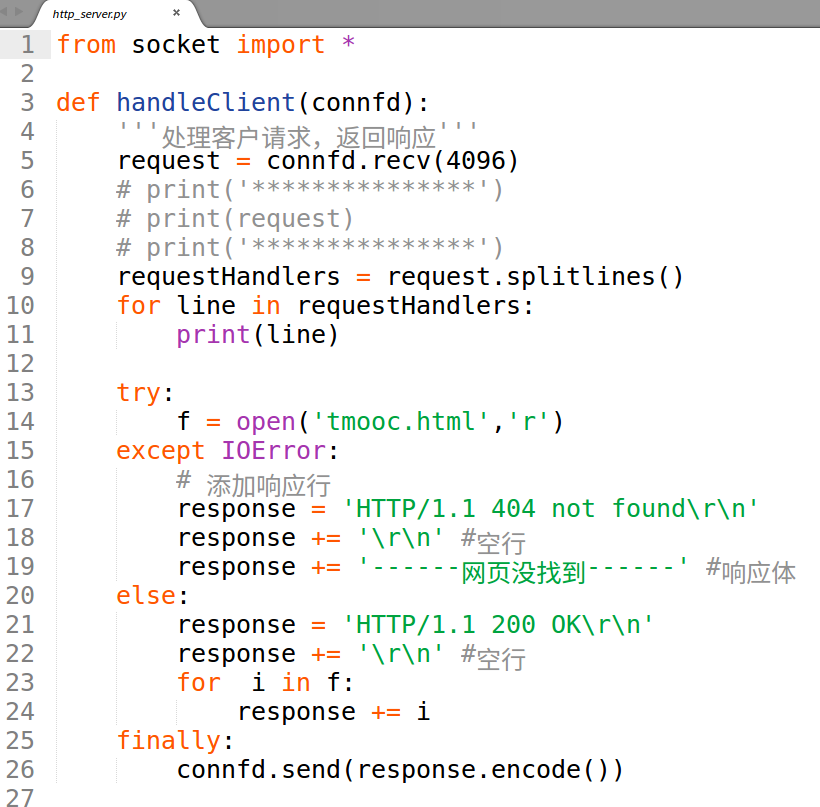
s.format(‘lea’)🡪’hello,lea’

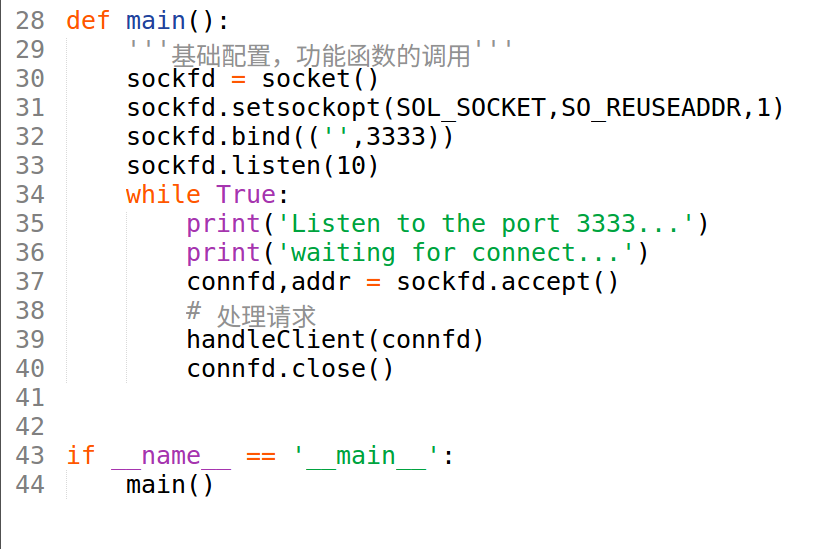
s2 = ‘hello,{1},{0}’

s2.format(‘lea’,’troye’)🡪’hello,troye,lea’

### 基础的http服务器

1. 接收HTTP请求
2. 给出一定的响应
3. 示例：





### IO input output

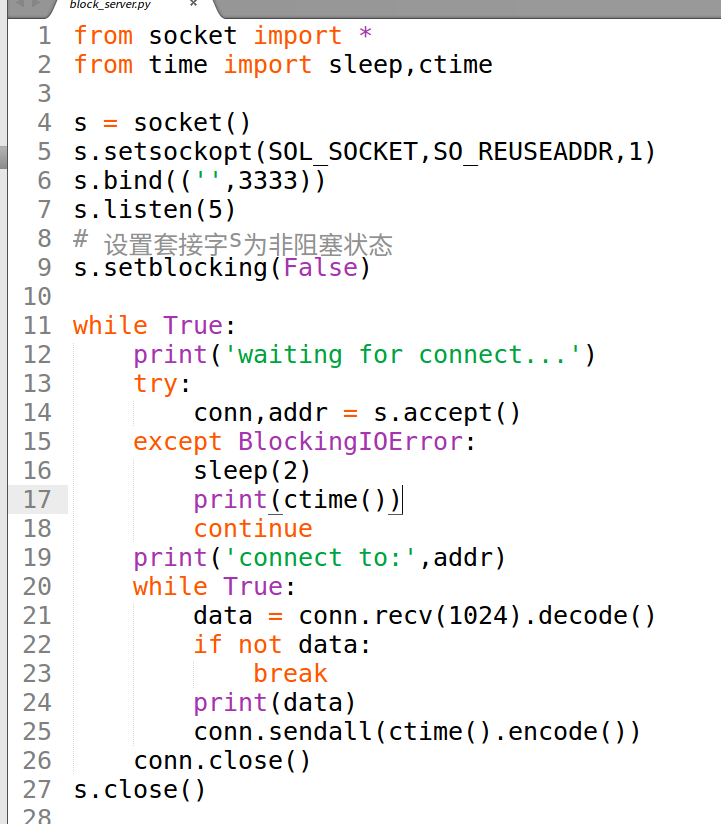
1. 概念：在内存中存在数据交换的操作都可以认为是输入输出，

例如：内存和磁盘交互 🡪 读写操作；内存和网络交互🡪recv send

1. 程序分类：
   1. IO密集型程序：程序执行中有大量的IO操作，而较少的CPU运算。特点：消耗CPU较少，运行时间长
   2. CPU密集型程序（计算密集型）：程序中大量的操作都需要CPU运算，IO操作较少。特点：消耗CPU大，运行速度快
2. IO分类：阻塞IO 非阻塞IO IO多路复用 事件IO 异步IO…
   1. 阻塞IO：默认形态，效率很低的一种IO情形
      1. 阻塞情况1：因为某种条件没有达成造成的阻塞，如：accept，recv，input…
      2. 阻塞情况2：处理IO事件的时候耗时长，形成阻塞，如：文件的读写过程、网络的数据发送过程
   2. 非阻塞IO：通过修改IO事件的属性使其变为非阻塞的状态（改变了阻塞情况1）
3. 通常和循环搭配使用，不断检测阻塞条件是否已经满足
4. s.setblocking()

功能：将套接字设置为非阻塞状态；

参数：bool 设置为False则表示设置为非阻塞



1. 超时检测（超时等待）：将原本阻塞的函数设置一个阻塞时间，在规定时间内如果条件达到，则正常执行，如果仍然阻塞，则抛出异常

s.settimeout(second)

功能：设置套接字超时时间

参数：设置的时间



* 1. IO多路复用
     1. 定义：同时监控多个IO事件，当哪个IO事件准备就绪就执行哪个IO事件，以此形成，多个IO事件都可以操作，不必逐个等待执行的效果
     2. 准备就绪：IO事件即将发生的临界状态
     3. import select
        1. select ----> windows linux unix
           1. r,w,x = select(rlist, wlist, xlist[, timeout])

功能：监控IO事件，阻塞等待IO事件的发生

参数：rlist：列表 存放监控等待处理的IO事件

wlist：列表 存放要主动处理的IO事件

xlist：列表 存放如果发生异常需要处理的IO事件

timeout：超时时间

返回值：r 列表 rlist中准备就绪的IO

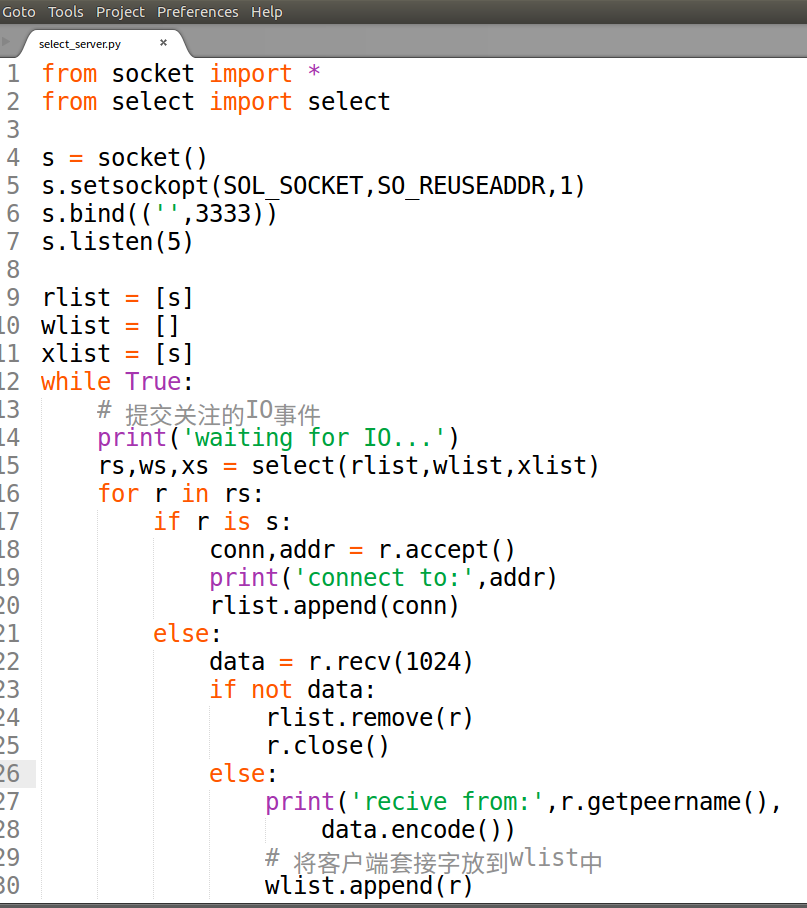
w 列表 slist中准备就绪的IO

x 列表 xlist中准备就绪的IO

* + - * 1. 注意事项：

在处理IO过程中不应该发生死循环（某个IO单独占有服务器）

IO多路复用形成了一种并发的效果，效率较高



2. poll---->linux unix

1. 创建poll对象

p = select.poll()

1. 添加关注对象

p.register(s，POLLIN | POLLERR) p.unregister(s)

POLL IO事件分类

POLLIN：rlist

POLLOUT：wlist

POLLERR：xlist

POLLHUP：断开

POLLPRI：紧急处理

POLLVAL：数据无效

1. 进行监控

events = p.poll()

功能：阻塞等待register的事件发生

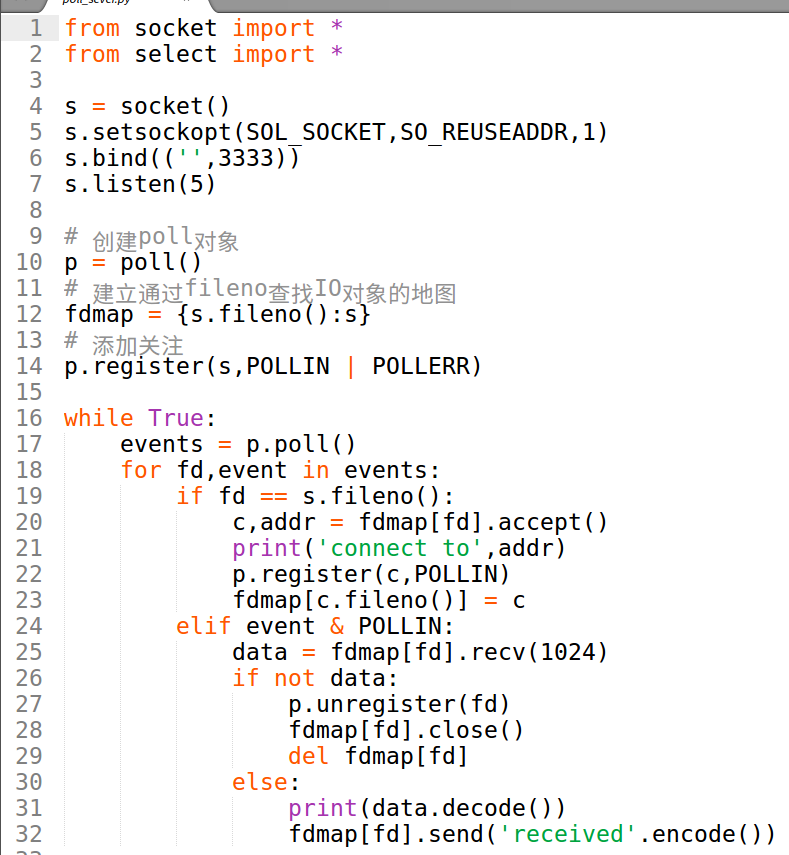
返回值：列表，每个元素表示准备就绪需要处理的IO

[(fileno,event), (), (),…]

fileno：描述符；event：具体什么就绪了

描述符地图：{s.fileno():s}

1. 处理IO事件



day3要求：1.熟练写出select server代码

2.描述IO多路复用的执行原理和内部机制

3.巩固HTTPserver的代码，理解HTTP服务器执行流程

3. epoll---->linux unix

使用方法：代码基本与poll相同

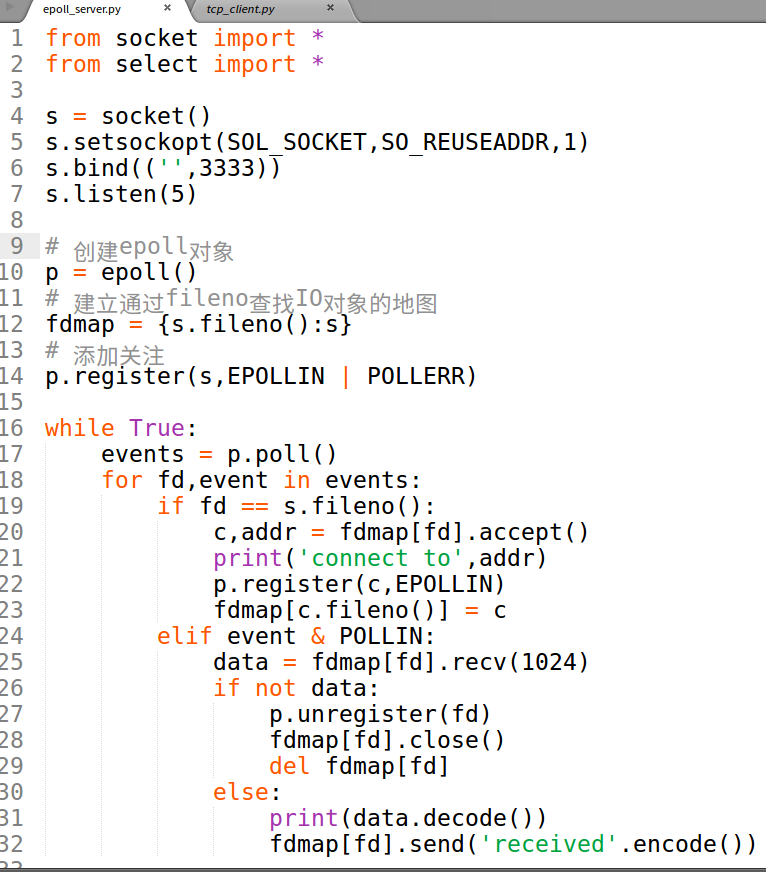
1.将生成对象的poll（）函数变为epoll（）

2.将register注册IO事件时关注的事件

区别：

1.epoll效率要高于select 和 poll

2.epoll的关注触发方式多一些



### extra：位运算

按照二进制位进行操作运算

&（按位与）

|（按位或）

^（按位亦或，相同为0，不同为1）

<<（左移）：1<<2🡪4

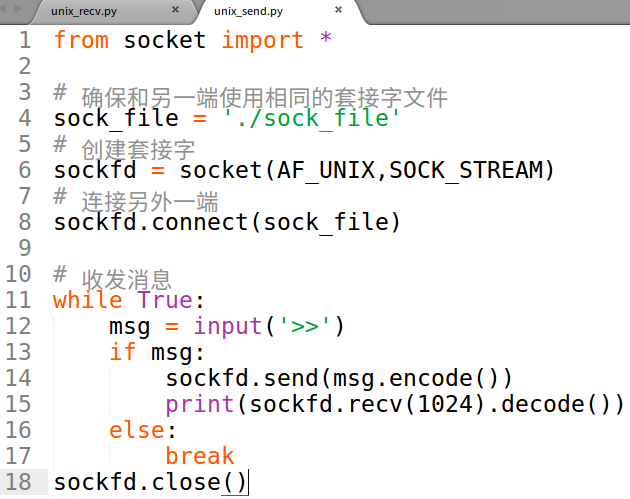
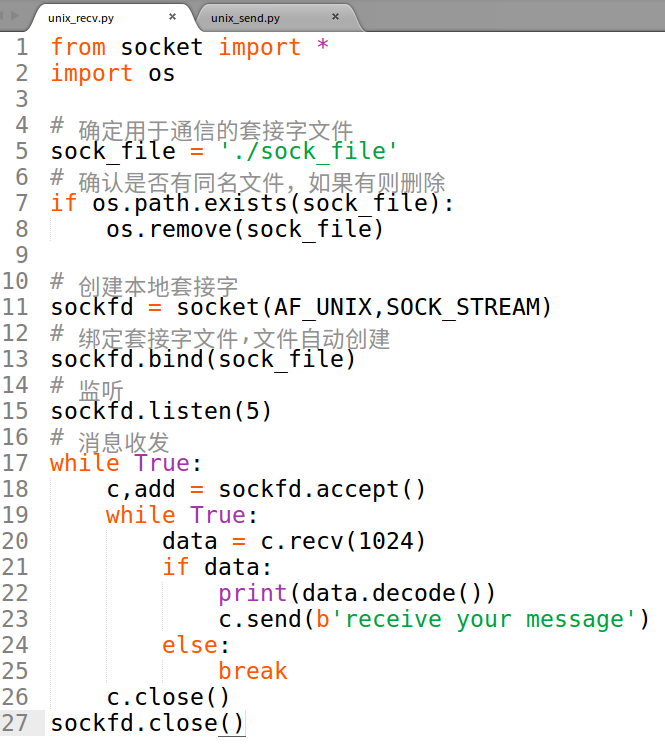
>>（右移）:挤掉低位的数字，11>>2🡪2

使用：1. 在做底层硬件的寄存器操作

2. 在做标志位过滤时

### 本地套接字

1. linux下文件类型
   1. b（块设备文件）
   2. c（字符设备文件），如：键盘，鼠标，打印机…
   3. d（目录）
   4. –（普通文件）
   5. l（链接文件）
   6. s（套接字文件）
   7. p（管道文件）
2. 作用：用于本地不同的程序间进行通信
3. 本地套接字创建流程
   1. 创建套接字对象：sockfd = socket（AF\_UNIX，SOCK\_STREAM）
   2. 建立套接字文件：sockfd.bind（文件路径字符串）绑定一个文件
   3. 监听：
   4. 接收发送消息



### extra

os.path.exists(path)

功能：判断一个文件是否存在

参数：文件位置

返回值：存在返回True，否则返回False

os.unlink()/os.remove()

功能：删除一个文件

参数：要删除的文件

# 二、多任务编程（进程线程）

1. 意义：充分利用计算机资源提供程序的运行效率
2. 定义：通过应用程序利用计算机的多个核心达到同时执行多个任务的目的，以此来提升程序的执行效率
3. 实施方案：多进程 多线程
4. 并行：多个计算机核心在同时处理多个任务，这多个任务之间是并行关系
5. 并发：同时处理多个任务，内核在任务间不断地切换，达到好像都在处理运行的效果

### 一、进程

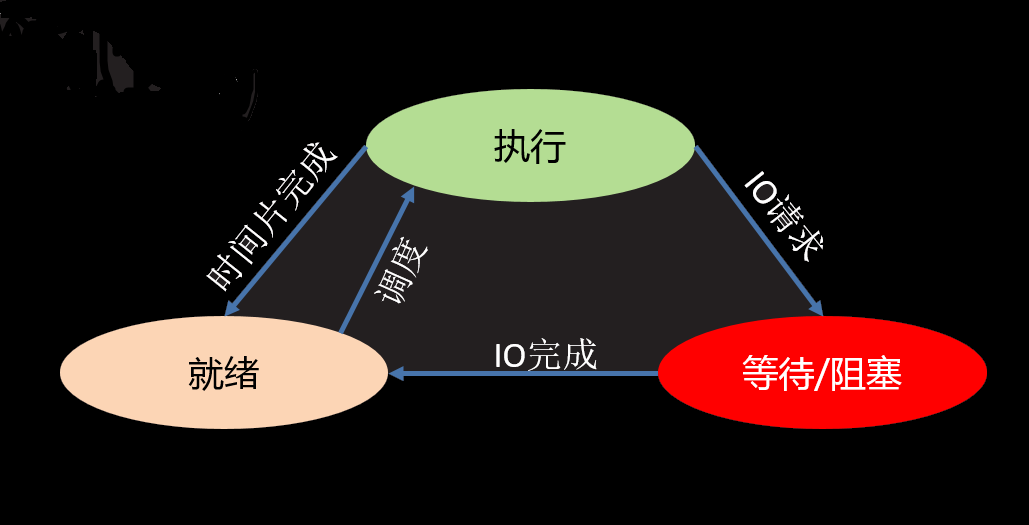
1. 意义：程序在计算机中的一次执行过程，是一个动态的过程描述，占有计算机的资源，有一定的生命周期

\*同一个程序的不同运行过程是不同的进程，因为分配的计算机资源不同，生命周期也不同

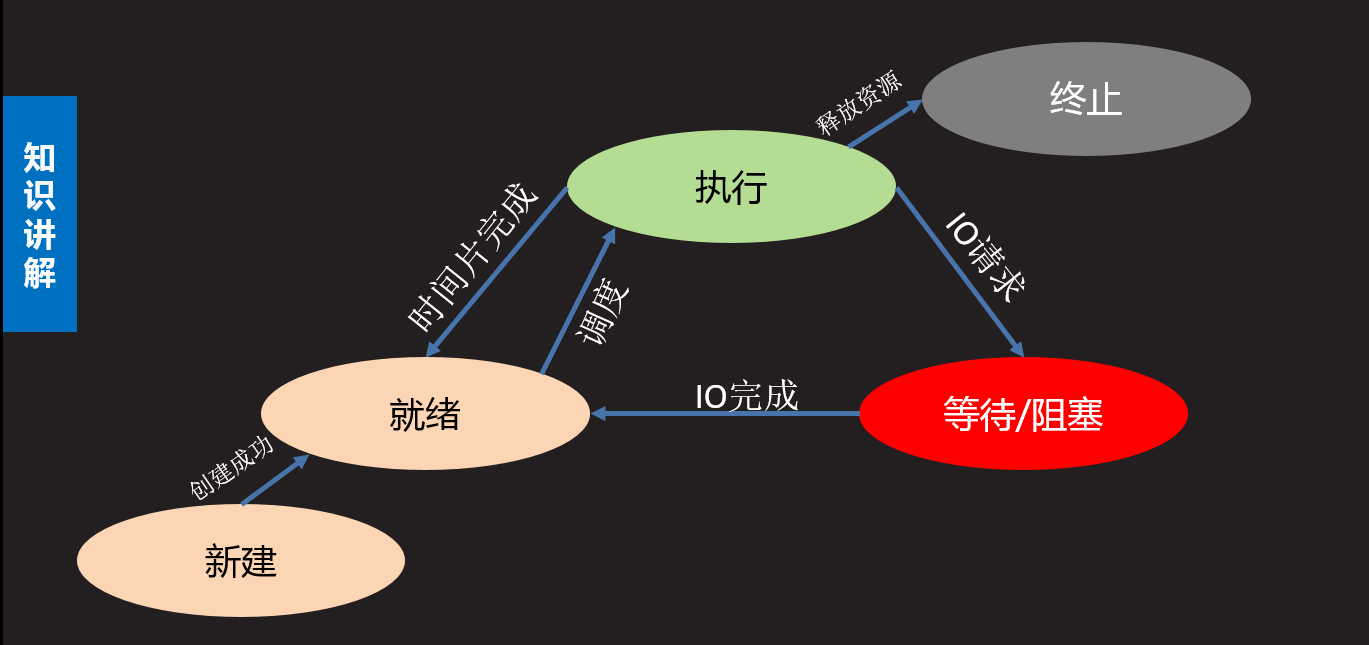
1. 程序：是一个可执行文件，是静态的，占有磁盘，不占计算机的运行资源
2. 进程的创建流程
   1. 用户空间运行一个程序，发起进程的创建
   2. 操作系统接受用户申请，开启进程创建
   3. 操作系统分配计算机资源，确定进程状态
   4. 将新创建的进程交给用户使用
3. cpu时间片：如果一个进程占有计算机的核心，我们称为该进程占有cpu时间片。多个任务实际会对cpu内核进行争夺，由操作系统分配cpu资源
4. 进程信息
   1. PCB（进程控制块）：在\*nix操作系统中，进程创建后会自动在内存中产生一个空间存放进程信息，称为PCB
   2. 进程信息：进程的ID 进程占有内存设置 创建时间 创建用户…
   3. 进程信息查看命令：ps -aux

\*PID（process id）：在操作系统中进程的唯一标志，大于0的整数，由系统自动分配

* 1. 进程的特征：
     1. 进程是操作系统分配计算机资源的最小单位
     2. 每个进程有自己单独的虚拟内存空间
     3. 进程之间的执行上相互独立，互不影响
  2. 进程的状态
     1. 三态
        1. 就绪态：进程具备执行条件，等待系统分配处理器资源
        2. 运行态：进程占有cpu处于运行的状态
        3. 等待态：进程暂时不具备运行条件，需要阻塞等待



* + 1. 五态
       1. 新建态：创建一个新的进程，获取资源的过程
       2. 就绪态：进程具备执行条件，等待系统分配处理器资源
       3. 运行态：进程占有cpu处于运行的状态
       4. 等待态：进程暂时不具备运行条件，需要阻塞等待
       5. 终止态：进程执行结束，资源释放回收的过程



* + 1. ps -aux 🡪STAT 表示进程状态
       1. D：等待态（不可中断等待）
       2. S：等待态（可中断等待）
       3. T：等待态（暂停状态）
       4. R：运行态（包含就绪态）
       5. Z：僵尸态
       6. +：前台进程，在终端运行
       7. <：高优先级
       8. N：低优先级
       9. l：有进程链接
       10. s：会话组
  1. 进程优先级：优先级决定了一个进程的执行权限和占有资源的优先程度
     + 1. 查看优先级：top 动态查看进程优先级 按</>号翻页
       2. 优先级的取值范围：-20—19 -20的优先级最高
       3. nice：以指定的优先级运行一个程序

例如：nice -9 ./while.py

nice --9 ./while.py

* 1. 父子进程
     1. 在系统中除了初始化进程，每个进程都有一个父进程，可能有0个或者多个子进程，由此形成进程的父子关系，可以认为每个进程都是父进程创造的
     2. 查看进程树：pstree
     3. 查看父进程PID：ps -ajx
  2. 要求：什么是进程，进程与程序的区别，进程与程序的特征，清楚进程的每一种状态，以及状态转换
  3. 需求：编写一个程序，能够同时做多件任务
  4. 方案：写一个程序，根据需要在程序内部可以创建多个进程完成任务

1. import os
2. pid = os.fork()

功能：创建一个新的进程

参数：无

返回值：失败： 返回一个负数-1

成功：在原有进程中返回新进程的PID号，在新进程中返回0

\*子进程会复制父进程全部代码段，包括fork之前产生的内存空间

\*父子进程通常会根据fork返回值的差异选择执行不同的代码

\*子进程虽然复制父进程的代码空间，但是有自己的特有属性，比如PID号 PCB等

\*父子进程在执行上互不干扰，执行顺序不确定

\*父子进程空间独立，在本进程中对空间的操作不会影响到其他进程

1. 进程相关函数使用
   1. os.getpid()：获取进程的PID（进程号），返回进程号
   2. os.getppid()：获取父进程的PID（进程号），返回进程号
2. 进程的退出
   1. os.\_exit(status)

功能：退出进程

参数：进程的退出状态，整数

* 1. sys.exit([status])

功能：退出进程

参数：默认为0

传入一个整数表示退出状态

传入一个字符串，则在进程退出时会打印该字符串

1. 孤儿进程：父进程先于子进程退出，此时子进程就会成为孤儿进程

\* 孤儿进程会被指定的进程收养，即系统进程会成为孤儿进程新的父进程，当孤儿进程退出时，系统父进程会处理孤儿进程退出状态，使其不会成为僵尸

1. 僵尸进程：子进程先于父进程退出，父进程没有处理子进程的退出状态，此时，子进程就会成为僵尸进程

\* 僵尸进程会滞留部分PCB信息在内存中，大量的僵尸进程会消耗系统资源，应该尽量避免僵尸进程的产生

\* 如何避免僵尸进程：

1.父进程先退出（不佳，因为不好控制）

2.父进程处理子进程的退出状态

* + - 1. pid,status = os.wait()

功能：在父进程中阻塞等待子进程的退出

返回值：一个二元元祖，pid：退出的子进程的PID号；status：退出的子进程的退出状态

* + - 1. pid,status = os.waitpid(pid, option)

功能：处理子进程的退出状态

参数：pid -1 表示等待任意子进程退出

正整数 表示等待对应子进程退出

option：0 表示阻塞等待

WNOHANG 表示非阻塞

返回值：同wait（）

waitpid(-1, 0) == wait()

3.创建二级子进程

1.父进程创建子进程后等待子进程退出

2.子进程创建二级子进程后马上退出

3.二级子进程称为孤儿，处理具体事件

#### 聊天室项目：

功能：类似QQ群聊

1.进入聊天室之前需要输入用户名

2.有人进入聊天室会像其他用户发起通知：\*\*\*进入了聊天室

3.一个人发消息，其他人都能收到：\*\*\*说：\*\*\*\*

4.某个人退出聊天室，其他人也会收到通知：\*\*\*退出了聊天室

5.管理员喊话功能：管理员发言所有客户端都能收到：管理员说：\*\*\*

1.方法：

存储用户：字典或列表（姓名和地址）

套接字：暂定udp套接字

怎么发消息：转发，即客户端发给服务器，然后服务器转发给其他人

\*在客户端让发送和接收相互独立，各使用一个进程

\*在服务端，管理员喊话和请求的接收需要独立，各使用一个进程

2.注意点：1.功能封装，将每个功能进行模块化

2.测试，每实现一个功能，每个功能进行测试

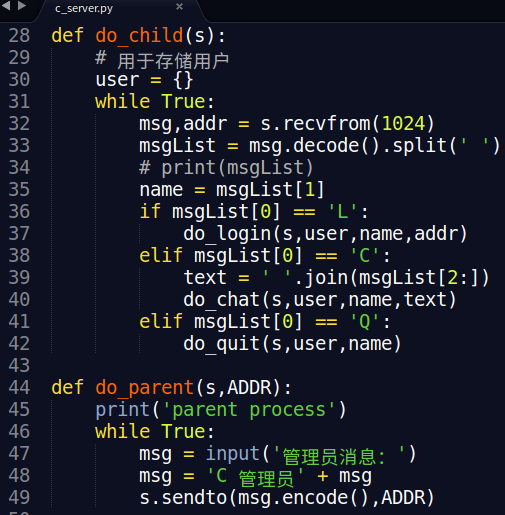
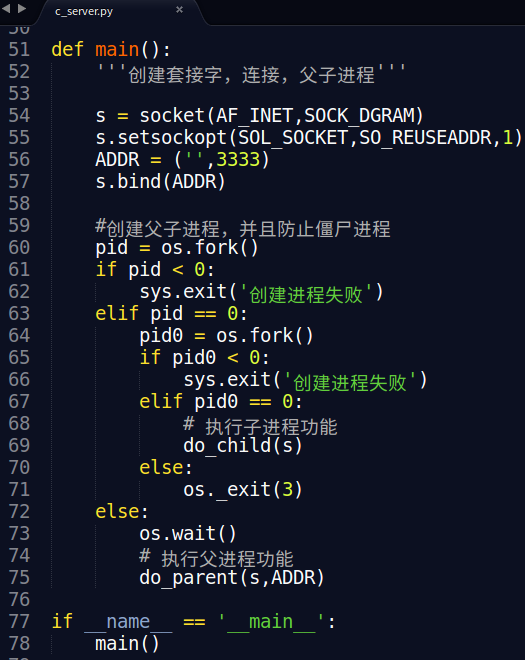
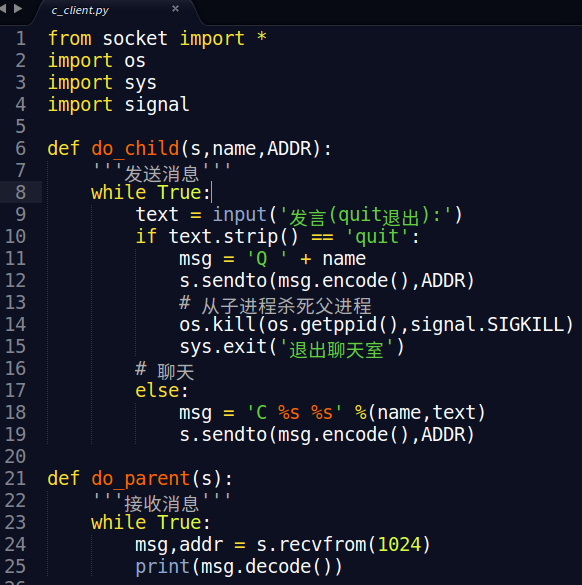
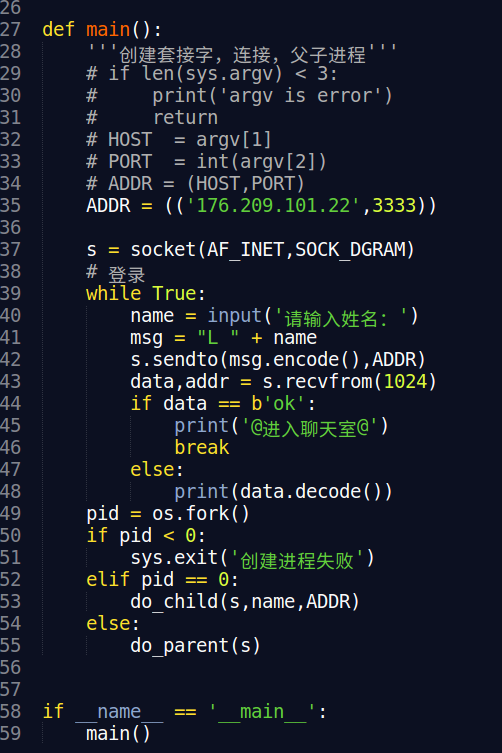
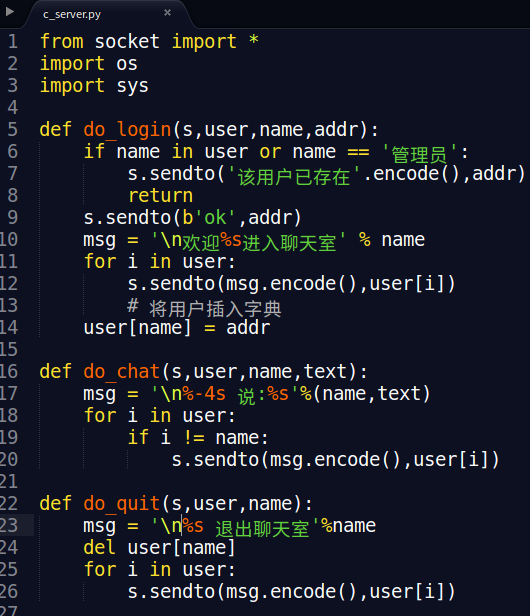
3.客户端可能发送的请求

1.进入聊天室

2.聊天请求

3.退出聊天室

4.代码编写流程：

搭建网络连接🡪多进程的创建🡪每个功能的编写🡪项目功能的逐个编写

#### multiprocessing 模块创建进程

* 1. 要求将要做的事件进行封装成函数
  2. 使用multiprocessing提供的类process创建进程对象
  3. 通过对象和process的初始化函数对进程进行设置，以及绑定要执行的事件
  4. 启动进程会自动地执行函数代表的事件
  5. 完成进程的回收

1. 创建进程对象
   1. Process（）

功能：创建进程对象

参数：name：给创建的进程对象起一个名字，默认为Process-1

target：绑定的函数

args：元组，用来给target函数传参，按位置传参

kwargs：字典，用来target函数传参，按关键字传参

* 1. p.start()

功能：启动进程，进程被创建，自动运行对应函数

* 1. p.join([timeout])

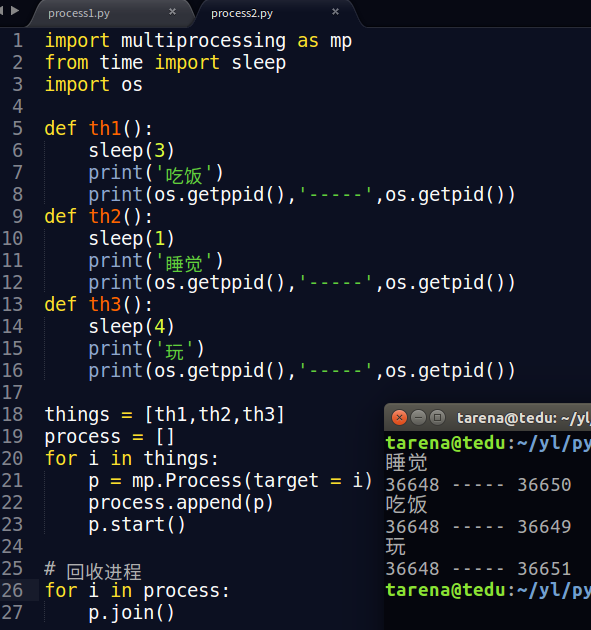
功能：阻塞等待对应子进程的退出，回收子进程

参数：超时时间

* 1. 注意：1.如果不使用join（），子进程将成为僵尸进程

2.在使用multiprocessing创建进程中，一般父进程功能就是创建子进程等待回收，不做过多的其他事情

3.使用multiprocessing创建子进程，同样子进程复制父进程空间，之后有自己独立的执行空间，互不干扰



* 1. p.is\_alive() 判断进程是否在生命周期状态
  2. p.name() 进程名称 默认为Process-1，如果起名字则为自己取的名字
  3. p.pid() 进程的PID号
  4. p.daemon 默认为False，主进程退出不会影响子进程，如果设置为True，则主进程退出时会让所有子进程都退出

\*该属性的设置必须在start（）前设置

\*该属性一般不用和join同时出现



##### 创建自定义进程

1. 继承Process类
2. 运行Process类的\_\_init\_\_以获取父类属性
3. 重写run方法，在通过自定义类生成对象后，调用start（）会自动执行这个方法

多进程：

优点：并行执行多个任务，提高效率

创建方便

运行独立，不收其他进程影响

数据安全

缺点：在进程的创建和删除过程中消耗计算机资源较多

##### 进程池技术

产生原因：如果有大量的任务需要多进程完成，可能需要频繁地创建和删除过程，给计算机带来较多的消耗

使用：大量可以短时间完成的任务需要多进程操作的时候比较适用于进程池

操作流程：

1. 创建进程池，在池内放入适当的进程
2. 将事件加入进程池队列
3. 进程事件不断运行，直到所有事件运行完成
4. 关闭进程池，回收进程

from multiprocessing import Pool

Pool（processes）

功能：创建进程池对象

参数：processes 表示进程池中进程数量

对象：进程池对象

1.pool.apply\_async(func，args，kwds)

功能：同时将事件放入进程池进行等待

参数：func 要放入进程池的事件函数；args 给func函数以元组传参；kwds 给func函数以字典传参

返回值：返回一个事件对象，通过get()方法可以获取事件函数的返回值

2.pool.apply (func，args，kwds)

功能：依次将事件放入进程池进行等待

参数：func 要放入进程池的事件函数；args 给func函数以元组传参；kwds 给func函数以字典传参

pool.close()

功能：关闭进程池，不能再添加新的事件

pool.join()

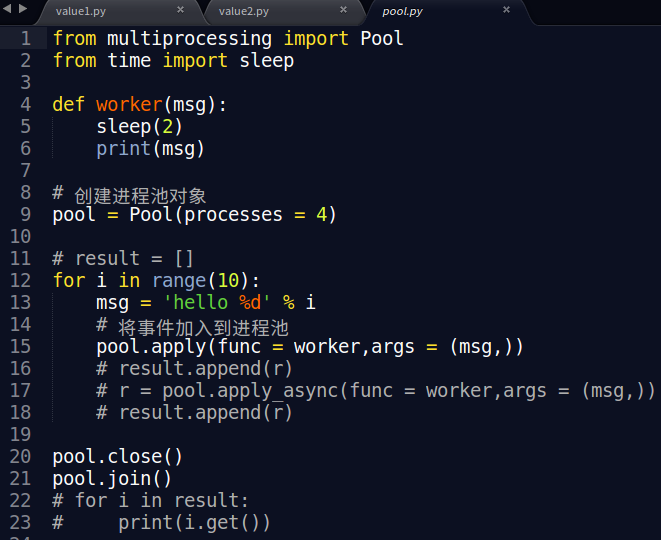
功能：回收进程池

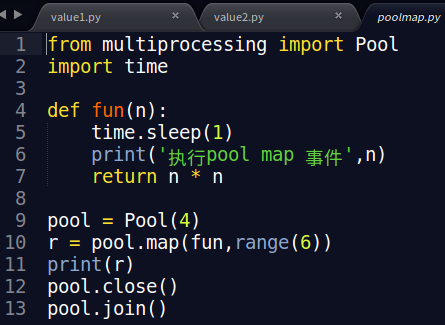
3. pool.map(func,iter)

功能：将要完成的事件放入到进程池

参数：func：要完成的事件函数；iter：要给func传递的参数的迭代器

返回值：返回事件函数的返回值列表





#### 进程间通信

进程间由于空间独立，资源互相无法直接获取，此时在不同的进程间传递数据就需要专门的进程间通信方法

和磁盘交互：使用中间文件，但是不安全，速度慢

进程间通信方法（IPC）

管道 消息队列 共享内存 信号 信号量 套接字

##### 管道通信Pipe

在内存中开辟一块空间，形成管道结构，管道对多个进程可见，进程可以通过对管道的读写操作进行通信

multiprocessing🡪Pipe

fd1,fd2 = Pipe(duplex = True)

功能：创建一个管道

参数：默认表示管道为双向管道，如果设置为False，则表示单向管道

返回值：返回两个管道流对象，表示管道两端，如果是双向管道，则都可以读写，如果是单向管道，则fd1只读，fd2只写

fd1.recv()

功能：从管道内读信息

返回值：读到的内容

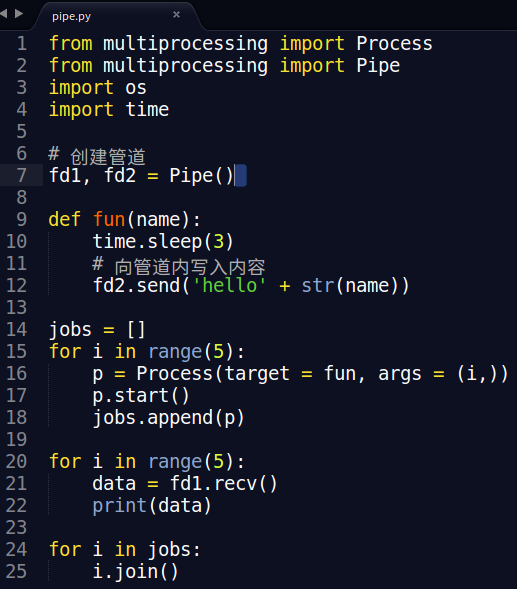
\*当管道内无内容的时候回阻塞

fd2.send(data)

功能：向管道写入内容

参数：要写的内容

\*可以发送几乎python的任意数据类型



##### 消息队列

队列：先进先出

在内存中开辟队列结构空间，对多个进程可见，多个进程向队列中存入消息，取出消息，完成进程间通信

创建队列

from multiprocessing import Queue

q = Queue（maxsize = 0）

功能：创建队列

参数：maxsize默认表示根据系统分配空间存储消息，如果传入一个正整数则表示最多存入消息数量

返回值：队列对象

q.put(data,[block,tiimeout])

功能：向队列中存入消息

参数：data：存入的数据（支持python数据类型）；block：默认为True，表示阻塞，当队列满时阻塞，设置为False，表示非阻塞；timeout：当block为True时，表示超时时间

data= q.get([block,timeout])

功能：从队列获取消息

参数：block：默认为True，表示阻塞，当队列满时阻塞，设置为False，表示非阻塞；timeout：当block为True时，表示超时时间

返回值：返回获取到的消息

q.full() 判断队列是否为满

q.empty() 判断队列是否为空

q.qsize() 获取队列中消息数量

q.close() 关闭队列



##### 共享内存

在内存中开辟一端空间存储数据，对多个进程可见，每次写入共享每次的数据会覆盖之前的内容

from multiprocessing import Value, Array

1.obj = Value（ctype，obj）

功能：开辟共享内存空间

参数：ctype 字符串 要变的c的类型code（对照ctype表）；obj：共享内存初始值

返回值：返回共享内存对象

obj.value：表示共享内存中的值，对其修改或者使用即为使用共享内存中的值

2.obj = Array(ctype,obj)

功能：开辟共享内存空间

参数：ctype 要转换的类型；obj 要存入共享内存的数据

列表 将列表存入共享内存，要求类型一致

整数 在共享内存内开辟几个单元的空间

返回值：返回一个共享内存对象

\*\*\*\*\*\*\*\*\*\*\*管道 消息队列 共享内存

开辟空间 内存 内存 内存

读写方式 两端读写 先进先出 操作覆盖内存

双向/单向

效率 一般 一般 较快

应用 多用于父子进程 广泛灵活 较复杂，需要互斥机制

作业：熟练掌握进程间通信方法

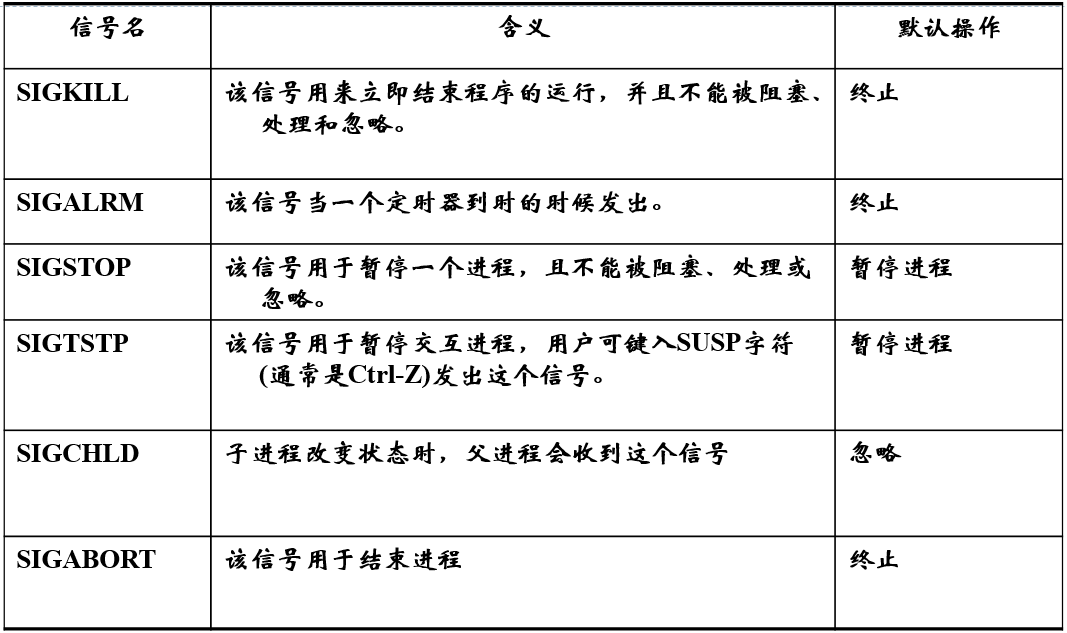
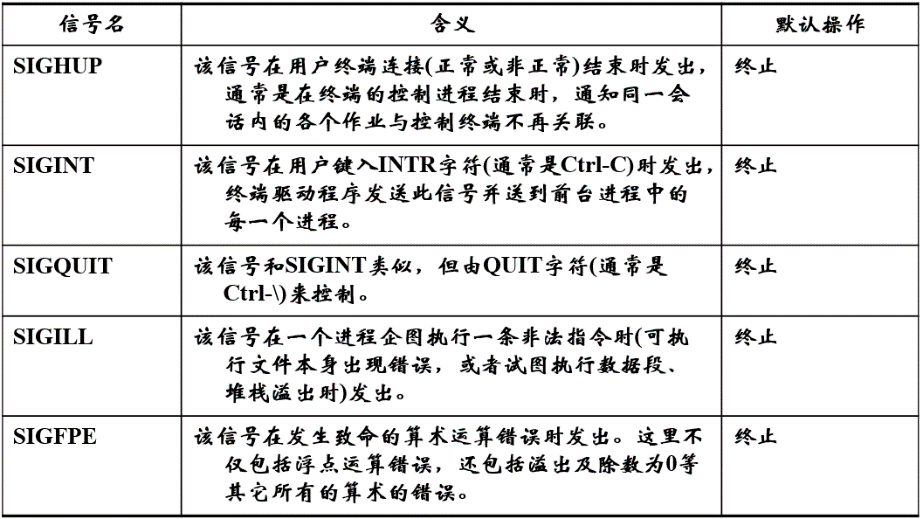
自己实现进程池的使用 直到进程池特性

##### 信号

一个进程向另一个进程通过信号传递某种讯息，接收方在接收到信号时进行相应的处理

kill –l 查看信号

kill -signame PID 给PID的进程发送一个信号：kill 9 56632 杀死PID56632进程



关于信号

1. 信号名称：系统定义，名字或者数字
2. 信号含义：系统定义，信号的作用
3. 默认处理方法：当一个进程接收到信号时默认产生的效果
4. 例如：SIGHUP 连接断开

SIGINT CTRL + C

SIGQUTT CTRL + \

SIGTSTP CTRL + Z

SIGKILL 终止进程且不能被处理

SIGSTOP 暂停进程且不能被处理

SIGALRM 时钟信号

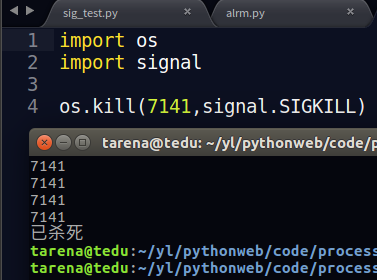
SIGCHID 子进程状态改变给父进程发送

1. Python进行信号处理
   1. os.kill(pid,sig)

功能：发送信号给某个进程

参数：pid 接收信号的进程的进程号

sig 要发送的信号



* 1. signal.alarm(sec)

功能：一定时间后向自身发送一个SIGALRM信号

参数：定时时间

\*在一个进程中只允许有一个时钟，设置第二个的时候时间就会重置



* 1. 程序执行的异步和同步

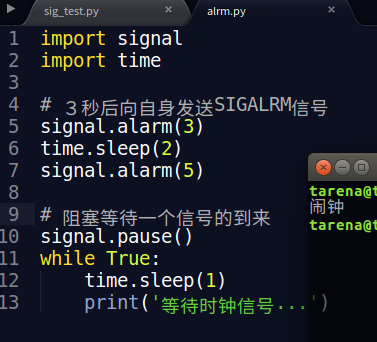
同步：按照步骤往下顺序执行

异步：在程序执行中利用内核，不影响应用层的持续执行

\*信号是唯一的异步通信方式

* 1. signal.pause()

功能：阻塞等待一个信号的发生



* 1. signal.signal(signum,handler)

功能：信号处理函数

参数：signum：要处理的信号

handler：信号的处理方法

可选值：1.SIG\_DFL 表示使用默认方法处理

2.SIG\_IGN：表示忽略这个信号

3.func：自定义函数

格式：def func（sig，frame）

sig：接收到的信号

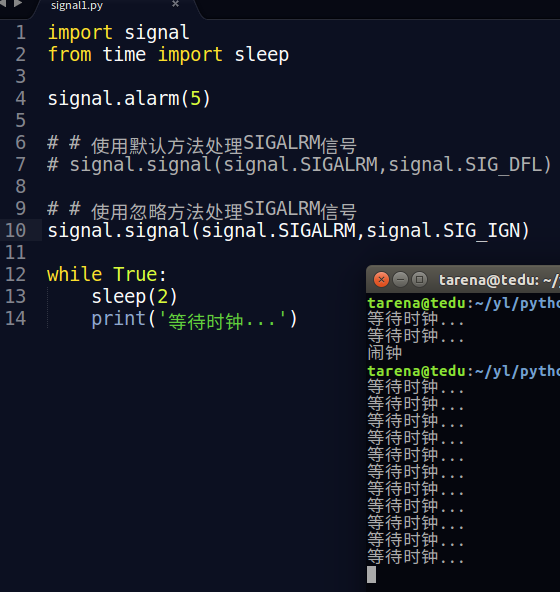
frame：信号结构对象

\*singal函数是一个异步处理函数

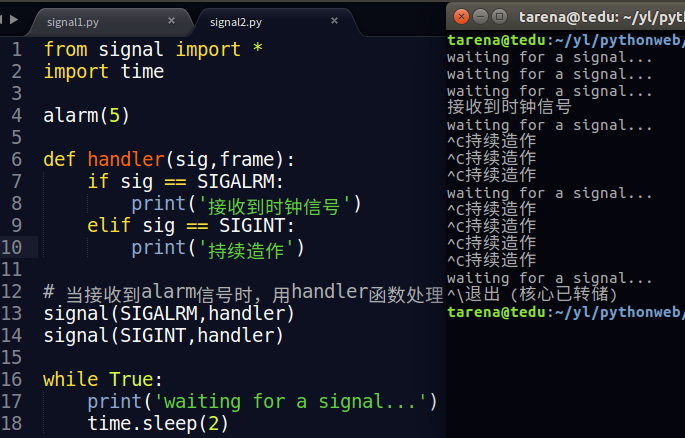
\*signal函数不能处理SIGKILL和SIGSTOP信号

\*在父进程中加上signal（SIGCHLD,SIG\_IGN），当子进程退出时会自动交由系统处理

例如：处理alarm



自定义处理函数



##### 信号量

给定一定的数量，对多个进程可见，并且多个进程根据信号的数量多少确定不同的行为

multiprocessing 🡪Semaphore（）

sem = Semaphore(num)

功能：生成信号量对象

参数：信号量的初始值

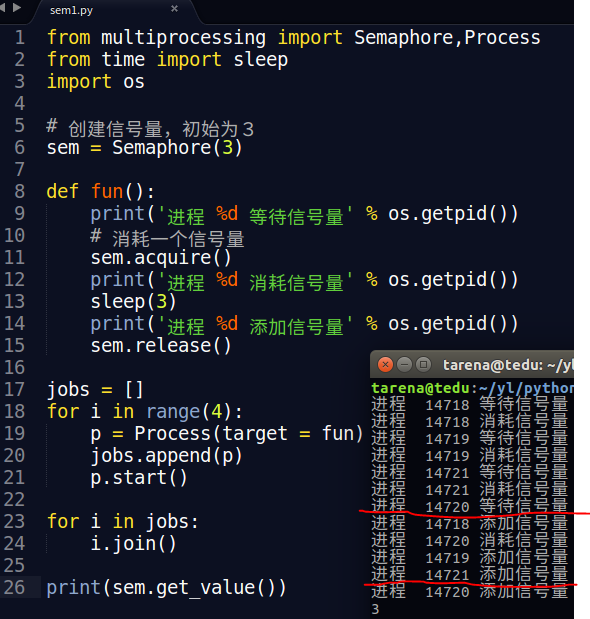
返回值：信号量对象

sem.acquire() 将信号量数量减一，信号量为0会阻塞

sem.release() 将信号量数量加一

sem.get\_value() 获取当前信号值

例如：

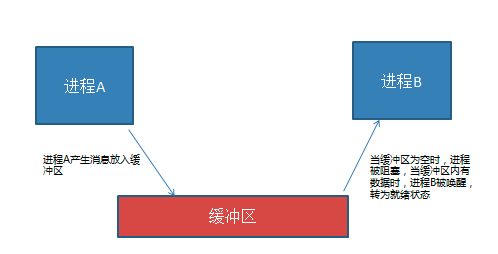


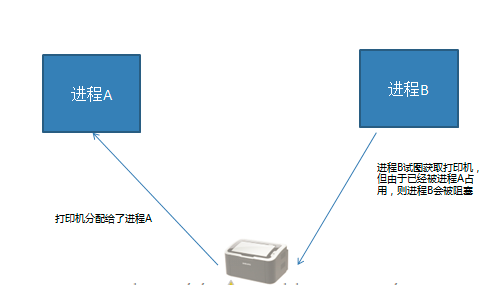
#### 同步互斥机制

目的：解决对共有资源操作产生的争夺

临界资源：多个进程或线程都能操作的资源

临界区：操作临界资源的代码段

同步：一种合作关系，为完成某个任务，多进程和多线程之间形成一种协调，按照约定或条件一次执行操作临界资源，相互告知资源使用情况

互斥：是一种制约关系，当一个进程或者线程进入临界区，会进行加锁的操作，此时，其他进程或线程再企图使用临界资源时就会阻塞，直到资源被释放才能使用 

1. Event 事件

multiprocessing 🡪Event

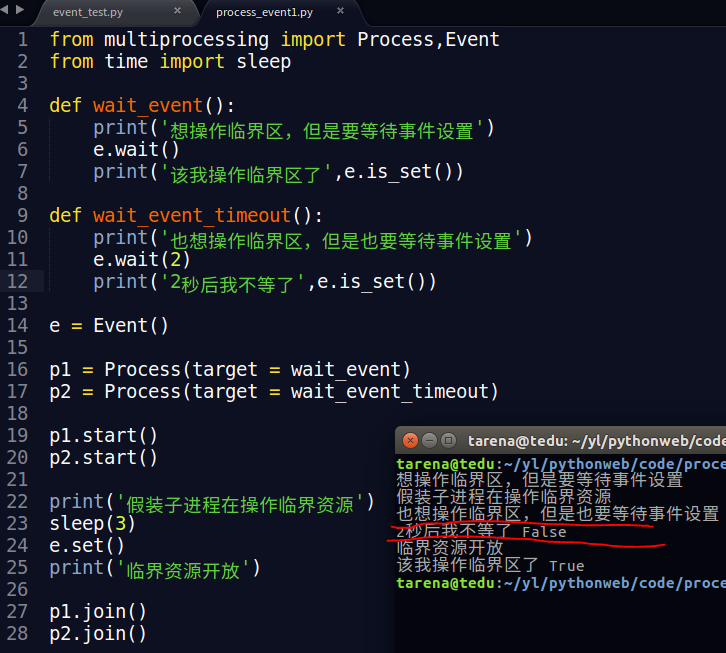
创建事件对象：e = Event()

事件阻塞：e.wait()

事件设置：e.set()，e处于设置状态，e.wait()不再阻塞

事件清除：e.clear()，e处于未设置状态，e.wait()阻塞

事件判断：e.is\_set()，判断e的当前状态



1. 锁 Lock

multiprocessing 🡪 Lock

创建对象

lock = Lock（）

lock.acquire() 上锁

lock.release() 解锁

上锁状态：此时执行acquire（）操作会阻塞

解锁状态：执行acquire（）操作为非阻塞

with lock 🡪 上锁

……..

……..

with代码段结束后自动解锁



### 二、线程

什么是线程：线程是一种多任务编程方式，可以使用计算机的多核资源。线程被称为轻量级的进程

#### 线程的特征：

1. 线程是计算机多核分配的最小单位
2. 一个进程可以包含多个线程
3. 线程也是一个运行过程，也要消耗计算机资源。多个线程共享进程的资源和空间
4. 线程也拥有自己特有的资源属性，比如指令集，TID等
5. 线程无论是创建、删除或是运行，消耗的资源都小于进程
6. 多个线程之间并行执行，执行上互不干扰

#### threading模块创建线程

1. threading.Thread()

功能：创建线程对象

参数：name 线程名称，默认为Thread-1

target 线程函数

args 元组 给线程函数位置传参

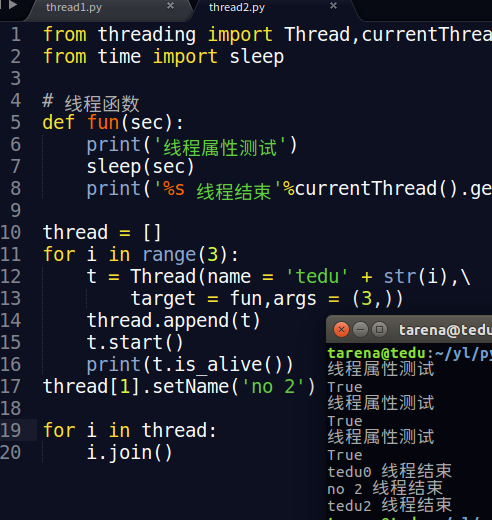
kwargs 字典 给线程函数键值传参

返回：线程对象

1. t.start() 启动线程
2. t.join([timeout]) 回收线程

##### 线程对象的属性

1. t.is\_alive() 查看线程的状态
2. t.name() 线程名称
3. t.setName() 设置线程名称
4. threading.currentThread() 获取当前线程对象
5. t.daemon属性

默认为False，主线程的结束不会影响到分支线程的执行，如果设置为True 则主线程退出，分支线程也会退出

设置方法：

t.daemon = True

t.setDaemon(True)

判断daemon属性：t.isDaemon()

\*线程daemon属性的设置在start之前

\*一般设置daemon为True后不再使用join



##### 创建自己的线程类

1. 继承Tread类
2. 运行Thread类中的\_\_init\_\_方法以获取父类原有属性
3. 重写run方法
4. 作业

创建父子进程，分别表示司机和售票员

当售票员捕捉到SIGINT信号，给司机发送SIGUSER1信号，此时司机打印‘老司机开车了’

当售票员捕捉到SIGQUIT信号，给司机发型SIGUSER2信号，此时司机打印‘车速有点快，系好安全带’

当司机扑捉到SIGTSTP信号，给售票员发送SIGUSER1信号，此时售票员打印‘到站了，请下车’

到站后，售票员先下车（子进程先退出），然后司机下车

#### 线程间通信

1. 通信方法：多个线程共用进程空间，所以进程的全局变量对进程内的线程均可见，因此线程间没有特有的通信方式，往往使用全局变量通信
2. 注意事项：线程间使用全局变量通信往往需要同步互斥机制做为通信的安全保证

##### 线程的同步和互斥

1. 线程event对象、

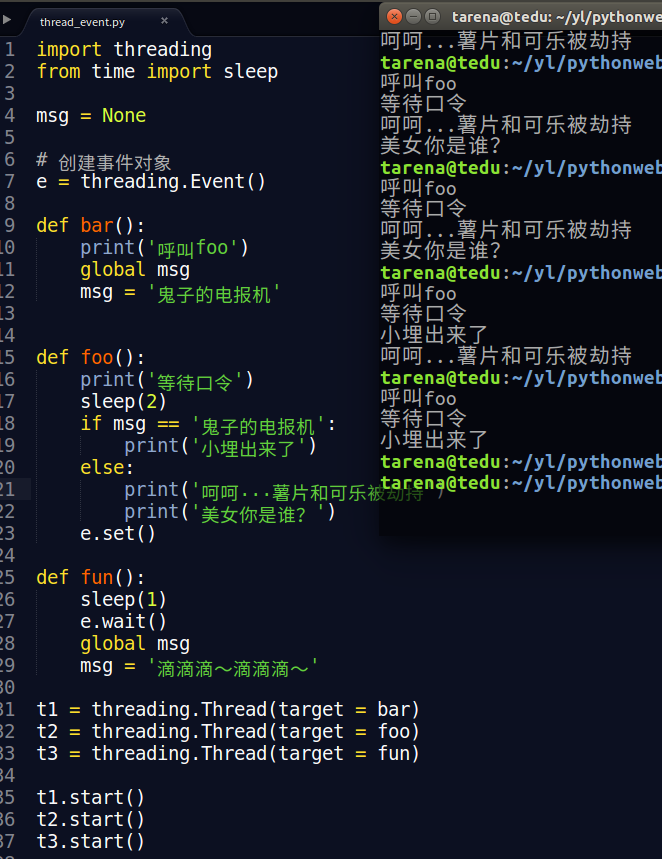
创建对象：

e = threading.Event()

e.wait([timeout]) 如果e为设置状态则不会阻塞，未设置则阻塞

e.set() 将e变为设置状态

e.clear() 将e的设置去除

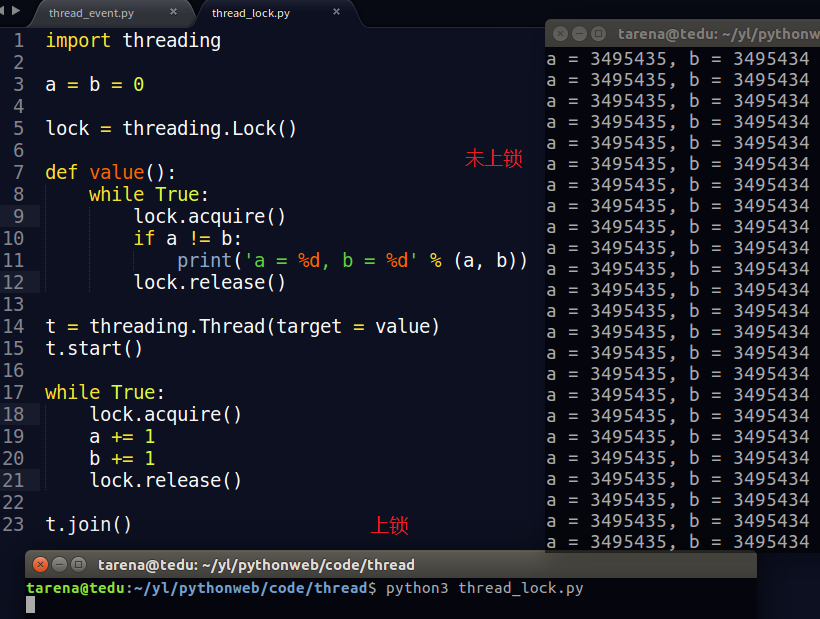


1. 线程锁

lock = threading.Lock() 创建锁对象

lock.acquire() 上锁

lock.release() 解锁

\*在一个进程中对一个线程锁重复上锁，则会阻塞

##### python线程的GIL问题（全局解释器锁）

python 🡪支持多线程🡪同步互斥🡪加锁🡪超级锁，给解释器加锁

后果：\*一个解释器，同一时刻只能解释一个线程，因此大大降低了python线程的执行效率，称为python的GIL问题

解决方案：

* 1. 尽量使用进程方式并行
  2. 不使用c作为解释器 c# java
  3. python线程适用于高延迟的IO操作，网络操作，不适合用cpu密集型。线程在遇到IO阻塞时会让出解释器和cpu

效率测试：分别在IO密集型程序和cpu密集型程序下测试单进程，多进程，多线程，

Line cpu: 8.499228477478027

Line IO: 5.998496770858765

Thread cpu: 8.477752447128296

Thread io: 5.551389217376709

Process cpu:4.72834086418152

Process io: 3.999227523803711

#### 进程和线程的区别和联系

1. 两者都是多任务编程的方式，都能够使用计算机多核
2. 进程的创建和删除要比线程消耗更多计算机资源
3. 进程空间独立，数据安全性好，有专门的进程间通信方法
4. 线程使用全局变量通信，更加简单，但是往往要与同步互斥机制共用
5. 一个进程可以包含多个线程，线程共享进程的资源空间
6. 进程线程都有自己特有的资源，如命令，属性，id等

使用场景：

1. 需要创建较多的并发，任务比较简单时，适用线程
2. 如果程序数据资源比较复杂，特别是共享资源较多时，需要考虑到线程锁的复杂性
3. 如果多个任务无关联性，不宜将其强行融入到一个进程中，适用多进程
4. python线程不适合cpu密集型程序

总结：

1. 进程线程的区别和关系
2. 进程间以什么方式通信，有什么特点
3. 同步互斥的意义，有什么方法实现同步互斥
4. 僵尸进程，进程状态，线程GIL的概念
5. 给一个情景，问选择进程还是线程，怎么做，为什么

# 三、服务器开发训练

### 服务器模型

硬件服务器：主机 集群

厂商：IBM HP 联想 浪潮

软件服务器：编写的服务端程序，依托于硬件服务器运行，提供给用户一定的软件服务

分类：webserver 🡪网络后端程序提供网络请求的后端处理和数据交互

httpserver🡪处理http请求，回复http响应

邮箱服务器🡪提供邮箱服务

文件服务器🡪提供文件的上传下载

功能：网络连接，逻辑处理，数据的交互，数据的传输，协议的实现

模型结构：c/s （客户端服务器模型）

b/s （浏览器服务器模型）

服务器目标：处理速度快，数据更安全，并发量大

硬件：更高的配置，使用集成分布的技术，更好的网络速度，更多的主机，网络安全

软件：程序占有更少的计算机资源，更稳定的运行效果，更流畅的运行速度，采用更合理的技术，处理更高的并发

服务器模型

1. 循环服务器：单进程程序，循环地接收客户端的请求，处理请求，每处理完一个请求后，再去接收处理下一个请求
   1. 优点：实现简单，占用系统资源少
   2. 缺点：无法同时连接多个客户端，当一个客户端长期占有服务器时，会形成其他客户端无法访问的情况
   3. 适用情况：任务比较短暂，udp套接字更适合循环
2. 并发服务器：同时能够处理多个客户端的任务请求，并发可分为IO并发，或者多进程、多线程并发
   1. IO并发：IO多路复用 协程

优点：资源消耗少，适用于IO类型服务器

缺点：不能监控cpu密集的情况，单线程，不能长期阻塞在消息的收发

* 1. 进程/线程并发：为每个客户端单独提供一个进程线程，处理客户端请求

优点：客户端可以长期占用服务器

缺点：消耗系统资源较多

#### 多进程并发模型

使用fork完成并发

1. 创建套接字 绑定 监听
2. 等待接收客户端请求
3. 创建子进程处理客户端请求，父进程继续接收其他客户端连接
4. 客户端退出则子进程结束

#### tftp文件服务器

\*客户端有简单的页面命令提示

\*功能包含： 1.查看服务端文件列表

2.下载其中的某个文件到本地

3.将本地文件上传到服务器文件库中

4.退出

\*服务端需求：1.处理客户端的各种请求

2.允许多个客户端同时进行操作

技术分析：tcp套接字

并发🡪多进程

### extra 文件处理函数

os.listdir(PATH):获取一个目录内的文件列表（包括隐藏文件）

os.path.isfile()：判断一个文件的类型是否为普通文件

os.path.isdir()：判断一个文件的类型是否为目录

### Thread并发服务器

优点：资源消耗少

缺点：需要注意共享该资源的争夺

python多线程收到GIL的影响

#### threading socket

实现步骤：

1. 创建套接字，绑定，监听
2. 接收客户端连接请求，创建新的线程
3. 主线程继续接收下一个客户端连接请求，分支线程处理客户端具体请求事件
4. 处理事件结束，退出线程，关闭套接字

### socket套接字服务器集成模块

#### python3 socketserver

功能：通过模块的接口完成基于多进程/多线程的tcp/udp的socket并发程序

'DatagramRequestHandler' 处理udp请求

'StreamRequestHandler' 处理tcp请求

'ForkingMixIn' 提供进程创建

'ForkingTCPServer'

'ForkingUDPServer

'TCPServer'

'UDPServer'

'ThreadingMixIn'

'ThreadingTCPServer'

'ThreadingUDPServer'

### 基于多线程并发的HTTPServer

HTTPServer的作用

1. 接收浏览器发送的http请求
2. 对http请求进行解析
3. 组织响应内容进行回发

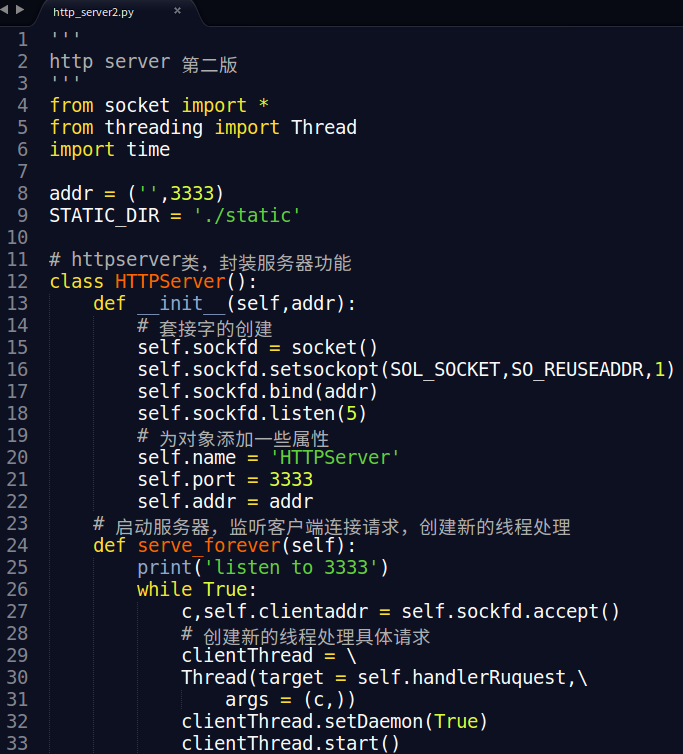
升级：

1. 使用多线程的并发可以同时处理多个客户端请求
2. 增加简单的应用程序，使浏览器不仅可以访问静态网页，也可以访问后台程序
3. 使用类对服务器功能模块进行封装

技术点：threading并发

tcp socket 套接字

http协议响应内容进行字符串拼接







### 协程

定义：纤程 微线程，协程本质只要一个线程在运行

功能原理：通过应用层，记录上下文栈区，实现在程序执行过程中的跳跃，选择可以不阻塞的部分执行，从而提高IO的执行效率

优点：资源消耗很少，无需多线程那样来回切换的开销，无需进行同步互斥操作，IO并发性好

缺点：无法利用计算机的多核资源，程序不能够一个客户端长时间占有服务器

yield 🡪协程的及基本实现关键字

安装

sudo pip3 install greenlet

sudo pip3 install gevent

##### greenlet

gr = greenlet.greenlet()

gr.switch

##### gevent

将协程事件封装为函数

gevent.spawn(func,argv)

功能：将事件变为协程事件并启动（不需要特意启动）

参数：func 传入一个函数变为协程

argv 给func函数传参

返回值：协程对象

gevent.joinall()

功能：回收协程

gevent.sleep()

功能：模拟IO阻塞的情况

from gevent import monkey

monkey.patch\_all()

功能：在导入socket之前使用，修改socket的IO设置行为



# 四、mongodb

数据存储阶段

文件管理阶段（.txt .doc .xls）

优点：数据可以长期保存，可以存储大量的数据，使用简单

缺点：数据的一致性差，数据的查找、修改不方便，数据 冗余性

数据库存储阶段

优点：数据组织结构化，降低了冗余，提高了增删改查的效率，方便扩展，方便程序调用，做自动化的处理

缺点：数据库使用往往需要特定的语句或命令进行操作，相对比较复杂

几个概念：

数据：能够输入计算机中并被识别处理的信息的集合

数据结构：研究一个数据集合中，数据之间关系的学科

数据库：按照数据结构，存储管理数据的仓库。数据库是在数据库管理和控制下在一定介质上的数据集合

数据库管理系统：数据库管理软件，用于建立维护数据库

数据库系统：由数据库和数据库管理系统等开发工具组成的集合

### 关系型数据库

优点：容易理解，逻辑类似常见表格，使用方便，都使用sql语句，sql语句非常成熟，数据的一致性高，冗余低，完整性好，技术成熟，可以使用外部链接等复杂的操作

缺点：每次都需要sql语句的解析，消耗较大，不能很好满足并发需求，特别是海量数据的爆发，关系型数据库普遍读写能力不足，关系型数据库内部每一步都要加锁的操作，以保证操作的原子性，增加了数据库负担，数据的一致性有时会导致大量的空间浪费

### 非关系型数据库（nosql）🡪not only sql

优点：高并发，读写能力强；普遍比关系型数据库容易扩展；弱化了数据结构，降低了数据的一致性

缺点：通用性差，没有sql一样的一致化的操作；操作灵活，容易混乱；没有join等操作，有的数据库不支持关系型数据库中类似事务等操作，版本更新快

### 关系型数据库和非关系型数据库的比较

1. 不是以关系模型构建数据结构，结构比较自由，不保证数据的一致性
2. 非关系型数据库弥补了关系型数据库的一些不足，能够在处理高并发，海量数据上体现优势
3. 非关系型数据库的个性化使其可以在节省空空间，提高效率方面发挥作用

### NoSql的使用情况

1. 数据的一致性要求低
2. 数据库处理海量并发
3. 数据库大小不容易估算，需要分布扩展
4. 给定的数据比较容易建立NoSql模型

NoSql的分类：

键值型数据库：Redis

列存储数据库

文档性数据库：MongoDB

图形数据库

MongoDB数据库（非关系型数据库🡪文档性数据库）

1. 由c++编写的数据库管理系统
2. 支持丰富的增删改查操作
3. 支持丰富的数据类型
4. 支持众多的编程语言接口
5. 使用方便，便于部署。在非关系数据库中属于比较成熟的数据库

### MongoDB安装

1. 自动安装：sudo apt-get install mongodb

默认安装位置：/var/lib/mongodb

配置文件位置：/etc/mongodb.conf

命令集：/usr/bin /usr/local/bin

1. 手动安装
   1. 下载MongoDB（开源）

www.mongodb.com-->get mongodb --> community server 选择想要的版本下载

* 1. 选择适合的位置解压（/usr/local /opt）,tar 解压后得到Mongodb文件夹
  2. 将MongoDB文件夹中的bin文件夹变为环境变量

PATH = $PATH:/opt/mongo…/bin（文件夹路径）

export PATH

将以上两句写入启动脚本中：/etc/rc.local

* 1. 重启系统

### MongoDB命令

1. 设置数据库的存储位置：mongod --dbpath 目录名
2. 设置数据库端口：mongod --port 端口号，如果不设置默认为27017
3. 进入mongo shell界面：mongo mongo的交互界面，用来通过mongo语句操作MongoDB数据库
4. 退出mongo shell使用quit（）或者ctrl+c
5. 组织结构：键值对🡪文档🡪集合🡪数据库

mysql 与mongodb 概念比较

mysql mongodb 含义

database database 数据库

table collection 表/集合

column field 字段/域

row document 记录/文档

index index 索引

1. 创建数据库：use databaseName

e.g. use stu

\*use 实际上是表示选择使用的数据库，如果这个数据库不存在，则表示同时创建这个数据库

\*使用use后数据库不会被马上创建，而是在插入如数据后才会创建

\*数据库命名规则：

* 1. 使用utf-8字符
  2. 不能含有空格，点 / \ ’\0’字符
  3. 长度不能超过64字节
  4. 不能和系统数据库重名
  5. 习惯用小写字母，表达数据库功能
  6. db：mongo系统全局变量，代表当前正在使用的数据库；当不用use选择任何数据库时，db表示test，此时插入数据则创建test数据

1. 查看当前系统下数据库：show dbs

\*系统数据库：admin：存放用户及其权限；local：存储本地数据；config：存储分片信息

1. 数据库的备份和恢复：
   1. 备份：mongodump –h dbhost –d dbname –o dbdir

e.g. mongodump -h 176.209.101.22 -d stu -o /home/tarena/yl/mongodb/

将‘176.209.101.22’主机上的stu数据库备份到mongodb文件夹

* 1. 恢复：mongorestore -h dbhost:port -d dbname path

e.g. mongorestore -h 176.209.101.22:27017 -d student ./stu

将‘176.209.101.22’主机上的备份的stu数据库恢复到student数据库中，不存在则会自动创建

1. 数据库的监测：
   1. mongostat

insert query，update ，delete：每秒增删改查的次数

command：每秒运行命令的次数

flushes：每秒和磁盘交互次数

vsize：使用虚拟内存

* 1. mongotop：监测每个数据库的读写时长

ns total read write

数据集合 总时长 读时长 写时长

1. 删除数据库：db.dropDatabase(): 删除db当前所代表的数据库
2. 创建集合：
   1. db.createCollection(collection\_name)

e.g. db.createCollection('class1')

* 1. 当向一个集合中插入数据的时候，如果这个集合不存在，则会自动创建

db.collectionName.insert(…)

e.g. db.class2.insert({name:'Lucy'})

\*集合命名规则：

1. 合法的utf-8字符串
2. 不能有‘\0’
3. 不能以system.开头，这是系统的保留前缀
4. 不能和关键字重复
5. 删除集合：db.collectionName.drop()

e.g. db.class4.drop() 删除集合class4

1. 重命名集合：db.collectionName.renameCollection(‘new\_name’)

e.g. db.class2.renameCollection('class0') 将集合class2重命名为class0

### 文档

mongodb中数据的组织形式🡪文档

mongo文档：以键值对的形式组成的类似于字典的数据描述形式

键：即文档的域

键的命名规则：

1. utf-8格式字符串
2. 不能使用‘\0’，通常不会使用点和$
3. 一个文档中键是不能重复的

\*文档中的键值对是有序的

\*mongodb严格地区分大小写

### 值：即文档存储的数据，支持bson类型

类型 值

整型 整数

布尔类型 true false

浮点型 小数

Array 数组

Timestamp 时间戳

Date 时间日期

Object 内部文档

Null 空值 null

Symbol 特殊字符

String 字符串

Binary data 二进制字符串

code 代码

regex 正则表达式

ObjectId ObjectId子串

ObjectId

"\_id" : ObjectId("5b28b432bf656af50a445c85")

\_id：当在mongodb中插入文档时，如果不指定\_id则会自动添加这个域，值是一个ObjectId类型数据

24位16进制数🡪保证\_id值得唯一性

前8位：文档创建时间 6位机器id 4位进程id 6位计数器

集合中的文档

1. 集合中的文档不一定都有相同的域
2. 集合中文档域的个数也不一定相同

### 集合的设计

1. 集合中的文档尽可能描述同一类数据
2. 同一类数据不要过多地分散集合存放
3. 集合中文档的层次不要包含太多

### 插入文档：db.collectionName.insert()

1. 插入单个文档：db.class0.insert({name:'Lily',age:20})

\*插入操作时，键可以不加引号

\*查看插入结果：db.class0.find()

\* db.class0.insert({\_id:3,name:'troye',age:23})，可覆盖自动生成的\_id号

\*\_id为系统自动添加主键，不能重复

1. 插入多个文档：db.collectionName.insert([{},{},{},…])

e.g. db.class1.insert([{\_id:1,name:'lea',age:'23'},{\_id:2,name:'jacob',age:25,sex:'man'},{\_id:3,name:'troye',age:23,sex:'man',muxic:'bloom'}])

1. 插入文档：save：db.collectionName.save()

e.g. db.class1.save({\_id:4,name:'kalyer',age:24,sex:'woman'})

\*如果不加\_id选项时，save和insert相同

\*如果加\_id选项，则如果此\_id值存在，则save表示修改该文档内容，如果不存在则正常插入

作业：关系型数据库和非关系型数据库的区别

复习mysql增删改查操作

练习mongo数据库的创建、删除，集合的创建、删除、文档插入

1. 获取集合对象
   1. db.getCollection(‘class0’)

e.g. db.getCollection('class0').insert({name:'阿西',age:29})

### 文档查找：db.collection.find(query,field)

1. find(query,field)
   1. 查找所有内容

db.collection.find()🡪select \* from table

* 1. 功能：查找数据

参数：

query：筛选条件，相当于mysql中的where子句，以键值对的形式给出筛选条件：db.class1.find({name:'troye'})

field：选定要展示的域，相当于要展示的字段名，以键值对的形式给出要展示或者不展示的域，值为1则表示展示，0表示不展示，例如：db.class1.find({},{name:1})

\*如果某个域设置为0，则表示不展示该域，其他域均显示，如果某个域设置为1，则表示展示该域，其他域均不显示

\*在field显示设置的时候普通域0和1不能同时出现（\_id除外）

\*\_id默认永远显示，除非设置为0

\*如果不写field则表示所有内容都显示

返回值：返回查找到的文档

1. findOne（query，field）

功能：查找符合条件的第一条文档

参数：完全同find

返回值：返回查找到的文档

e.g. db.class1.findOne({age:23},{\_id:0})

#### query的筛选功能

操作符：使用$符号注明的一个特殊字串，表达一定的含义，比如使用$lt表示小于

* 1. 比较操作符

|  |  |  |
| --- | --- | --- |
| 操作符 | 含义 | 说明/示例 |
| $eq | 等于 = | db.class1.find({age:{$eq:27}},{\_id:0}) |
| $lt | 小于 < | db.class1.find({age:{$lt:27}},{\_id:0}) |
| $lte | 小于等于 <= |  |
| $gt | 大于 > |  |
| $gte | 大于等于 >= |  |
| $ne | 不等于 != | 若无相关域，亦为不等于  db.class1.find({age:{$eq:23}}) |
| $in | 包含 in | db.class1.find({age:{$in:[30,29,23]}}) |
| $nin | 不包含 not in | db.class1.find({age:{$nin:[30,29,23]}}) |

查找年龄大于20小于27的文档：

* 1. 逻辑操作符

|  |  |  |
| --- | --- | --- |
| 操作符 | 含义 | 说明/示例 |
| $and | 逻辑与 | db.class1.find({age:{$gt:20,$lt:27 }}) |
| db.class1.find({$and:[{name:{$lt:'tom'}},{age:23}]}) |
| $or | 逻辑或 | db.class1.find({$or:[{age:{$lt:26}},{age:{$gt:28}}]}) |
| $not | 逻辑非 | db.class1.find({age:{$not:{$eq:23}}})🡪等同于db.class1.find({age:{$ne:23}}) |
| $nor | 既不…也不… | db.class1.find({$nor:[{age:17},{name:'emma'}]})  查找年龄不为17且姓名不为emma的文档 |

#### 数组值

[1,2,3,4,…]

\*数值类型可以混合

\*是有序的

#### 数组查找

1. 查看数组中是否包含某一项

db.class2.find({score:{$lt:60}},{\_id:0})

db.class2.find({score:{$lt:80,$gt:50}},{\_id:0})

\*只要数组中有任意一项符合查找条件即可被查找

1. $all：查找数组中同时包含多项的文档

db.class2.find({score:{$all:[65,92]}},{\_id:0})

1. $size：通过数组中元素个数查找

db.class2.find({score:{$size:5}},{\_id:0})

1. $slice：取数组的部分进行显示，放在field中

db.class2.find({},{\_id:0,score:{$slice:2}})

score数组中只显示前两项

db.class2.find({},{\_id:0,score:{$slice:[1,2]}})

score数组中跳过第一项显示，显示第一项后面两项

1. 其他查询方法
   1. $exists：判断一个域是否存在

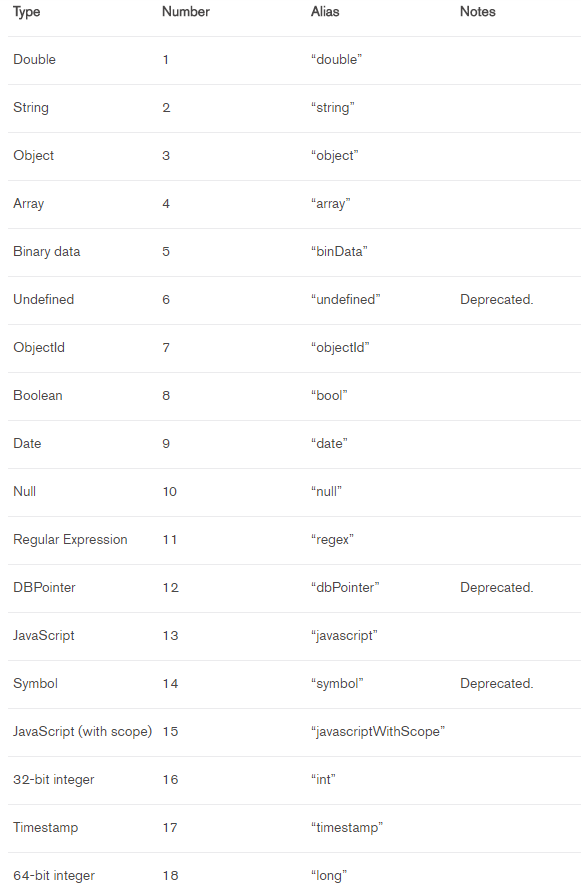
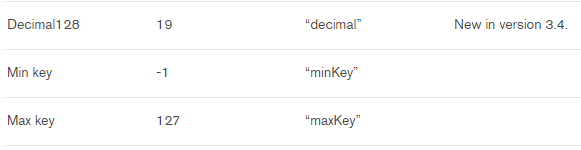
db.class1.find({sex:{$exists:true}},{\_id:0})

* 1. $mod：余数查找

db.class1.find({age:{$mod:[2,1]}},{\_id:0})

查找年龄为奇数的文档

* 1. $type：查找指定数据类型的文档（<https://docs.mongodb.com/manual/reference/operator/query/type/index.html>）

db.class1.find({age:{$type:1}},{\_id:0})  

\*如果对数据进行查找则表示查找数组内的值对应的类型

### 查找结果处理函数

1. distinct()

功能：查看一个集合中某个域的值的范围

e.g. 查看class2集合中的score的取值情况

> db.class2.distinct('score')

[ 50, 65, 88, 92, 69, 81, 87, 78, 80, 83, 89, 5, 68, 82, 85 ]

1. pretty()

功能：将查询结果格式化显示

db.class2.find().pretty()

1. limit(n)

功能：显示查找结果前n条

db.class1.find({},{}).limit(2)

1. skip(n)

功能：显示查找结果跳过前n条

db.class1.find({},{}).skip(2)

1. count()

功能：查找结果计数，最好配合聚合一起使用，否则可能计数不准

返回值：查询的结果的条数

db.class1.find({},{}).count()

1. sort({域：1/-1})

功能：对查找结果排序显示

参数：以键值对的方式给出，其中1表示按照升序排列，-1表示降序排列

按照年龄升序：db.class2.find({},{\_id:0}).sort({age:1})

按照年龄降序：db.class2.find({},{\_id:0}).sort({age:-1})

符合排序：当第一排序项相同时，按照第二排序项排序db.class2.find({},{\_id:0}).sort({age:1,score:1})

1. 函数连续使用：当函数返回的是文档集合时可以连续调用操作函数

db.class2.find({},{\_id:0}).sort({age:1}).limit(3)

### 删除文档

mysql：delete from table where…

1.mongo：db.collection.remove(query,justOne)

功能：删除文档

参数

1. query：筛选要删除的文档，相当于where，用法同查找操作
2. justOne：布尔值，默认为false，表示删除所有符合条件的文档，如果赋值为true，则只删除第一条符合条件的文档
3. 示例：db.class1.remove({sex:{$exists:true}})

db.class1.remove({sex:{$exists:true}},true)

2.db.collection.remove({})：表示删除集合中所有文档

示例：db.class0.remove({})

### 练习

1. 创建数据库 名字 grade

2. 数据库中创建集合名字 class

3. 集合中插入若干文档，文档格式如下

｛name:'zhang',age:10,sex:'m',hobby:['a','b']｝

hobby 可以是 ： draw sing dance basketball football pingpong computer

4. 查找练习

查看班级所有人信息

查看班级中年龄为8岁的学生信息

查看年龄大于10岁的学生信息

查看年龄在8-11岁之间的学生信息

db.class.find({age:{$gt:8,$lt:11}},{\_id:0})

找到年龄为9岁且为男生的学生

db.class.find({age:9,sex:'m'},{\_id:0})

找到年龄小于7岁或者大于11岁的学生

db.class.find({$or:[{age:{$lt:7}},{age:{$gt:11}}]},{\_id:0})

找到年龄是8岁或者11岁的学生

db.class.find({$or:[{age:8},{age:11}]},{\_id:0})

找到有两项兴趣爱好的学生

db.class.find({hobby:{$size:2}},{\_id:0})

找到兴趣爱好中有draw的学生

db.class.find({hobby:'draw'},{\_id:0})

找到喜欢画画和跳舞的学生

db.class.find({$and:[{hobby:'draw'},{hobby:'dance'}]},{\_id:0})

db.class.find({hobby:{$all:['draw','dance']}},{\_id:0})

统计兴趣有三项的学生人数

db.class.find({hobby:{$size:3}},{\_id:0}).count()

找出本班年龄第二大的学生

db.class.find({},{\_id:0}).sort({age:-1}).skip(1).limit(1)

查看学生的兴趣范围

db.class.distinct('hobby')

找到班级中年龄最大的三个学生

db.class.find({},{\_id:0}).sort({age:-1}).limit(3)

5. 删除所有年龄大于12或者小于6岁的学生

db.class.remove({$or:[{age:{$gt:12}},{age:{$lt:6}}]})

### 修改数据（修改器）

mysql:update table set…where…

mongo:db.collection.update(query, update,upsert,multi)

功能：修改文档

参数：query：筛选要修改的文档，相当于where子句，用法同查找

update：将筛选的文档修改的内容，相当于set，需配合修改操作符使用

upsert：布尔值，默认为false，表示如果query的文档不存在，则无法修改，如果设置为true，则表示如果筛选的文档不存在，则根据修改内容插入一条文档

multi：布尔值，默认为false，表示如果有多条文档符合筛选条件，只修改第一条，如果修改为true，则表示全部修改

示例：1.db.class1.update({name:'lea'},{$set:{age:23,hobby:’music’}}),可同时修改多个域的值

2.db.class1.update({name:'lea'},{$set:{age:23},true} 如果没有符合条件的文档，则插入

3.db.class1.update({name:{$in:['lea','emma']}},{$set:{age:23},false,true},如果有多条符合条件的文档，则全部修改

#### 添加域

db.class1.update({},{$set:{sex:'w'}},false,true)，给所有符合条件的文档添加sex域

#### 删除域（$unset）

db.class1.update({name:'jerry'},{$unset:{sex:0}})，删除sex域，后面数字0或1都表示删除

#### 修改域名（$rename）

db.class1.update({},{$rename:{sex:'gender'}},false,true)，将符合条件的所有文档的sex域名修改为gender

#### （$setOnInsert）

如果第三个参数为true并插入新的文档，则作为插入文档的补充内容

db.class1.update({name:'jim'},{$set:{age:19},$setOnInsert:{gender:'m',tel:1233123}},true)，如果插入了新的文档，则将setOnInsert中的内容也插入到文档中

\*update参数中可以写多个修改器

#### $inc加减修改器

db.class1.update({name:'jerry'},{$inc:{age:1}})，给jerry的年龄加一

\*操作数可以是正数、负数、小数

#### $mul乘法修改器

db.class1.update({name:'jerry'},{$mul:{age:2}})，将jerry的年龄乘2

\*操作数可以是正数、负数、小数

#### $min

如果筛选的文档指定的值小于min则不修改，如果大于min给定的值则修改为min值

db.class1.update({},{$min:{age:25}},false,true)，将年龄大于25的文档的年龄修改为25

#### $max

如果筛选的文档指定的值大于max值则不修改，小于max给定的值则修改为max值

db.class1.update({},{$max:{age:20}},false,true)，将年龄小于20的文档年龄修改为20

#### 数组修改器

1. $push：向数组中添加一项，添加到末尾

db.class2.update({name:'小刚'},{$push:{score:10}})，给小刚添加一项为10的成绩

1. $pushAll：向数组中添加多项，添加到末尾

db.class2.update({name:'小明'},{$pushAll:{score:[5,10]}})

1. $pull：从数组中删除一项

db.class2.update({name:'小明'},{$pull:{score:5}})

1. $pullAll：从数组中删除多项

db.class2.update({name:'小明'},{$pullAll:{score:[10,50]}})

1. $each：对多个值进行逐一操作

db.class2.update({name:'小明'},{$push:{score:{$each:[10,5]}}})

1. $position：指定插入位置，需要搭配$each使用

db.class2.update({name:'小红'},{$push:{score:{$each:[78],$position:1}}})

1. $sort，对数组进行排序,1为升序，-1为降序

db.class2.update({name:'小明'},{$push:{score:{$each:[],$sort:1}}})

1. $pop，弹出一项，1为弹出最后一项，-1为弹出第一项

db.class2.update({name:'小明'},{$pop:{score:1}})

1. $addToSet：向数组中添加一项，不能和其他的重复

db.class2.update({name:'小明'},{$addToSet:{score:98}})

### 时间类型

mongo中存储时间格式：ISODate()

1. new Date()：自动生成当前时间

db.class0.insert({title:'mofa',date:new Date()})

1. ISODate()：生成当前时间

db.class0.insert({title:'ha',date:ISODate()})

1. Date()：将生成的当前时间变为字符串

db.class0.insert({title:'gou',date:Date()})

ISODate()：

功能：生成mongo时间存储类型

参数：参数为指定时间，不指定即为当前时间

“2018-1-1 12:10:34”

“20180101 12:10:34”

“20181102”

示例：>>db.class0.insert({title:'xu',date:ISODate('20180621 14:20:20')})

{ "title" : "xu", "date" : ISODate("2018-06-21T14:20:20Z") }

1. valueOf：时间戳

>>db.class0.insert({title:'yi',date:ISODate().valueOf()})

{ "title" : "yi", "date" : 1529562518894 }

### null

1. 如果某个域存在却没有值，可以设置为null

> db.class0.insert({title:'miao',price:null})

{ "title" : "miao", "price" : null }

1. 某个域不存在，可以使用null来匹配

> db.class1.find({gender:null},{\_id:0})

{ "name" : "jerry", "age" : 25 }

### Object 内部文档

文档内某个域的值还是文档，则这个值为内部文档

当需要使用内部文档某个域时，可以使用外部文档引用内部文档的方式，但是注意在使用时需要添加引号

1.> db.class3.find({'book2.title':'css'},{\_id:0})

{ "book2" : { "title" : "css", "price" : 64, "author" : "Dive" } }

2. db.class3.update({'book3.title':'java'},{$set:{'book3.price':53}})，修改book3的价格

### 数组的下标引用

使用数组时，可以使用数组域.数组序列下标的方式使用数组中的具体某一个元素

1.db.class2.update({name:'小明'},{$set:{'score.0':87}})，修改数组中第0项的值

2.db.class2.find({'score.0':65},{\_id:0})，通过数组中的第0项查找

### 查找结果的有序性

可以对find查找结果使用[]引用序列下标的方式获取查找结果的第几项

db.class2.find({},{\_id:0})[0]，显示查找结果的第0项

### 索引

指建立指定的键值及所在文档中存储位置的对照清单。使用索引可以方便我们快速查找，减少遍历次数，提高查找效率

1. mongo如何创建索引

ensureIndex()

功能：创建索引

参数：索引域，索引类别，索引选项

db.class1.ensureIndex({name:1})

\*1表示正向索引，-1表示逆向索引

1. 查看集合索引

db.class1.getIndexes()

\_id域是由系统自动为集合创建的索引

1. 自定义索引名称

通过第二个参数中name选项

db.class1.ensureIndex({'age':1},{name:'myindex'})

1. 删除索引
   1. dropIndex()

功能：删除索引

参数：要删除的索引，可以是索引的名称或者索引的键值对

db.class1.dropIndex({'name':1})

db.class1.dropIndex('myindex')

* 1. dropIndexes()

功能：删除除了\_id外的所有索引

db.class1.dropIndexes()

1. 索引类型
   1. 复合索引：根据多个域创建一个索引

db.class1.ensureIndex({'name':1,'age':-1})

* 1. 数组和子文档索引

如果对某个域的值为数组或者子文档的域创建索引，则通过数组或者子文档进行查找时也是索引查找

db.class2.find({‘score.0’:15},{\_id:0})

如果对score创建了索引，则该查找也为索引查找

* 1. 唯一索引

创建索引时希望集合中创建索引的域的值均不重复

db.class1.ensureIndex({name:1},{unique:true})

\*当对某个域创建了唯一索引则无法插入重复值

* 1. 稀疏索引/间隙索引

只针对有\_id域的文档创建索引表，如果某个文档没有该域则不会在索引表中插入内容

db.class1.ensureIndex({gender:1},{sparse:true})

### 聚合操作

对文档的筛选结果进行整理统计（更高级的查找）

db.collection.aggregate()

功能：聚合函数，完成聚合操作

参数：聚合条件🡪聚合操作符

#### 聚合操作符

1. $group：分组聚合，需要配合具体的分组操作符使用
2. $sum:求和

1. >db.class1.aggregate({$group:{\_id:'$age',num:{$sum:1}}})

{ "\_id" : 25, "num" : 2 }

{ "\_id" : 20, "num" : 4 }

num为统计结果的域名，$sum表示对统计进行的操作

2. > db.class1.aggregate({$group:{\_id:'$gender',num:{$sum:'$age'}}})

{ "\_id" : "m", "num" : 20 }

{ "\_id" : null, "num" : 25 }

{ "\_id" : "w", "num" : 85 }

1. $avg：求平均数

> db.class1.aggregate({$group:{\_id:'$gender',avg\_age:{$avg:'$age'}}})

{ "\_id" : "m", "avg\_age" : 22.5 }

{ "\_id" : "w", "avg\_age" : 21.25 }

1. $max：求最大值

> db.class1.aggregate({$group:{\_id:'$gender',max\_age:{$max:'$age'}}})

{ "\_id" : "m", "max\_age" : 25 }

{ "\_id" : "w", "max\_age" : 25 }

1. $main：求最小值

> db.class1.aggregate({$group:{\_id:'$gender',min\_age:{$min:'$age'}}})

{ "\_id" : "m", "min\_age" : 20 }

{ "\_id" : "w", "min\_age" : 20 }

1. $project：用于修改文档的显示效果

1. > db.class1.aggregate({$project:{\_id:0,name:1,age:1}})

2.> db.class1.aggregate({$project:{\_id:0,Name:'$name',Age:'$age'}})

1. $match：过滤数据

> db.class1.aggregate({$match:{age:{$gt:18}}})

1. $limit：显示前几条文档

db.class1.aggregate({$limit:3})

1. $skip：跳过几条文档

db.class1.aggregate({$skip:3})

1. $sort：排序

db.class1.aggregate({$sort:{age:-1}})

#### 聚合管道

将前一个聚合操作符得到的结果给后一个聚合操作继续使用

db.collection.aggregate([聚合1，聚合2])

示例：1.对女性年龄排序

> db.class1.aggregate([{$match:{gender:'m'}},{$project:{\_id:0}},{$sort:{age:-1}}])

{ "name" : "matt", "age" : 30, "gender" : "m" }

{ "name" : "jacob", "age" : 27, "gender" : "m" }

{ "name" : "troye", "age" : 23, "gender" : "m" }

2.查找重名的名字

> db.class1.aggregate([{$group:{\_id:'$name',num:{$sum:1}}},{$match:{num:{$gt:1}}}])

{ "\_id" : "amy", "num" : 2 }

### 固定集合

mongodb中可以创建大小固定的集合，称之为固定集合

特点：插入速度快，顺序查找速度快

能够淘汰早期的数据

可以控制集合的空间大小

使用：临时缓存、日志处理

创建：db.createCollection(collectionName,{cappde:true,size:10000,max:10000})

cappde:true：表示创建的集合是固定集合

size：表示集合的大小，单位kb

max：表示最多存放的文档个数

### 文件存储

文件存储数据库的方式

1. 存储路径：将文件在本地的路径以字符串的形式存储到数据库
   1. 优点：节省数据库空间
   2. 缺点：当数据库或者文件位置发生变化时需要修改数据库内容
2. 存储文件本身：将文件以二进制（数据库指定编码）的形式存入到数据库中
   1. 优点：数据库在，文件就在
   2. 缺点：占用数据库的空间大，文件较大的时候，由于数据库都是结构化地存储数据，所以对大文件的处理效率较低

mongo存储大文件的方案：GridFS存储大文件（超过16M的文件为大文件）

GridFS方案解释

在mongodb一个数据库中创建两个集合，共同完成文件的存储

fs.files：存储文件的相关信息（文件名、类型…）

fs.chunks：分块存储文件的实际内容

如何存储：mongfiles –d dbname put file（Linux终端）

数据库 要存储的文件

\*数据库不存在则自动创建

\*对于同一个文件 fs.files的\_id值等于fs.chunks中的files\_id域的值

提取文件：mongfiles –d dbname get file（Linux终端）

Grid的优缺点

优点：存储方便，提供较好的存命令，方便移植

缺点：读写效率低

### 游标

通过获取操作数据

操作步骤

1. 创建mongodb数据库连接对象

conn = pymongo.MongoClient(‘localhost’,27017)

1. 生成要操作的数据库对象

db = conn.stu或

db = conn.[‘stu’]

1. 获取集合对象

myset = db.class1或myset = db[‘class1’]

1. 通过集合对象操作mongodb数据库

增、删、改、查、索引、聚合、文件操作

\*注意python中操作数据库时，域名必须加引号

* 1. 插入数据：

insert()：参数用法同mongoshell中insert

insert\_many()

insert\_one

save()

* 1. 查找功能

find()

功能：查找数据库内容

参数：同mongo shell find()

返回值：返回一个结果游标，命名为cursor

\*在pymongo中使用操作符的方法和在mongoshell中一样，只需要加引号以字符串的方式给出

\*当游标使用了next或for取值后就不能再进行limit、skip或者sort操作了

cursor属性函数

* + - 1. next()
      2. limit()
      3. skip()
      4. count()
      5. sort()
         1. pymongo：sort([(‘age’,-1),(‘name’,1)])
         2. mongo shell：sort({age:-1,name:1})

find\_one()

* 1. 修改操作

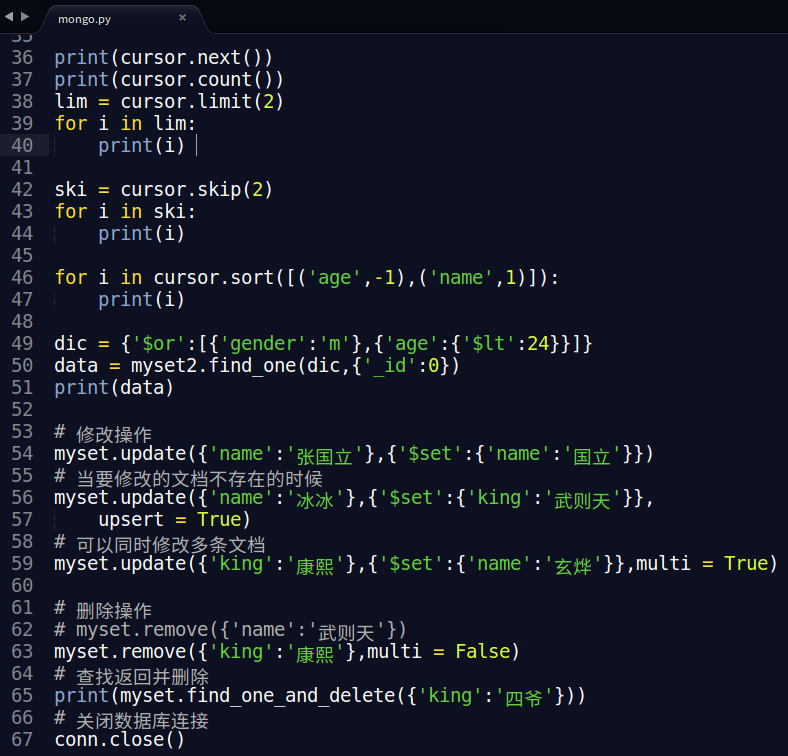
update(query,update,upsert=False,multi=False)

* 1. 删除操作

remove(query,multi = True)

multi默认True，表示删除所有符合条件的数据，设置为False表示只删除一条

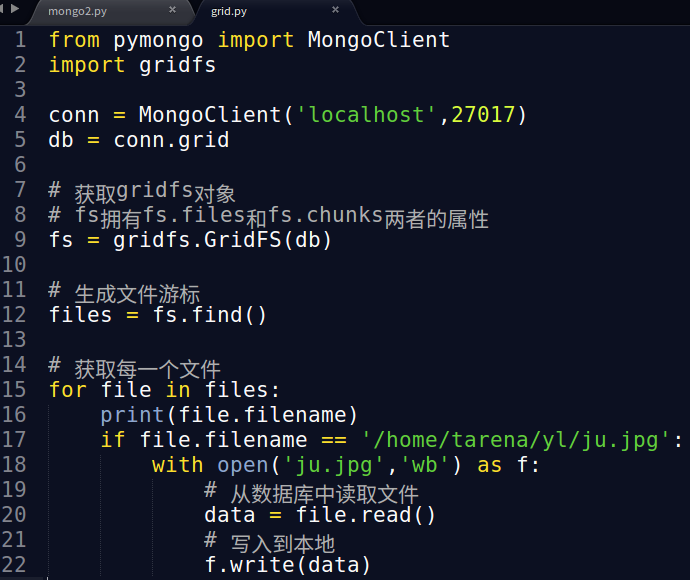


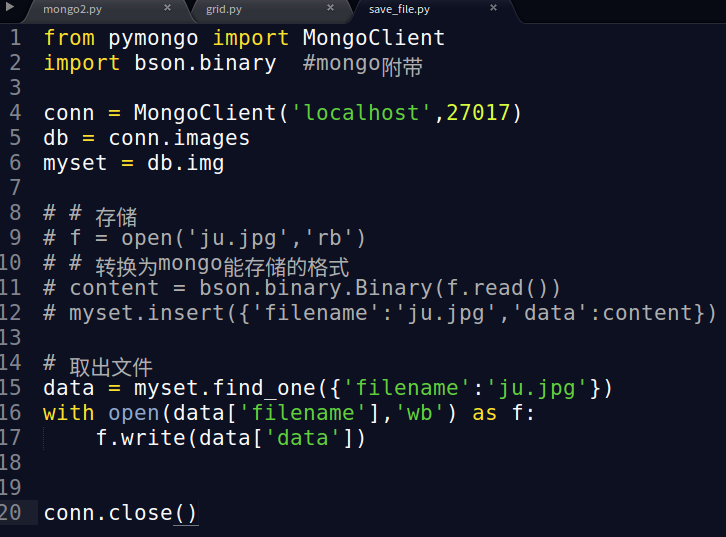


* 1. 索引操作



* 1. 文件操作





1. 关闭数据库连接

conn.close()

作业：总结mongo和mysql的使用区别

复习tftp和聊天室项目

对mongo使用pymongo改写之前的grade练习

# 五、正则表达式

### 动机：

1. 文本处理已经成为计算机的常见工作之一
2. 对文本内容的搜索，定为，提取逻辑比较复杂
3. 为了快速方便地解决上述问题，产生了正则表达式技术

定义：即文本的高级匹配模式，提供搜索、替代等功能，本质是由一系列特殊符号和字符组成的字串，这个字串即是正则表达式。这个表达式描述了字符和字符的重复行为，可以匹配一类特征的字符串

目标：1.熟练掌握正则表达式符号；2.能够编写一定程度的正则表达式和理解较难的正则；3.能使用python操作正则表达式

### 特点

1. 方便进行检索修改的文本操作
2. 支持语言众多
3. 使用灵活多样

### 元字符

python🡪re模块 处理正则表达式

1. re.findall(pattern,string)

功能：使用这种不都是匹配字符串

参数：pattern 正则表达式

string 目标字符串

返回值：返回一个列表，列表中为匹配到的内容

元字符（即正则表达式中有特殊含义的符号）

1. 普通字符

元字符：a b c

匹配规则：匹配相应的字符

In [2]: re.findall('你好','小平，你好')

Out[2]: ['你好']

In [3]: re.findall('abc','abcfgsgsabc')

Out[3]: ['abc', 'abc']

1. 或

元字符：|

匹配规则：匹配|两边任意一本正则表达式的内容

In [4]: re.findall('ab|cd','abccdsgsabc')

Out[4]: ['ab', 'cd', 'ab']

In [5]: re.findall('ab|bc','abccdsgsabc')

Out[5]: ['ab', 'ab']

1. 匹配单一字符

元字符：.

匹配规则：匹配除换行外的任意字符

In [6]: re.findall('f.o','foo is not fao')

Out[6]: ['foo', 'fao']

1. 匹配开始位置

元字符：^

匹配规则：匹配字符串的开头位置

^hello---->hello world（匹配以hello开头的字符串）

In [7]: re.findall('^hello','hello world')

Out[7]: ['hello']

In [8]: re.findall('^hello','lea hello')

Out[8]: []

1. 匹配结束位置

元字符：$

匹配规则：匹配目标字符串的结束位置

py$---->hello.py（匹配以py结尾的字符串）

In [9]: re.findall('py$','python')

Out[9]: []

In [10]: re.findall('py$','hello.py')

Out[10]: ['py']

1. 匹配重复

元字符：\*

匹配规则：匹配前面的正则表达式0次或多次

fo\* ----> foo fooooooo f

In [11]: re.findall('ab\*','abababcadgadg')

Out[11]: ['ab', 'ab', 'ab', 'a', 'a']

1. 匹配重复

元字符：+

匹配规则：匹配前面的正则表达式一次或多次

In [16]: re.findall('.+py','hello.py')

Out[16]: ['hello.py']

In [19]: re.findall('.+py','hello.python')

Out[19]: ['hello.py']

1. 匹配重复

元字符：?

匹配规则：匹配前面的正则表达式0次或1次

ab?---->ab a

In [22]: re.findall('ab?','abbbabaa')

Out[22]: ['ab', 'ab', 'a', 'a']

1. 匹配重复

元字符：{n}

匹配规则：匹配指定的重复次数

ab{3}---->abbb

In [25]: re.findall('ab{3}','abbbbbabaa')

Out[25]: ['abbb']

1. 匹配重复

元字符：{m,n}

匹配规则：匹配前面的正则表达式m-n次

ab{3,5}---->abbb abbbb abbbbb

In [26]: re.findall('ab{3,5}','abbbbbabaa')

Out[26]: ['abbbbb']

1. 匹配字符集合

元字符：[字符集]

匹配规则：匹配括号内的任意一个字符

[abc123d]---->a b c 1 2 3 d

[a-z]

[A-Z]

[0-9]

[123a-zA-Z]

In [27]: re.findall('[a-c]','abhgjfbabaa')

Out[27]: ['a', 'b', 'b', 'a', 'b', 'a', 'a']

In [31]: re.findall('[\_0-9a-zA-Z]+','hello world 123')

Out[31]: ['hello', 'world', '123']

In [32]: re.findall('[\_0-9a-zA-Z]','hello\_world123')

Out[32]: ['h', 'e', 'l', 'l', 'o', '\_', 'w', 'o', 'r', 'l', 'd', '1', '2', '3']

1. 匹配字符集合

元字符：[^字符集]

匹配规则：匹配除了括号中字符集的任意一个字符

[^abc] 匹配除了abc之外的任意一个字符

In [36]: re.findall('[^hello]','hello.py')

Out[36]: ['.', 'p', 'y']

1. 匹配任意（非）数字字符

元字符：\d：数字字符 \D：非数字字符

匹配规则：\d匹配任意数字字符（[0-9]），\D匹配任意非数字字符[^0-9]

In [38]: re.findall('1\d{10}','18090563707566')

Out[38]: ['18090563707']

In [39]: re.findall('\D+','南京 300000')

Out[39]: ['南京 ']

1. 匹配（非）普通字符（数字，字母，下划线为普通字符）

元字符：\w \W

匹配规则：\w：普通字符（[\_0-9a-zA-Z]） \W：非普通字符（^[\_0-9a-zA-Z]）

In [42]: re.findall('\w+','money$1')

Out[42]: ['money', '1']

\*\*：\w可以匹配普通的utf-8字符（比如汉字）

In [45]: re.findall('\w+','money 钱')

Out[45]: ['money', '钱']

In [43]: re.findall('\W+','money$1')

Out[43]: ['$']

1. 匹配（非）空字符（空格，\r,\t,\n,\0等）

元字符：\s \S

匹配规则：\s 空字符 \S 非空字符

In [47]: re.findall('\s+','hello world\r\n')

Out[47]: [' ', '\r\n']

In [48]: re.findall('\S+','hello world\r\n')

Out[48]: ['hello', 'world']

1. 匹配起止位置

元字符：-A -Z

匹配规则：-A：起始位置（^） -Z：结束位置（$）

In [58]: re.findall('\Ahello\Z','hello')

Out[58]: ['hello']

\* 绝对匹配：使用开头和结尾位置的元字符将正则表达式放在其中，则目标字符串仅有正则匹配内容时才能匹配上

1. 匹配（非）单词边界位置（普通字符和非普通字符的交界处为单词边界）

元字符：\b \B

匹配规则：\b：单词边界位置 \B：非单词边界位置

In [72]: re.findall(r'\bis','this is a test')

Out[72]: ['is']

In [73]: re.findall(r'\Bis','this is a test')

Out[73]: ['is']

In [74]: re.findall(r'成都\b','成都，成都的街头走一走')

Out[74]: ['成都']

元字符总结

1. 匹配单个字符：a . \d \D \w \W \s \S [..] [^..]
2. 匹配重复性：\* + ? {n} {m,n}
3. 匹配位置：^ $ \A \Z \b \B
4. 其他：| () \

### 正则表达式转义

正则表达式中的特殊符号：. \ $ ^ [] {} \* ?

In [80]: re.findall('.+\?$','How are you?')

Out[80]: ['How are you?']

### raw字符串

又名原生字符串，字符串内容就是本身，不进行任何转义

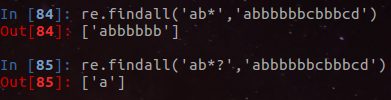
用法：r”hello”

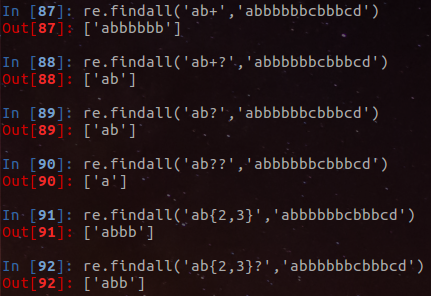
贪婪与非贪婪

贪婪模式：正则表达式中的重复匹配默认总是尽可能地多向后匹配内容（\* + ? {m,n}）

非贪婪模式：尽可能少地匹配内容

贪婪---->非贪婪模式 \*? +? ?? {m,n}?





### 正则表达式分组

使用（）可以为一个正则表达式建立子组，子组并不会影响正则表达式的整体匹配内容，子组可以看做是一个内部的整体部分

子组的作用

1. 子组可以改变正则表达式的重复行为
2. 子组在某些操作中可以被单独提取出来

In [103]: re.search(r'(cd)+','abcdcdcdcdef').group()

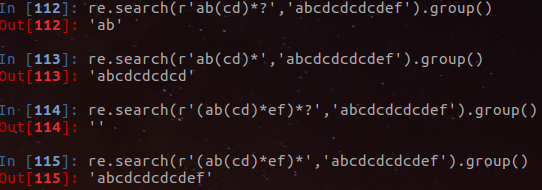
Out[103]: 'cdcdcdcd'

In [104]: re.search(r'(cd)+','abcdcdcdcdef').group(1)

Out[104]: 'cd'

子组的使用注意事项

1. 一个正则表达式中可以有多个子组，从外到内，从左到右分别是第一第二第三…子组
2. 子组不存在交叉的情况
3. 非贪婪模式只针对子组为末尾的情况有用



### 捕获组和非捕获组（命名组和非命名组）

格式：(?P<name>pattern)

In [118]: re.search(r'ab(?P<name1>cd)\*','abcdcdcdcdef').group()

Out[118]: 'abcdcdcdcd'

In [124]: re.search(r'\d{17}(\d|x)','51132519900423415x').group()

Out[124]: '51132519900423415x'

1. 很多编程接口可以直接通过名字获取子组匹配内容
2. 命名组可以被重复调用

格式：(?P = name)

In [121]: re.search(r'ab(?P<name1>cd)\*.\*(?P=name1)','abcdcdcdcdefcdcdcd').group()

Out[121]: 'abcdcdcdcdefcdcdcd'

正则表达式使用要求

1. 唯一性：正则表达式只能匹配目标字符串，而不能匹配其他内容
2. 准确性：尽可能全面地考虑目标类别的字符串特征，做到不遗漏

### re模块

1. regex = re.compile(pattern,flags = 0)

功能：生成正则表达式对象

参数：pattern：正则表达式

flags：功能标志位，提供更丰富的筛选功能

返回值：正则表达式对象

1. re.findall(pattern,string,flags)

功能：查找正则表达式匹配内容

参数：pattern：正则表达式

string：目标字符串

flags：功能标志位

返回值：将匹配的内容放入到一个列表返回，如果有子组，只能返回子组匹配到的内容

regex.findall(string=None,pos=0,endpos=999)

功能：查找正则表达式匹配内容

参数：string：目标字符串

pos：匹配目标字符串的起始位置

flags：匹配目标字符串的结束位置

返回值：将匹配的内容放入到一个列表返回，如果有子组，只能返回子组匹配到的内容

1. re.split(pattern,string,flags=0)

功能：通过正则表达式分割字符串

参数：pattern：正则表达式

string：目标字符串

返回值：返回分割后的字符串列表

1. re.sub(pattern,replaceStr,string,max,flags)

功能；替换正则表达式匹配到的内容

参数：pattern：正则表达式

replacestr：要替换的内容

string：目标字符串

max：最多替换几处

返回值：返回替换后的字符串

1. re.subn(pattern,replaceStr,string,max,flags)

功能；替换正则表达式匹配到的内容

参数：pattern：正则表达式

replacestr：要替换的内容

string：目标字符串

max：最多替换几处

返回值：返回替换后的字符串和实际替换的个数

1. re.finditer(pattern,string,flags)

功能：使用正则表达式匹配内容

参数：pattern：正则表达式

string：目标字符串

返回值：返回匹配到的内容的迭代器对象

1. re.fullmatch(pattern,string,flags)

功能：完全匹配目标字符串

参数：pattern：正则表达式

string：目标字符串

返回值：match对象，匹配不到返回None

1. re.match (pattern,string,flags)

功能：匹配目标字符串的开头

参数：pattern：正则表达式

string：目标字符串

返回值：match对象，匹配不到返回None

1. re.search (pattern,string,flags)

功能：匹配目标字符串

参数：pattern：正则表达式

string：目标字符串

返回值：match对象，匹配不到返回None

1. 正则对象的其他属性
   1. flags：标志位
   2. pattern：正则表达式
   3. groups：有多少个子组
   4. groupindex：捕获组形成的字典，组名为键，第几组为值

### match对象属性变量

pos：目标字符串的开头位置

endpos：目标字符串结束位置

re：正则表达式对象

string：目标字符串

lastgroup：最后一组名字

lastindex：最后一组是第几组

### match对象属性方法

start()：获取匹配到的内容的开始位置

end()：获取匹配到的内容的结束位置

span()：获取匹配到的内容的开始和结束位置的元组

group(n=0)：

功能：获取match对象对应匹配到的内容

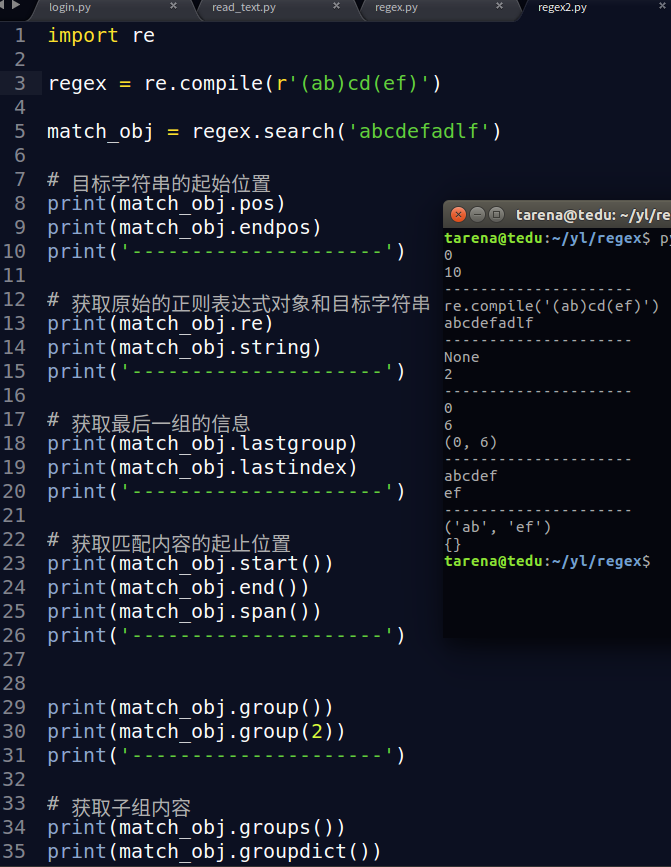
参数：默认为0表示获取正则表达式整体的匹配内容，

如果赋值1,2,3…则表示获取某个子组的匹配内容

返回值：返回匹配字符串

groups()：获取所有子组匹配内容

groupdict()：将所有捕获组内容形成一个字典



flags参数

re.compile re.findall re.search re.match re.finditer re.fullmatch re.sub

re.subn re.split

作用：辅助正则表达式，丰富匹配内容

A ==ASCII

S == DOTALL 元字符.可以匹配\n

I ==IGNORECASE 忽略大小写

L ==LOCALE

M == MULTILINE 元字符^ $可以匹配每一行的开头、结尾位置

T == TEMPLATE

U == UNICODE

X == VERBOSE 可以给正则添加注释

同时使用多个flag用|隔开



# 六、第二阶段项目训练

### 项目

1. 什么是项目

软件项目：实现具有一定完整性实用功能的软件

需求分析🡪概要设计🡪项目规划🡪详细设计🡪修改补充🡪项目测试🡪项目发布🡪后期维护

🡪代码实现🡪代码自测

需求分析：确定用户的真实需求

1. 对用户需求反复确认，形成需求文档
2. 对项目的实施难度和可行性有基本的判断
3. 文档需要使用通俗准确的表达，让客户确认

概要设计：针对项目进行分析，确定整体技术方向

1. 确定功能模块
2. 确定使用的技术
3. 确定整体的逻辑流程
4. 确定自定义的协议
5. 完成流程图，概要设计文档等开发指导文件

项目规划：确定项目开发的时间轴和流程

1. 确定开发工作的先后顺序和并行事件
2. 确定人员分配，时间轴节点，事件里程碑
3. 通过甘特图，思维导图等指导工作

详细设计：对项目进行具体设计

1. 对项目具体实现进行设计
2. 完成详细设计文档：思路、逻辑说明，功能说明，重要变量，技术的，数据结构，代码模块说明等

项目编码：实现项目代码

1. 写代码
2. 技术攻关
3. 代码整合自测

项目测试：按照项目要求进行测试

1. 测试工程师完成测试工作
2. 根据测试结果进行bug修改

项目发布：项目部署发布

1. 项目提交用户，部署发布
2. 编写使用说明文档
3. 项目注意事项
4. 良好的项目规划是按时完成项目的前提
5. 良好的文档是项目的保障
6. 及时避免项目中出现的冲突
7. 项目中的文档说明
8. 项目工具推荐
9. 编写文档：word、ppt、Excel、markdowm、LaTex
10. 项目流程图：xMind（思维导图）、Mindmanager（思维导图）、visio（逻辑流程图）
11. 项目管理：project（）
12. 代码管理：svn、git

### \*电子词典

功能说明

1. 用户能够注册和登录，登录需用用户名和密码进行登录，注册要求用户名和密码，用户名不能和已有的重复
2. 用户信息能够长期保存，要求使用数据库进行保存，
3. 要求能够满足多个客户端同时操作词典
4. 用户运行客户端即进入一级界面（登录、注册、退出）
5. 用户登陆成功进入二级界面（查词、查看历史记录、退出）
6. 用户从二级界面退出，会回到一级界面
7. 查词：可以循环查询英语单词 ##退出查词状态

文本操作、将单词本中的单词存入数据库

1. 查看历史记录：可以查看当前用户的历史查词情况

username word times

1. 单词本特征

\*每个词一行

\*单词按升序排列

\*单词与解释之间有空格

Tips

结构：c/s结构

服务器：数据库交互，注册，登录，查词，历史记录，并发（多进程）

客户端：打印命令选择界面发出请求，接收服务端信息，展示内容

技术点：并发、fork、数据库操作、建立表、网络传输（tcp）

需要模块：os、socket、pymysql

1. 确定数据表结构
2. 考虑如何处理单词本
3. 确定项目的整体结构

服务器：

main（）：创建套接字🡪父子进程🡪父进程等待新的客户端连接

child（）：子进程能够接收请求，根据请求调用相应的功能函数

login（）：和数据库比对是否允许登录

register（）：判断是否和其他用户重名

query（）：查词，给客户端，存入到历史记录

history（）：查询数据库，将结果给客户端

客户端：

main（）：创建套接字，发起连接🡪一级界面

login（）：发登录信息🡪进入二级界面

regster（）：注册

query（）：发送单词，接收反馈，打印

history（）：发送请求，接收反馈，打印

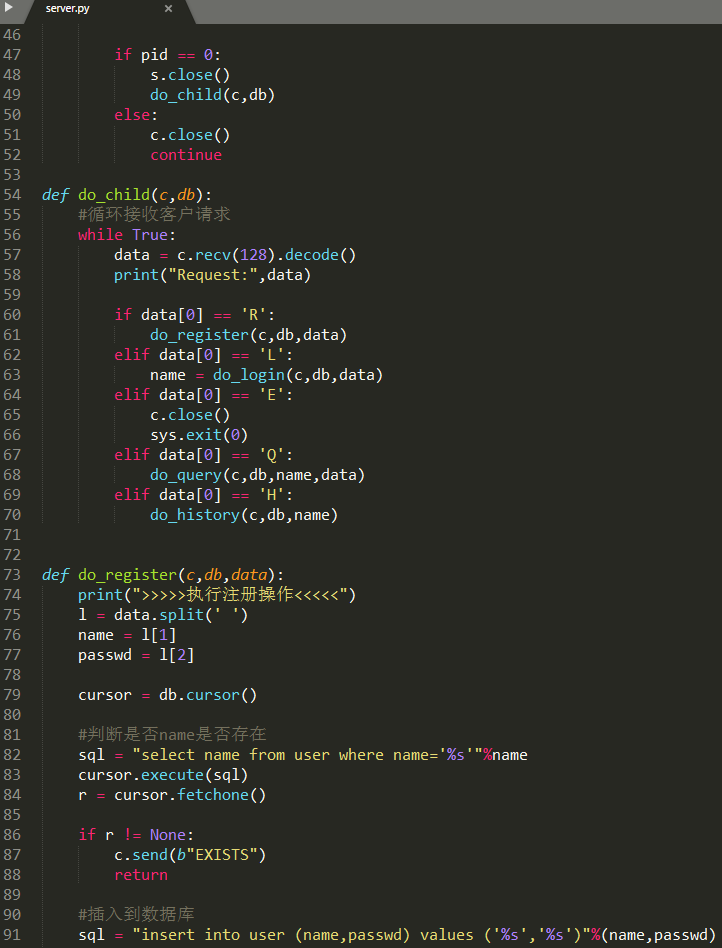
### extra

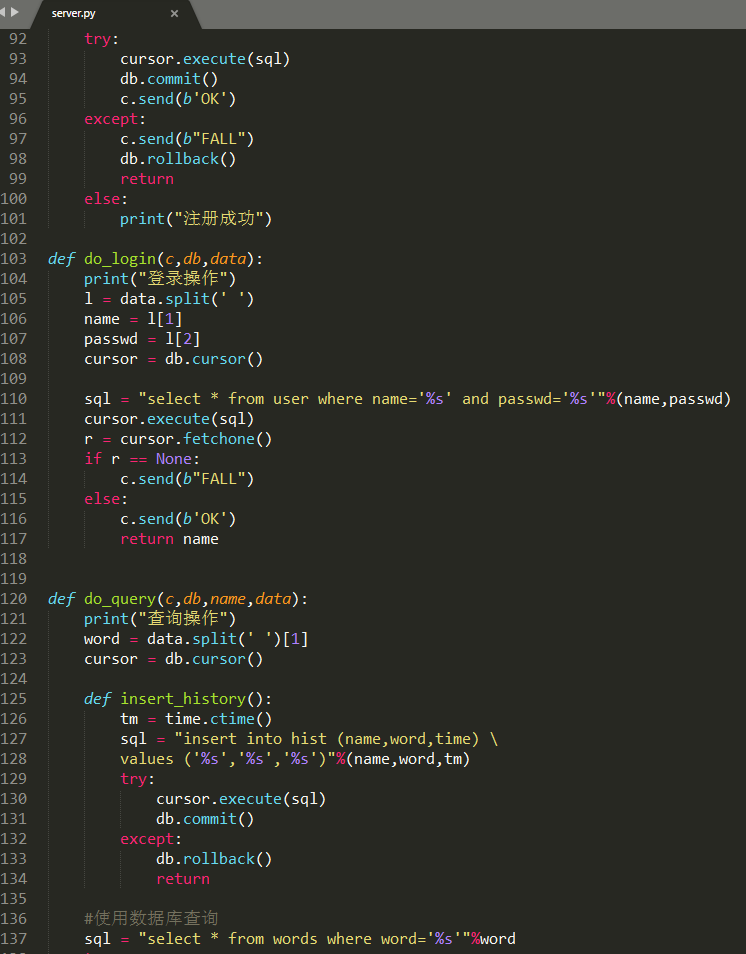
1.在except中导入traceback模块使用traceback.print\_exc() 打印更详细的异常信息

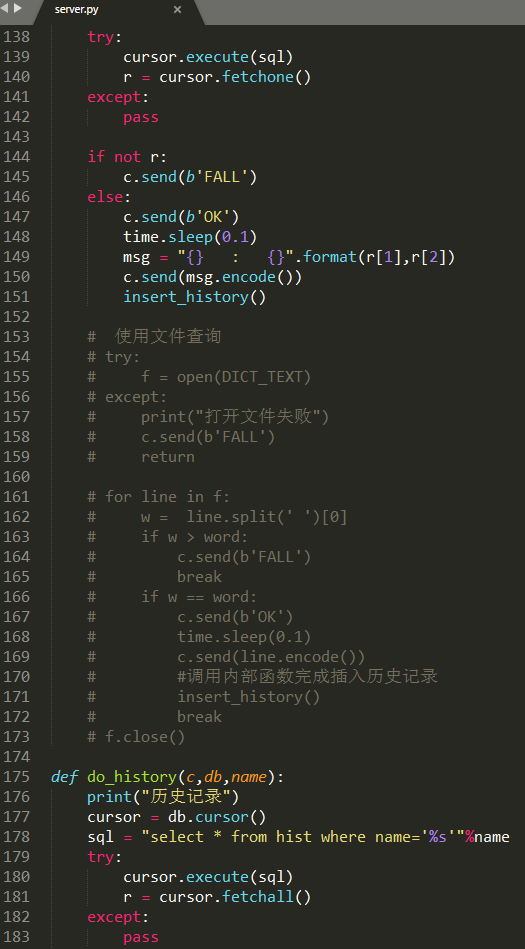
2.在输入密码时可以使用getpass模块，p = getpass.getpass()，可使密码不可见

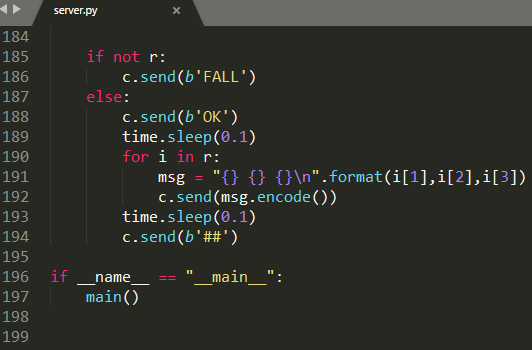
#### 服务端



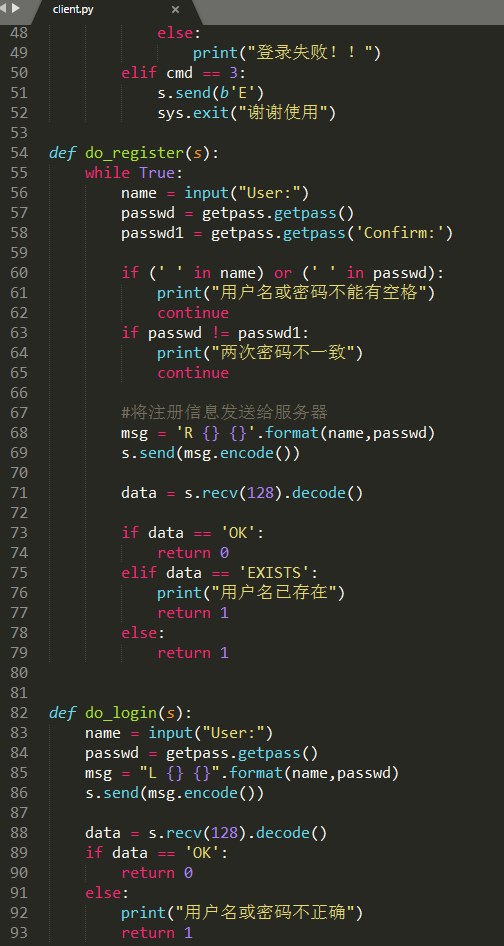
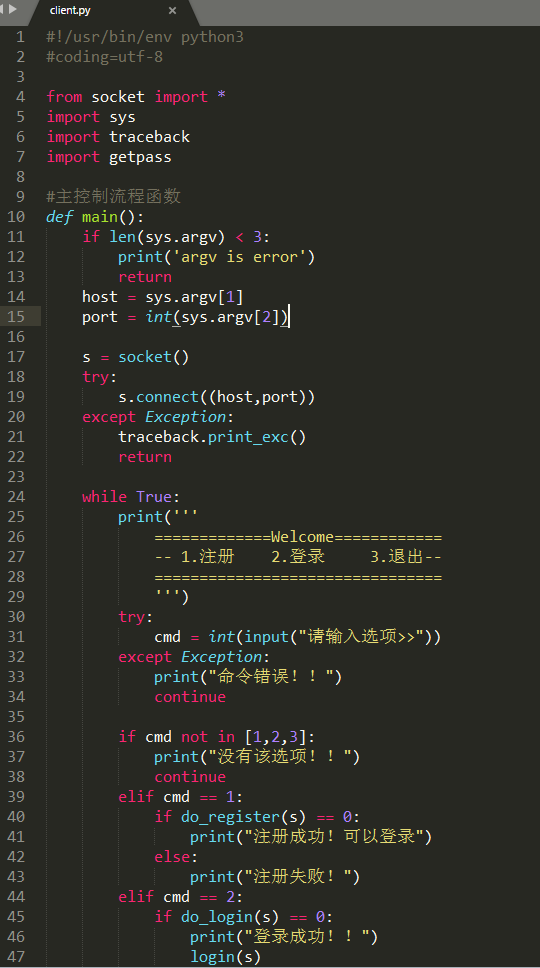


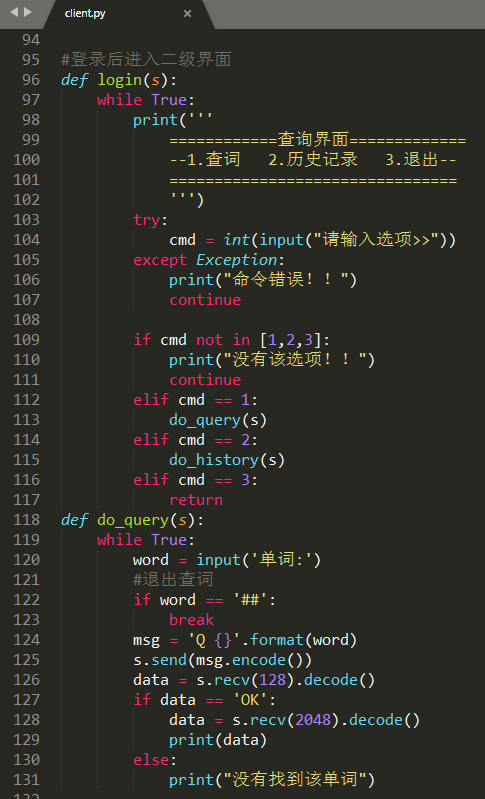






#### 客户端





### pip工具的使用

pip：python的标准第三方库管理工具

pip安装：

python2：sudo apt-get install python-pip

python3：sudo apt-get install python3-pip

安装python包：

sudo pip3 install Package（第三方包名）

e.g. sudo pip3 install image

升级：pip3 install --upgrade Package（软件包名）

卸载；pip3 uninstall Package

查看软件包清单：pip3 list

查看软件包信息：pip3 show Package

查找软件包：pip3 search Package

导出python3软件环境

pip3 freeze > requirements.txt

根据上述文档进行python软件环境安装

pip3 install -r requirement.txt

### pdb调试

pdb模块🡪提供调试手段，去调试代码

功能：设置断点 单步执行 进入函数 查看变量值…

调试命令：break/b：设置断点

continue/c：继续执行

list/l：查看当前代码段

step/s：进入函数

exit/q：退出

next/n：执行下一行

pp：打印变量

help；帮助

import pdb

pdb.set\_trace()

功能：设置初始断点，开始pdb调试

直接以pdb调试模式运行程序：python3 –m pdb 程序名

### \*httpserver带框架版本

前端 前台 客户端 用户端

功能：

1. 和用户进行交互获取用户请求
2. 和后端进行通信，整理用户请求（一般按照某种协议）给后端
3. 接收后端发送的内容，以一定形式呈现给用户

要求：

1. 良好的用户体验
2. 全面方便的交互能力
3. 良好的跨平台性
4. 良好的扩展性及优化

后端 后台 服务端

功能：

1. 接收前端的请求
2. 进行逻辑处理和运行
3. 和磁盘进行交互（文件、数据库）
4. 对数据进行整理，向前端发送结果

要求：

1. 更高的并发性
2. 更快的处理速度（良好的算法）
3. 更强的健壮性
4. 良好的可维护性（高内聚低耦合）

网站后端

httpserver：接收http请求（tcp socket）

解析http请求（GET 请求具体内容）

将具体请求交给后端服务程序处理

接收服务器返回的结果

将返回结果发送给客户端 response

后端服务程序：接收具体的请求内容

如果是静态网页，调用文件处理，是应用则调用函数处理

将结果发送给httpserver

HTTPServer：-static 存放静态网页

-index.html

…

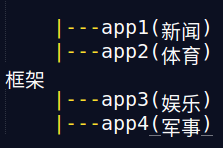
-wsgiPy 存放应用模块

-HTTPServer.py httpserver程序

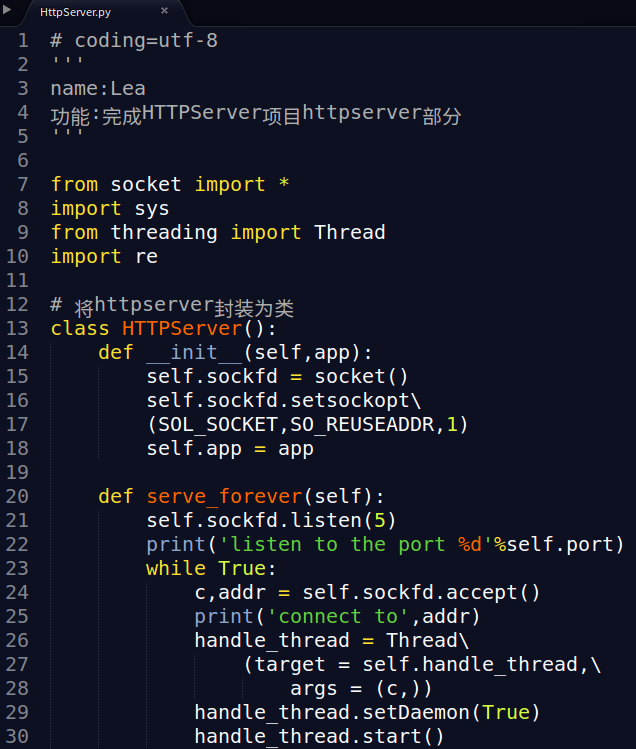
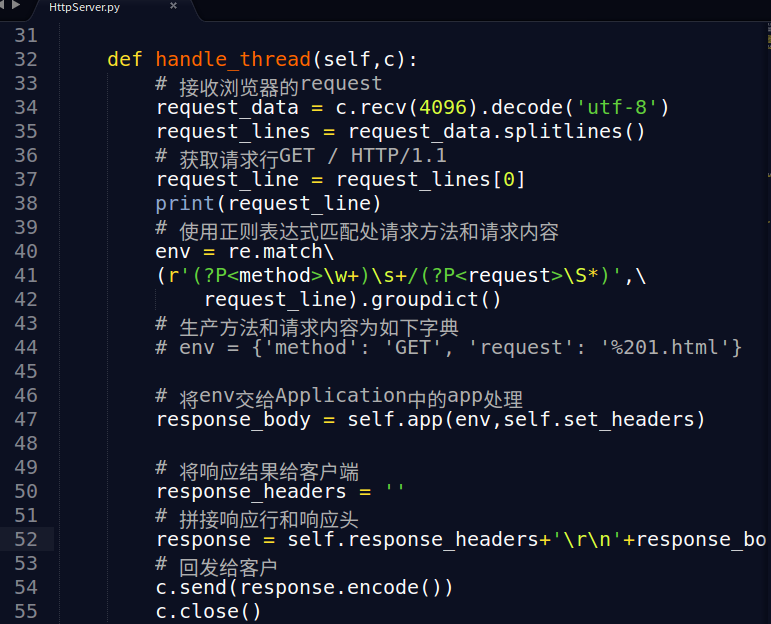
-WebFrameWork.py 应用服务程序：框架

\*\*HTTPServer与后台的框架往往使用进程间通信的方式传输信息（套接字）

服务端框架结构



### httpserver端

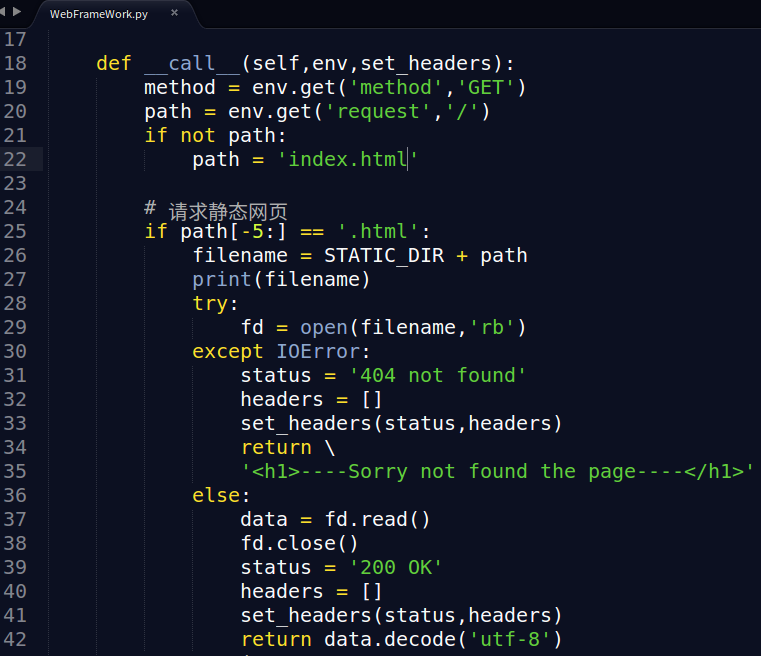
 





### 后端服务程序







### extra

1.什么是框架

框架即开发模板，通过填写不同的内容可以快速地完成开发

2.如何随时导入模块

1. 将模块所在的文件夹目录加入到python环境变量

sys.path：是一个列表，表示python环境变量，只需将目录添加到列表中即可

1. 使用\_\_import\_\_()导入

3.\_\_call\_\_（）方法的功能

在类中\_\_call\_\_()方法可以让该类的实例对象通过（）像函数一样使用，这时\_\_call\_\_方法就会被自动调用，同时也可以传参，有返回值



## httpserver模块

python2 BaseHTTPServer

python3 http.server

