—————————————————————————

Python 阶段二

文件 - 设置 - Python解释器

1.系统性能定时监控模块安装命令： pip3 install psutil

2.发送邮件安装命令： pip3 install yagmail

3.虚拟环境安装命令：

1.pip install virtualenv - 虚拟环境

2.pip install virtualenvwrapper - 更加友好的命令

4.对于元组的转义为字符串 str(元组)

5.用于获取当前以占用的端口的命令：netstat

6.协程 greenlet :pip3 install greenlet

7.协程 gevent :pip3 install gevent

8.深浅拷贝：import copy

9.接受系统的相关东西：import sys

10.模块的重加载：import reload

—————————————————————————

>>系统性能定时监控 - 介绍（第一课）

psutil 是 python 中的系统监控及进程的管理工具

————————————————————————》获取CPU信息（1）

格式： psutil.cpu\_+++( )

---------------------------------------------

1.cpu\_count()

逻辑cpu核心数

可以通过 logical 来返回物理 cpu 的个数

- 什么叫物理？

物理就是实实在在的，假设一核cpu分化8核虚拟cpu

传参传 False 就是物理

想要虚拟可以不传，默认True

psustil.cpu\_count(logical = False)

---------------------------------------------

3.cpu\_percent() %

获取cpu使用率

- 为什么会返回0.0？

默认情况下是0.0，要获取cpu使用率得设置它的频率

每隔多长时间获取一次

应为没做时间的设置，所以默认0.0

**- interval = 设置间隔，不写即为默认**

**- percpu = 返回每一个cpu使用率情况，默认为False**

设置刷新间隔为1秒

print(psutil.cpu\_percent(interval = 1,percpu= True))

返回：[9.0, 1.5]

————————————————————————》获取内存信息（2）

格式：+++\_memory( )

---------------------------------------------

1.virtual\_memory()

获取内存整体信息

print(psutil.virtual\_memory())

**- total = 物理内存总数**

**- available = 可用内存，表示没有进入交换区的内存，可以直接分配给进程**

**- percent = 百分比**

**- used = 已使用的内存数**

**- free = 空闲的，指完全没有被使用**

**- active = 活跃的**

**- cache = 缓存的使用数目**

**- buffer = 缓冲的使用数目**

**- swap = 交换分区使用的数目**

如果要获取某个具体的信息：

print(psutil.virtual\_memory().free)

————————————————————————》获取硬盘信息（3）

我们比较关心的是磁盘的利用率以及I/O信息还有分区信息等

格式：disk\_+++( )

---------------------------------------------

1.disk\_partitions()

获取所有硬盘信息

print(psutil.disk\_partitions())

**- device = 分区**

**- mountpoint = 挂载到**

**- fstype = 格式**

---------------------------------------------

2.disk\_usage()

获取指定的目录硬盘信息

print(psutil.disk\_usage("C:/"))

**- 获取C盘信息**

**- total = 总计有多少字节**

**- used = 以使用字节**

**- free = 空闲多少字节**

**- percent = 使用率**

获取指定数值

print(psutil.disk\_usage("C:/").percent)

————————————————————————》获取网络信息（4）

我们比较关心的是磁盘的利用率以及I/O信息还有分区信息等

格式：net\_+++( )

---------------------------------------------

1.net\_io\_counters()

获取收到的数据包数量

**- bytes\_sent = 发送的数据包/字节**

**- bytes\_recv = 收到的数据包/字节**

—————————————————————————

### >>获取开机时间（扩展）

print(psutil.boot\_time())

获取1970年1月1日0点到现在的秒数

—————————————————————————

### >>获取当前时间（扩展）

datetime.datetime.now().strftime("%F %T")

—————————————————————————

### >>获取活动用户（扩展）

print(psutil.users( ))

获取活跃用户

—————————————————————————

>>yagmail模块发送邮件命令（第二课）

- yagmail模块可以发送简单命令来实现发邮件功能

例子：

yag=yagmail.SMTP(user="user@126.com",password="1234",host='smtp.123.com')

contents = ["这是一个测试"]

yag.send("taaa@126.com", "subject", contents)

**- user@123.com = 发件人邮箱**

**- 1234 = 发件人邮箱授权码，不是验证码**

**- smtp.126.com = 网易126邮箱发件服务器**

**- send() = 发送邮件**

**- taaa#126.com = 收件人邮箱**

**- subject = 邮件主题**

**- 注意，发送邮件必须先在电脑上登陆到邮箱**

个人注释：

首先导入模块：import yagmial

定义变量：ya\_obj 去接受返回值

变量接受 = 导入模块.STMP(user="发送人邮箱",password="授权码",服务器="stmp.163.com")

定义变量去接受要发送的内容 = centent = "文本"

上面定义的变量.send("收件人邮箱","标题文本",接受内容的变量)

—————————————————————————

>>虚拟环境以及用途（第三课）

- 为什么虚拟环境？

**假设一个Py3代码用的框架比Py2框架新，那么Py2代码会运行不了**

**所以要单独开设两个环境去分别存储Py2框架与Py3、框架**

**互不影响**

virtualenv用于创建独立的Py环境，多个Py相互独立，互不影响它可以：

- 在没有权限的情况下安装新插件

- 不同的应用可以使用不同的套件版本

- 套件的升级不影响其他应用

—————————————————————————

>>虚拟环境搭建（第四课）

- 如果不指定Py版本，则默认安装Py2的虚拟环境

- 创建默认Py版本的虚拟环境

mkvirtualenv 虚拟环境名称

例：

mkvirtualenv py\_flask

注意：

1.创建虚拟环境需要联网

2.创建成功后，会自动工作在这个虚拟环境上

3.工作在虚拟环境上，提示符最前面会出现“虚拟环境名称”

虚拟环境的默认位置：

/home/demo/.Envs/虚拟环境名

咱这新建项目就行了

用不了命令

—————————————————————————

>>网络通信概述（第五课）

- 网络的概念

**一些以共享资源为目的的计算机的集合，**

**网络编程：通过编写程序实现计算机之间的数据传递**

—————————————————————————

>>ip地址（第六课）

**- ip地址是用于标识网络上唯一的一台主机**

概念：

**1.IP地址是指互联网协议地址，是IP Address的缩写，IP地址是IP协议提供的一种统一的**

地址格式

2.IP地址用来给 Internet 上电脑的一个编号，IP地址就相当于电话号码

- IPv4与IPv6的区别

**xxx.xxx.xxx.xxx ——》IPv4 点分十进制**

**xxx:xxx:xxx...... ——》IPv6 冒号分十六进制**

—————————————————————————

>>ip地址查看（第六课）

- ifconfig ——》ipconfig

用于查看IP地址

- ping

用于查看连接

—————————————————————————

>>端口（第七课）

- 知道端口分为知名端口，动态端口两类

- 能够说出80，21，22知名端口的作用

端口的作用：

- 能让计算机知道，某个软件消息发给那个特定的软件

**- 可以理解为设备与外界通讯的出口，端口分为虚拟端口与物理端口**

**其中虚拟端口指计算机内部或交换机路由器内的端口，不可见**

例如计算机中的80端口，21端口，23端口

知名端口：0 ~ 1023（不可动）

动态端口：1023 ~ 65535

知名端口： 默认端口号 协议基本使用

FTP 21 文件上传/下载

SSH 22 安全的远程登陆

TELNET 23 远程登陆

SMTP 25 邮件传输

DNS 53 域名解析

HTTP 80 超文本传输

POP3 110 邮件接收

HTTPS 443 加密传输的HTTPS

—————————————————————————

>>网络传输方式（第八课）

**- 传输方式：数据发送的方法主要是 UDP , TCP**

- 网络通信中根据数据发送方法进行多种分类，分类方法主要分两种

1.面向有连接型(TCP)

**- 面向有连接在发送数据之前，需要在收发主机之间建立一条连接通信线路**

**- 面向连接就像我们平时打电话，必须在通信传输前后，专门进行建立和断开**

- 连接的处理

**- 只有在确认通信对端存在才会收发数据，可以控制通信流量的浪费**

**- TCP提供了数据传输时的各种控制功能，丢包时也可以进行**

**- 丢包重发控制，还可以将次序乱掉的部分进行顺序控制**

2.面向无连接型（ UDP ）

**- 面向无连接型不要求建立或断开连接，发送端可由任何时候自由发送数据**

- 反之，**接收端也永远不知道自己在何时从那里接受到数据**，

- 因此，面向无连接情况下，接收端需要时常确认是否接收到数据

**- 在面向无连接通信中，不需要确认对端是否存在，**

**- 即使接收端不存在或无法接受数据，发送端也能将数据发送出去**

- - - - - - - - - - - - - - - - - - -

**- 这玩意只负责发，丢了也不管**

—————————————————————————

>>socket简介（第九课）

- socket就像快递员一样

- UDP ，TCP可以理解为传输方式

- 端口理解：相当于门牌号一样，知道哪个软件的信息发到哪个软件的位置

- IP地址：通过端口发送消息，包括自己IP地址与对方IP地址

1.这节课主要讲Socket

**- 是网络通信的控制单元**

- 它里面拥有函数以及方法，能哦让我们知道网络的收发

TCP = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

UDP = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

TCP/UDP.close()

关闭协议

—————————————————————————

>>udp网络-发送数据（第九课）

1.socket.socket -- 建立套接字

- socket.AF\_IENT 表示 IPv4 地址

- socket.SOCK\_DGRAM 表示使用 UDP 传输数据

2.socket.sendto -- 发送数据

**- 要发送的二进制数据**

- 元组类型，指定把（参数1的数据）发给谁

（ip 地址字符串，端口数值）

udp\_socket.sendto("你好".encode(),("对方IP地址",端口号))

**- encode : 将前面的字符串转为二进制**

例子：

import socket

udp\_soclet = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

udp\_soclet.sendto("你好".encode(), ("123.0.0.1", 8080))

udp\_soclet.close()

—————————————————————————

>>udp网络-发送并接受数据（第十课）

**1.使用 socket 的 recvfrom 方法接受数据**

**2.使用 decode 方法能够对接收到的数据解码**

- decode() 接受数据时的组合 “GBK”解码方式

- encode() 发送数据时的拆解

- socket.AF\_IENT 表示 IPv4 地址

- socket.SOCK\_DGRAM 表示使用 UDP 传输数据

- socket.SOCK\_STREAM 表示使用 TCP 传输数据

两个一开一合，充当翻译工作

**- recvfrom ——》最大的接受缓存，字节为单位**

- recv\_data[0]:用户发来的二进制文本，用decode("GBK")翻译

- recv\_data[1]:用户的IP地址以及端口号

例子：

import socket

udp\_soclet = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

udp\_soclet.sendto("你好".encode(), ("123.0.0.1", 8080))

recv\_data = udp\_soclet.recvfrom(1024) ——》**返回一个列表，中有两个元组**

recv\_text = recv\_data[0].decode()

print("来自",recv\_data[1],"用户发的",recv\_text)

udp\_soclet.close()

可以使用手机虚拟机进行通信

—————————————————————————

>>python3编码转换（第十一课）

- 知道使用encode()与decode()对字符串进行编码与解码

**- 文本总是 Unicode str进行表示**

**- 二进制使用 bytes 进行表示**

**- 网络中的数据是以二进制编码进行传输的，所以要对文本进行编码与解码**

—————————————————————————

>>GBK 与 UTF - 8（扩展）

**- GBK通常在国内使用**

**- UTF - 8 被成为万国码，假设网站登陆外网，尽量转成万国码**

**- decode( ) 默认使用UTF - 8**

**- 什么封，什么解**

指定解码格式：

- 指定解码格式：UTF - 8/GBK

**- errors = "" = 出现错误后如何处理**

**- "ignore" = 忽略**

**- 默认严格模式**

encode(encoding="UTF-8", errors="ignore")

encode("UTF-8", "ignore")

—————————————————————————

>>udp端口绑定 - 发送端（第十二课）

1.问题：会变的端口号

- 如何绑定？

目标：使用 bing 方法在发送端给套接字绑定端口

- 发送端端口绑定

2.核心方法

**- socket.bing(元组(IP地址，端口号)) -- 将socket对象绑定到一个地址，但这个地址**

**- 不能被占用，否则会连接失败**

例子：

import socket

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

addr = ("192.168.31.122", 7890)

ip地址可以省略，省略后表示自己的IP地址

udp\_socket.bind(addr)

就是绑定自己的端口，让自己的端口不会变动

udp\_socket.sendto("你好".encode(),("192.168.31.167", 8080))

udp\_socket.close()

addr = ("", 7890) 《——方式

—————————————————————————

>>udp端口绑定 - 接收端（第十三课）

**a,b = (10,20) a = 10,b = 20**

- 使用 bind 方法在接收端套接字绑定端口

例子：

import socket

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

udp\_socket.bind(("127.0.0.1", 6666))

recv\_data, ip\_port = udp\_socket.recvfrom(1024)

print(recv\_data, ip\_port)

udp\_socket.close()

好像没啥本质上的区别

和上面的去比较好像就少了个发送

—————————————————————————

### >>内容扩展

**- NAT模式：虚拟机跟物理主机共用网络**

**- 桥接模式：虚拟机能够获取互联网的IP地址**

—————————————————————————

>>UDP广播（第十四课）

1.知道广播的概念

向 255.255.255.255 发送消息就是广播地址

向 192.168.1.255 就是向 192.168.1.0 ~192.168.1.255 发送广播消息

2.能够使用UDP群发广播消息

- 套接字默认不允许发送广播，需要开启相关权限

udp\_socket.setsockopt(套接字，属性，属性值)

1.socket.SOL\_SOCKET = 当前的套接字

2.socket.SO\_BROADCAST = 广播属性

3.默认为False,需要设置为True

例子：

udp\_socket.setsockopt(socket.SOL\_SOCKET,socket.SO\_BROADCAST, True)

—————————————————————————

>>UDP聊天器（第十五课）

- 假设给自己发送消息时，socket会有一个缓冲区，

**- 发一个会缓冲一个**

**- 接受一个会拿取一个**

- 可以len( )去判断默认

—————————————————————————

>>TCP简介（第十六课）

**1.tcp连接数据，传送，终止的三个步骤**

**2.tcp可靠传输的原因**

**3.tcp 与 udp 的区别**

- tcp通信要经过 创建连接，数据传送，终止连接 三个步骤

**4.面向连接**

- 双方必须为该连接分配必要的系统内核资源，以管理连接状态与内容上的传输

- 因此 TCP 不适用于广播程序，基于广播的使用 UDP

- 这种连接是一对一的

**5.可靠传输**

- TCP 发送的每个报文段必须得到接收方的应答才认为成功

**6.超时重传**

- 发送端发送一个报告段之后就启动一个定时器，如果在时间内没收到应答

- 就重发这个报文段

- TCP 为了保证不丢包，就给每一个包一个序号，同时序号也保证了传送到接受端

- 实体的包的按序接收，然后接收端实体对已成功收到的包发回一个

- 相应的确认，如果发送端实体在合理的往返时限内未收到确认

- 那么对应的数据包就被假设为已丢失将会被进行重传

**7.错误校验，去除重复**

- TCP 用一个校验和函数来检验数据是否有错误，在发送和接受时都要计算校验和

**8.流量控制和阻塞管理**

- 流量控制用来避免主机发送的过快而接收方来不及完全收下

—————————————————————————

>>TCP 与 UDP 不同点（第十七课）

**- 是面向连接的(确认有创建三方交握，连接以创建才作传输)**

**- 有序数据传输**

**- 重发丢失的数据包**

**- 舍弃重复的数据包**

**- 无差错的数据传输**

**- 阻塞/流量控制**

**1.TCP严格分客户端，服务器**

客户端：

socket对象 ——》connect——》send( recv )——》clese

服务器端

socket对象——》bing——》listen——》accept——》recv( send )——》close

—————————————————————————

>>tcp客户端（第十八课）

**- 服务器：TCP Server**

**- 客户端：TCP Client**

- connect = 连接到

**- 核心方法:**

**- 如果连接出错，返回socket.error错误**

1.socket.connect(address)

addr = ("192.168.4.206", 8080)

socket.connect(addr)

或

socket.connect(("192.168.4.206",8080))

2.socket.send("你好".encode())

3.socket.recv(1024)

例子：客户端向服务器发送消息

import socket

tcp\_client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_client\_socket.connect(("127.0.0.1", 4567))

tcp\_client\_socket.send("Just Monika".encode("GBK"))

recv\_data = tcp\_client\_socket.recv(1024)

print(recv\_data.decode("GBK"))

tcp\_client\_socket.close()

—————————————————————————

>>tcp服务端（第十九课）

- listen(第一个数字)：**设置socket为监听状态，它将不会再发送消息**

- **第一个数字：最大的连接数**，在windows系统下才有效

- accept()：**接受客户端连接，程序默认进入阻塞状态，等待客户端**

**- 连接成功时会有一个返回值(两个部分)**

- 1.返回了一个新的套接字 \_ 对象

- 2.客户端的IP地址和端口号

1.返回了一个新的套接字 \_ 对象的解释

**- 前面设置了listen()，那么上面的套接字将永远变成监听的，无法再发送**

**- 消息**

**- 所以它会再次创建一个套接字，用于与客户端往来消息**

**- 假设又连接了一位客户，那么他会再次创建一个套接字，互不干扰**

new\_client\_socket.close() - 关闭和当前客户端的连接

tcp\_server\_socket.close() - 服务器不在接受新的客户端，老客户端可以继续服务

- 当前问题：只能接受一条消息

—————————————————————————

>>tcp服务端增强（第二十课）

- 能够接受客户端发来的多条消息

- 能够接受多个客户的连接

import socket

tcp\_server\_socket= socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

address = ("", 8080)

tcp\_server\_socket.bind(address)

tcp\_server\_socket.listen(128)

#第一次阻塞

while True:

new\_client\_socket, client\_ip\_port = tcp\_server\_socket.accept()

#第二次阻塞

while True:

recv\_data = new\_client\_socket.recv(1024)

if recv\_data:

recv\_text = recv\_data.decode()

print("接收到", client\_ip\_port, "的消息:", recv\_text)

else:

print("客户端断开连接")

break

new\_client\_socket.close()

tcp\_server\_socket.close()

—————————————————————————

>>文件下载器（第二十一课）

**with open("文件名"，打开方式) as file**

**其实等于**

**file = open(文件名，打卡方式)**

其实道理一样，但是with open方法可以自动帮我们关闭

客户端

import socket

tcp\_client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_client\_socket.connect(("127.0.0.1", 8080))

file\_name = input("请输入要下载的文件名：\n")

tcp\_client\_socket.send(file\_name.encode())

with open("E:/cdvbcd/%s"% (file\_name), "wb") as file:

while True:

file\_data = tcp\_client\_socket.recv(1024)

if file\_data:

file.write(file\_data)

else:

break

tcp\_client\_socket.close()

服务端

import socket

tcp\_server\_socket= socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(100)

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

print("欢迎", ip\_port, "连接")

recv\_data = new\_client\_socket.recv(1024)

file\_name = recv\_data.decode()

print(file\_name)

with open(file\_name, "rb") as file:

while True:

file\_data = file.read(1024)

if file\_data:

new\_client\_socket.send(file\_data)

else:

break

—————————————————————————

>>文件下载器 - 1（第二十二课）

- 实现根据文件名，实现下载内容

- 文件下载完成，服务器端基于提示

- 下载的文件不存在，则提示错误

客户端

import socket

tcp\_client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_client\_socket.connect(("127.0.0.1", 8080))

file\_name = input("请输入要下载的文件名：\n")

tcp\_client\_socket.send(file\_name.encode())

with open("E:/cdvbcd/%s"% (file\_name), "wb") as file:

while True:

file\_data = tcp\_client\_socket.recv(1024)

if file\_data:

file.write(file\_data)

else:

break

tcp\_client\_socket.close()

服务端

import socket

tcp\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(100)

while True:

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

print("欢迎", ip\_port, "连接")

recv\_data = new\_client\_socket.recv(1024)

file\_name = recv\_data.decode()

print(file\_name)

try:

with open(file\_name, "rb") as file:

while True:

file\_data = file.read(1024)

if file\_data:

new\_client\_socket.send(file\_data)

else:

break

except FileNotFoundError as e:

print("文件下载失败")

else:

print("文件下载成功")

new\_client\_socket.close()

#tcp\_server\_socket.close()

—————————————————————————

>>tcp的三次握手（第二十三课）

- 知道 tcp 三次握手是用于连接

**- 所谓三次握手即建立TCP连接，就是指建立一个TCP连接时**

**- 需要客户端和服务端总共发送三个包以确认连接的建立**

- 在socket编程中，这一过程由客户端执行connect来触发

**- 三次握手是TCP建立连接的时候发送的三个数据包**

- 1.概念

- 2.简单描述通信过程

**- 3.必须是三次，如果少于三次，会造成系统资源的浪费**

—————————————————————————

>>tcp的四次挥手（第二十四课）

**- tcp的四次挥手，是用于断开连接**

**- tcp的四次挥手，主要说TCP断开连接的时候发送的四个数据包确保断开**

**设置套接字可以重用**

tcp\_server\_socket.setsockopt(当前套接字，属性名，属性值)

tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, True)

- socket.SO\_REUSEADDR = 地址是否可以重用

- True = 可以重用

TCP连接必须经过时间2MSL的时间后才断掉

**- MSL = 最大报文生存时间**

—————————————————————————

>>IP地址和域名（第二十五课）

- 域名的概念

**域名，简称DN,域名可以理解为一个网址，就是一个特殊的名字**

- 为什么要有域名？

互联网上的每一台主机，都有一个唯一的IP地址，**但是IP地址不方便记忆**

**- 所以才有了域名**

域名的构成：由字母，数字，中划线-，长度不超过255个字符

例如：www.baidu.com其中，**.com为顶级域名**

**常见的域名：**

**.com 用于商业机构，他是最常见的顶级域名，任何人都可以注册.com形式的域名**

**.cn 中国专用的顶级域名**

**.gov 国内域名，政府，企业单位常见域名**

**.org 是为各种组织包括非盈利组织而定的，任何人都可以注册.ORG结尾的域名**

**.net 最初是用于网络组织，例如英特网维修商和服务商，任何人都可以注册以.NET结尾的域名**

**.com.cn 国内常见二级域名**

**localhost 是个域名，不是地址，它可以被分配到任意的IP地址**

**- 去阿里云购买域名**

—————————————————————————

>>DNS及浏览器请求服务器过程（第二十六课）

**- 知道浏览器访问网站的大致流程**

1.DNS服务器

**- DNS是一台运行在互联网上的服务器**

**- 主要用于将域名转为对应的Ip地址**

**- DNS服务器可以看作是一个通讯录，(姓名——》域名——》电话——》ip地址)**

浏览器输入网址，**会先去本地DNS服务器去查询IP地址，没有的话会去远程DNS服务器**

- 查询IP地址，如果还没有就访问不到网页

**- 本地DNS服务器就是个文件**

C:\Windows\System32\drivers\etc -- windows

—————————————————————————

>>HTTP协议简述（第二十七课）

**- HTTP协议就是基于tcp连接的协议**

- http是应用层协议

- tcp是传输层协议

- IP是网络层协议

- 网络是链路层协议

- 知道HTTP协议是为了实现浏览器与服务器之间的通信

- 能够了解HTTP协议的组成和应用

**HTTP = 超文本传输协议 = 应用层协议**

**http协议分为两部分：请求协议与响应协议**

**协议由协议项来构成**

**协议名与协议值构成协议项**

注意：每一个协议都要**单独站一行**

—————————————————————————

>>HTTP协议格式查看（第二十八课）

谷歌

- 右键，检查，network，刷新

- 更多，开发者工具，

- F12

火狐

- 右键，检查元素，网络，刷新

—————————————————————————

>>HTTP请求报文格式（第二十九课）

**- 知道请求报文包含请求行，空行，请求体**

1.第一行是请求行

2.第二行到最后一行为请求头

3.请求空行

4.请求主体

**GET方式只有请求行，请求头，空行**

**POST方式有请求行，请求头，空行，主体**

**1.请求行分为三部分：请求方式、请求路径、请求协议及协议版本**

例子：GET /a/b/c/d HTTP/1.1

请求方式 路径,/便是跟目录 请求协议/版本

- 请求行需要单独占一行

- 请求路径不包括域名

- HTTP协议以前是1.0版本，现在是1.1版本

**2.请求头**

**请求头就是所有当前需要用到的协议项的集合**

协议项就是浏览器在请求服务器时事先告诉服务器的一些信息，一些事先的约定

常见的请求头有：

1.host:当前uri中所要请求的服务器的主机名（域名）

- Host:www.bbs.com

2.accept-encoding:是浏览器发给服务器，声明浏览器支持的压缩编码类型，比如gzip

- Accept-Encoding:gzip, deflate

解释：在我们浏览器请求服务器时，服务器会将数据进行一个打包，上面的例子表示

- 当前浏览器支持gzip与deflate的解包格式，让服务器以这个格式进行压缩

3.accept\_charest:表示浏览器支持的字符集

4.referer:表示，此次请求来自哪个网址

5.accept-language:可以接受的数据类型，cn,en有权重高低之分

6.权重的解释：差不多等于优先级

7.cookie:如果之前当前请求的服务器在浏览器端设置了数据(cookie)，那么当前浏览器

- 再次请求该服务器的时候，就会把对应的数据带过去

解释：假设当你浏览完以后，它会自动在你电脑上保存文件，保存了你之前感兴趣的信息

- 在你下一次打开网页时，他会自动的将这个文件发给服务器，然后服务器

- 就知道了（加密传输）

8.user-agent:用户代理，当前发出请求的浏览器的内核信息

- 在后期写爬虫时，可能需要将爬虫伪装成一个服务器

9.accept:表示浏览器可以接受的数据类型：text/html,image/img

10.content-length(post):只有post提交的时候才会有的请求头，显示的是当前要提交的

- 数据长度(字节)

11.if-modified-since(get):表示在客户端向服务端请求某个资源文件时，询问此资源是否

- 被修改过

3.空行

- 就是用来分离请求头和请求数据，意思是请求头到此结束

4.请求数据

- 只有post方式提交的时候，才有请求数据

5.分割方法

- 利用\r\n来换行与空行

—————————————————————————

>>HTTP响应报文格式（第三十课）

**- 知道响应报文包含响应行，空行，响应体**

**- 知道响应状态码404表示网站没有找到**

也就是服务器像浏览器做出相应(反馈数据)时候的协议

http响应也分成了四个部分：

1.响应行(状态行)：

- 第一行HTTP/1.1 200 OK 叫做响应行，共分成3部分:协议及版本号，状态码，状态描述

- 200表示浏览器已经接受浏览器的请求，并且返回了数据

- 状态码和状态描述是--对应的

- 状态码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值

- 1xx:指示信息--表示请求已接受，继续处理

- 2xx:成功--表示请求已接受，继续处理

- 3xx:重定向--要完成请求必须进行进一步的操作

- 4xx:客户端错误--请求有语法错误或请求无法实现

- 5xx:服务器端错误--服务器未能实现合法的请求

2.响应头

- 协议项构成

- 协议名/协议值

3.空行

- 分割响应头和响应主体

4.响应主体(响应数据)

- 服务器响应给客户端的数据

2.server:服务器主机信息

3.date:响应时间

4.last-modified:文件最后修改时间

5.content-length:响应主体的长度(字节)

6.content-type:响应内容的数据类型：text/html,image/png等

7.location:重定向，浏览器遇到这个选项，就立马跳转(不会解析后面内容)

8.refresh:重定向(刷新)，浏览器遇到这个选项就会装备跳转，刷新一般有时间限制

- 时间到了才会跳转，浏览器会继续向下解析

9.content-encodeing:文件编码格式

10.cache-control:缓存控制，no-cached不要缓存

—————————————————————————

>>长连接和短链接（第三十一课）

- 在Http1.0中默认使用短连接

- 在Http1.1起默认使用长连接，使用长连接的Http协议，会在响应头加入

- Connection:keep-alive

**- 在真正的读写操作之前，server与client之间必须建立一个连接**

**- 在读写完成之后，双方可以放弃这条连接**

**- 连接经过三次握手，释放需要经过四次挥手**

1.长连接

- 一次连接，多次数据传输，通信结束关闭连接

特点：要么连不上，一旦连接上，速度有保证，

- 当瞬间访问压力比较大的时候，服务器不可用

2.短连接

- 一次连接，一次传输，就关闭

特点：会频繁的建立和断开连接，当瞬间访问压力比较大的时候，服务器响应过慢

—————————————————————————

>>模拟浏览器实现（第三十二课）

import socket

def request\_handler(new\_client\_socket, ip\_port):

print("%s用户已介入" %(str(ip\_port)))

request\_data = new\_client\_socket.recv(4080)

response\_line = "HTTP/1.1 200 ok\r\n"

response\_header = "Server:daomeng/2.1\r\n"

response\_blank = "Hello World"

response\_body = "\r\n"

response\_data=response\_line+response\_header+response\_blank+response\_body

if not request\_data:

new\_client\_socket.close()

return

new\_client\_socket.send(response\_data.encode())

def main():

tcp\_server\_socket=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, True)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(120)

while True:

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

request\_handler(new\_client\_socket, ip\_port)

if \_\_name\_\_ == '\_\_main\_\_':

main()

—————————————————————————

>>返回固定界面（第三十三课）

- 将Hello world 变成了 with open("","") as file:

import socket

def request\_handler(new\_client\_socket, ip\_port):

print("%s用户已介入" %(str(ip\_port)))

request\_data = new\_client\_socket.recv(4080)

response\_line = "HTTP/1.1 200 ok\r\n"

response\_header = "Server:daomeng\r\n"

response\_blank = "\r\n"

#返回固定页面

with open("E:/pythonProject1/grand.html", "rb") as file:

response\_body = file.read()

response\_data = (response\_line+response\_header+response\_blank).encode()+response\_body

if not request\_data:

print("已断开")

new\_client\_socket.close()

return

new\_client\_socket.send(response\_data)

def main():

tcp\_server\_socket=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, True)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(120)

while True:

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

request\_handler(new\_client\_socket, ip\_port)

if \_\_name\_\_ == '\_\_main\_\_':

main()

—————————————————————————

>>返回指定界面（第三十四课）

- 能够实现 根据浏览器不同的请求，返回对应的页面资源 的Web服务器

import socket

def request\_handler(new\_client\_socket, ip\_port):

print("%s用户已介入" %(str(ip\_port)))

request\_data = new\_client\_socket.recv(1024)

if not request\_data:

print("已断开")

new\_client\_socket.close()

return

#根据客户端浏览器请求路径返回信息

request\_text = request\_data.decode()

loc = request\_text.find("\r\n")

request\_line\_list = request\_text[:loc].split(" ")

file\_path = request\_line\_list[1]

#设置默认首页

if file\_path == "/":

file\_path = "/index.html"

response\_line = "HTTP/1.1 200 ok\r\n"

response\_header = "Server:daomeng\r\n"

response\_blank = "\r\n"

#返回固定页面

try:

with open("E:/pythonProject1/game"+file\_path, "rb") as file:

response\_body = file.read()

except Exception as s:

response\_line = "HTTP/1.1 404 Not Found\r\n"

response\_body = "ERROR! (%s)"% str(s)

response\_body = response\_body.encode()

response\_data=(response\_line+response\_header+response\_blank).encode()+response\_body

new\_client\_socket.send(response\_data)

def main():

tcp\_server\_socket=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, True)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(120)

while True:

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

request\_handler(new\_client\_socket, ip\_port)

if \_\_name\_\_ == '\_\_main\_\_':

main()

—————————————————————————

>>面向对象封装（第三十五课）

import socket

class WebServer(object):

def \_\_init\_\_(self):

tcp\_server\_socket=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, True)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(120)

self.tcp\_server\_socket = tcp\_server\_socket

def start(self):

while True:

new\_client\_socket, ip\_port = self.tcp\_server\_socket.accept()

self.request\_handler(new\_client\_socket, ip\_port)

def request\_handler(self, new\_client\_socket, ip\_port):

print("%s用户已介入" %(str(ip\_port)))

request\_data = new\_client\_socket.recv(1024)

if not request\_data:

print("已断开")

new\_client\_socket.close()

return

#根据客户端浏览器请求路径返回信息

request\_text = request\_data.decode()

loc = request\_text.find("\r\n")

request\_line\_list = request\_text[:loc].split(" ")

file\_path = request\_line\_list[1]

#设置默认首页

if file\_path == "/":

file\_path = "/grand.html"

response\_line = "HTTP/1.1 200 ok\r\n"

response\_header = "Server:daomeng\r\n"

response\_blank = "\r\n"

#返回固定页面

try:

with open("E:/pythonProject1/game"+file\_path, "rb") as file:

response\_body = file.read()

except Exception as s:

response\_line = "HTTP/1.1 404 Not Found\r\n"

response\_body = "ERROR! (%s)"% str(s)

response\_body = response\_body.encode()

response\_data = (response\_line+response\_header+response\_blank).encode()+response\_body

new\_client\_socket.send(response\_data)

def main():

ws = WebServer()

ws.start()

if \_\_name\_\_ == '\_\_main\_\_':

main()

—————————————————————————

>>服务器基础框架构建（第三十六课）

所有代码放在服务器文件夹里了

—————————————————————————

>>使用终端中启动Web服务器（第三十七课）

1. argv

- 一般在命令行调用的时候由系统传递给程序，这个变量是个 List 列表

2.isdigit函数

- python isdigit 方法检测字符串是否只有数字组成，如果字符串只包含数字则返回 True

否则返回 False

—————————————————————————

>>使用终端中启动Web服务器（第三十七课）

1.dict() 空字典 == { }

2. .keys()方法可以返回字典的键

dict\_keys(['grand', 'grand2', 'grand3', 'grand4', 'grand5'])

3.enumerate(keys\_list) 会把列表组合为一个索引序列

4.Contect-Type:text/html 服务器响应头加入这句话，表示服务器优先使用html来

解析服务器返回的内容

—————————————————————————

>>网游服务器（第三十八课）

所有代码放在网游服务器文件夹里了

—————————————————————————

>>time模块（第三十九课）

**- import time**

**time.sleep(1) - 睡眠1秒**

—————————————————————————

>>多任务的介绍（第四十课）

- 知道多任务的概念

- 知道多任务与单任务程序的区别

多任务：同一时间，多个任务同时执行

**python 默认单任务**

—————————————————————————

>>线程 - 基本使用（第四十一课）

import threading

- 能够理解主线程和子线程关系

- 知道使用 threading.Thread 类能够创建线程对象

- 知道 threading.Thread 的 target 参数能够指定线程执行的人物

1.线程概念

**- 可以简单理解为程序执行的一条分支**

**- 是被系统独立调度和分派的基本单位**

- 返回值为对象

thread\_obj = threading.Thread(target=函数名) - 无括号,函数名

thread\_obj.start()

**- 主线程会在所有子线程结束后才结束**

—————————————————————————

>>线程 - 数量（第四十二课）

- 能够如何查看正在活动的线程数量

**1.查看线程数量**

**threading.enumerate( )**

**- 获取当前所有活跃的线程对象列表，使用len( ),对列表求长度可以看到当前的**

活跃线程个数

**2.查看线程名称**

**threading.current\_thread( )**

**- 获取当前线程对象**

—————————————————————————

>>线程 - 参数及顺序（第四十三课）

**- 能够向线程函数传递多个参数**

在线程中，传递参数有**三种方法**

**- 元组传递**

1.threading.Thread(target=xxx, args=(参数一，参数二......))

**- 字典传递**

2.threading.Thread(target=xxx, kwargs{"字典名":字典值,......})

**- 混合传递(元组与字典传递)**

3.threading.Thread(target=xxx, args=(参数一，参数二......), kwargs{"字典名":字典值,......})

- 能够说出多线程执行的顺序特点

—————————————————————————

>>线程 - 守护线程（第四十四课）

**- 使用 setDaemon 设置子线程守护主线程**

守护线程：如果在程序中将子线程设置为守护线程，则该子线程**会在主线程结束后自动退出**

设置方式：

**变量.setDaemon(True)**

—————————————————————————

>>线程 - 并行和并发（第四十五课）

- 能够说出并行和并发的区别

**1.并发：指的是任务数大于 cpu 核数，通过操作系统的各种调度算法，实现多个任务**

**"一起执行"，（切换速度相当快，所以认为在执行）**

真正的**并行执行多任务只能在多核cpu上实现**，但是由于任务数量远远多于cpu核心数

所以操作系统也会自动把很多任务轮流调度到每个核上运行

总结：**任务数量大于cpu核心数**

2.并行：指的是任务数小于等于cpu核心数，即任务真的是一起执行的

总结：**任务数量小于等于cpu核心数**

—————————————————————————

>>线程 - 自定义线程类（第四十六课）

- 知道通过继承 threading.Thread 可以实现自定义线程

- 导入模块

**- 创建类并且继承 threading.Thread( )**

- 重写父类 run( ) 方法

- 创建对象并且调用.start()

**- 总体来说应该是方便使用**

—————————————————————————

>>线程 - 共享全局变量（第四十七课）

- 知道多线程能够共享全局变量数据

**- 多线程之间可以共享全局变量**

—————————————————————————

>>线程 - 问题（第四十八课）

- 知道多线程共享全局变量数据会导致资源竞争问题

**- .join()：提权，优先让一个线程执行完毕**

—————————————————————————

>>线程 - 同步的概念（第四十九课）

- 能够使用 setDaemon 设置子线程守护主线程

**1.同步与异步**

**同步：多任务，多个任务之间执行要求先后顺序，必须一个执行完毕之后另一个才会执行**

**只有一个主线：如你说完，我再说（同一时间只做一件事情）**

**异步：多个任务之间没有先后顺序，可以同时执行，执行的先后顺序不会有什么影响**

**存在多条运行主线，如：发微信（可以不用等回复，继续发）**

—————————————————————————

>>线程 - 互斥锁（第五十课）

1.概念：

- 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，最简单的同步机制是引入互斥锁**

- 互斥锁为资源引入一个状态：**锁定/非锁定**

lock1 = threading.Lock() - 创建一把锁

lock1.acquire() - 上锁

lock1.release() - 解锁

—————————————————————————

>>线程 - 死锁（第五十一课）

- 知道多线程程序要避免死锁

在多线程中，两个线程都占用一些资源，而且**同一时间都在等待对方释放资源**

这种状态就叫做死锁

—————————————————————————

>>多任务版UDP聊天器（第五十二课）

import socket

import threading

def send\_message(udp\_socket):

"""返送信息的函数"""

# 1) 定义变量接收用户与输入的接收方的IP地址

ipaddr = input("请输入接收方的IP地址：\n")

# 判断是否需要默认

if len(ipaddr) == 0:

ipaddr = "192.168.199.132"

print("当前接收方默认IP设置为[%s]" % ipaddr)

# 2）定义变量接收用户与输入的接收方的端口号

port = input("请输入接收方的端口号：\n")

if len(port) == 0:

port = "8888"

print("当前接收方默认端口设置为[%s]" % port)

# 3)定义变量接收用户与输入的接收方的内容

content = input("请输入要发送的内容：\n")

# 4）使用socket的sendto()发送信息

udp\_socket.sendto(content.encode(), (ipaddr, int(port)))

def recv\_msg(udp\_socket):

while True:

"""接收信息的函数"""

# 1) 使用socket接收数据

recv\_data, ip\_port = udp\_socket.recvfrom(1024)

# 2）解码数据

recv\_txt = recv\_data.decode()

# 3）输出显示

print('收到', ip\_port, '发送的消息：', recv\_txt)

def main():

"""程序主入口"""

# 1）创建套接字

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

# 2）绑定端口

udp\_socket.bind(("", 8080))

thread\_recvmsg=threading.Thread(target=recv\_msg, args=(udp\_socket,))

thread\_recvmsg.setDaemon(True)

thread\_recvmsg.start()

# 3）打印菜单（循环）

while True:

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('\*\*\*\*\*\*\* 1、发送信息 \*\*\*\*\*\*\*')

print('\*\*\*\*\*\*\* 2、退出系统 \*\*\*\*\*\*\*')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

# 4）接收用户输入的选项

sel\_num = int(input('请输入选项：\n'))

# 5）判断用户的选择，并且调用对应的函数

if sel\_num == 1:

print('您选择的是发送信息')

send\_message(udp\_socket)

elif sel\_num == 2:

print('系统正在退出中...')

print('退出完成！')

break

else:

print('选项输入错误，请重新输入')

# 6）关闭套接字

udp\_socket.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

# 程序独立运行的时候，才去启动聊天器

—————————————————————————

>>TCP服务端框架（第五十三课）

- 能够使用多线程实现同时接受多个客户端的多条消息

import socket

import threading

def recv\_msg(new\_client\_socket):

while True:

recv\_data = new\_client\_socket.recv(1024)

if recv\_data:

recv\_text = recv\_data.decode("GBK")

print(recv\_text)

else:

break

new\_client\_socket.close()

tcp\_server\_socket= socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, True)

tcp\_server\_socket.bind(("", 8080))

tcp\_server\_socket.listen(100)

while True:

def xiancheng():

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

recv\_msg(new\_client\_socket)

s = threading.Thread(target=xiancheng)

s.start()

**- 总结，当遇到任务分配给子线程的代码时，主线程会跳过这一部分然后继续往下执行**

就是说当主线程创建了一条线时，遇到分配给子线程的命令后，**会越过这段代码**

然后返回上面继续待命

- 每来一个客户端就建立一个新的线程

—————————————————————————

>>内容回顾-查漏补缺（第五十四课）

在自定义线程类的过程中，实则线程类的功能比单程的线程更加强大

**记住底层原理：在 start() 方法里会自动调用 run( )方法**

—————————————————————————

>>多任务-进程以及状态（第五十五课）

- 知道多进程也是实现多任务的一种方式

1.进程概念：

**- 进程是资源分配的最小单位，也是线程的容器**

- 程序是固定不变的，**而进程会根据运行需要，让操作系统动态分配各种资源的**

2.进程：**资源分配的基本单位，也是线程的容器**

—————————————————————————

>>进程-基本使用（第五十六课）

- 知道使用 multiprocessing.Process类能够创建对象

- 能够 multiprocessing.Process 的 target 参数能够指定进程执行的任务函数

- 注意大小写

import multiprocessing

process\_obj = multiprocessing.Process(target=work1)

process\_obj.start()

—————————————————————————

>>进程-名称 : pid（第五十七课）

**- 能够使用 getpid 和 gitppid 的获取进程 id 和进程父 id**

**- name: 指定进程的名称**

1.进程名称获取

multiprocessing.current\_process()

2.获取进程编号

multiprocessing.current\_process().pid

2.1.使用os模块来获取进程编号

import os

os.getpid()

3.获取进程的父id

import os

os.getppid()

—————————————————————————

>>进程-参数传递,全局变量问题（第五十八课）

**- 多进程之间不能共享全局变量**

**- 能够给子线程指定的函数传递参数**

1.子线程传递参数

**- 给子进程传递参数方法和给子线程传递参数方法基本一致**

2.进程之间不会共享全局变量

**- 第一个进程修改的值，第二个进程不受影响**

—————————————————————————

>>进程-守护主进程（第五十九课）

1.守护主线程

- 变量.daemon = true

2.结束进程

- 变量.terminate()

—————————————————————————

>>进程,线程对比（第六十课）

**- 知道多进程和多线程的区别**

**1.功能**

**- 进程：能够完成任务，比如在一台电脑上同时运行多个QQ**

**- 线程：能够完成任务，比如一个QQ中多个聊天窗口**

- 进程是系统进行资源分配和调度的一个独立的单位

- 线程是进程的一个实体，是cpu调度和分配的基本单位

**对于多个文件操作时，可以使用多线程**

**对于单个文件多操作时，可以使用多进程**

多线程在参与大量的运算时，只会调用一个cpu核心，I/O时除外

多进程在参与大量的单独运算时，回调用全部的核心

**- 进程切换慢，线程切换快**

**- 线程不能独立运行，必须运行在线程中(线程提供资源)**

- cpu密集型 进程优先

- I/O密集型使用线程

- 程序更稳定：进程，线程相比较不稳定

—————————————————————————

>>消息队列（第六十一课）

- 知道 queue 的 put() 和 get() 作用

**process 之间有时需要通信，操作系统提供了很多机制来实现进程间通信**

- Queue 介绍

使用 multiprocessing 模块的 Queue 实现多进程之间数据传递

Queue本身就是个消息队列程序

**put() - 放入值**

**get() - 取值**

1.放值

s = multiprocessing.Queue( n )

n - 表示队列长度最大可以放多少消息

s.put( 值 ) - 向队列中放值

s.put\_nowait( 值 ) - 多了会直接报错

解释：当设定长度为5时，放入第六个值，**队列就进入阻塞状态，**

默认会等待队列先取出值再放入值，put\_nowait

**意思是不会等待，假设满了五个则不会进入等待状态**

**而是直接报错**

2.取值

value = queue.get(),取值之后会保存到变量，**接受队列头**

当队列已空时再去取值，**程序会进入阻塞状态**

知道放入新的值

queue.get\_nowait()

**取值不在等待**，跟上文描述情况差不多

—————————————————————————

>>消息队列 - 常见判断（第六十二课）

- queue 的 full() 与 empty() 的作用

- qsize() 获取队列的消息个数

**1.判断队列是否已满**

**- queue.full() - 返回布尔值**

**2.获取队列中消息的个数**

**- queue.qsize()**

**3.判断队列是否为空**

**queue.empty() - 返回布尔值**

**关于为什么会有输出错误信息**

**queue = multiprocessing.Queue(3)**

**queue.put(1)**

**queue.put(2)**

**queue.put(3)**

**print(queue.empty())**

**输出：True**

**- 可以理解为放值与取状态同时进行的，可以可以认为两个进程同时进行的**

**如果要避免的话就别写到及近的位置**

—————————————————————————

>>实现进程间通信（第六十三课）

- 知道使用 Queue 队列能够实现进程间数据共享

import multiprocessing

import time

def write\_queue(queue):

for i in range(10):

if queue.full():

print("队列满了")

break

queue.put(i)

print("写入成功",i)

time.sleep(0.5)

def read\_queue(queue):

while True:

if queue.qsize() == 0:

print("队列空了")

break

value = queue.get()

print("已经读取",value)

if \_\_name\_\_ == '\_\_main\_\_':

queue = multiprocessing.Queue(5)

write\_queue1=multiprocessing.Process(target=write\_queue, args=(queue,))

read\_queue1=multiprocessing.Process(target=read\_queue, args=(queue,))

write\_queue1.start()

write\_queue1.join()

read\_queue1.start()

—————————————————————————

>>进程池Pool（第六十四课）

- 知道进程池的作用

1.进程池概述

**- 当需要创建的子进程数量不多时，可以利用 multiprocessing 中的 Process 动态**

生成多个进程，但如果是上百甚至是上前个目标，手动的去创建进程的工作量巨大

**此时可以用到 multiprocessing 模块提供的Pool方法**

**- apply( ) = 进程池中以同步方式执行任务**

创建进程池 两个进程

pool = multiprocessing.Pool(2)

用进程池去完成事情

1.同步

pool.apply( 函数名，（传递给函数的参数1，传递给函数的参数2）) - 同步

- 不用去关闭

2.异步

pool.apply\_async(( 函数名，（传递给函数的参数1，传递给函数的参数2）) - 异步

- 如果使用异步的方式，则必须有 pool.close( ) 来关闭进程池接受新任务

- 如果以异步方式运行，我们的主线程不在等待进程池执行结束后再退出

需要进程池 join() 一次

- pool.join 让主进程等待进程池执行结束后退出

例子：

if \_\_name\_\_ == '\_\_main\_\_':

pool = multiprocessing.Pool(10)

for i in range(10):

# pool.apply(copy\_work) - 同步

pool.apply\_async(copy\_work) - 异步

pool.close()

pool.join()

—————————————————————————

>>进程池中的Queue（第六十五课）

- 知道实现进程池中进程间通信

用法：queue = multiprocessing.Manager().Queue( 3 )

apply\_async() 返回值 ApplyResult 对象，该对象有一个 wait() 方法

**该方法类似 join() 方法**

result = pool.apply\_async(write\_queue, (queue,))

result.wait()

—————————————————————————

>>文件夹 copy 器（第六十六课）

**- 导入os模块**

1.创建文件夹

os.mkdir( 文件路径 ) = 创建一个文件夹

例子：

first\_txt = "C:/Users/Lenovo/Desktop/test"

os.mkdir(first\_txt)

2.扫描文件夹

os.listdir(source\_dir) = 扫描文件夹的所有东西

代码：

import multiprocessing

import os

def copy\_work(source\_dir, dest\_dir, file\_name):

source\_path = source\_dir + "/" + file\_name

dest\_path = dest\_dir + "/" + file\_name

print(source\_path,"||||这是分段||||",dest\_path)

with open(source\_path,"rb") as source\_file:

with open(dest\_path,"wb") as dest\_file:

while True:

file\_data = source\_file.read(1024)

if file\_data:

dest\_file.write(file\_data)

else:

break

if \_\_name\_\_ == '\_\_main\_\_':

# 1.定义源文件，目标文件所在路径

source\_dir = "C:/Program Files (x86)/Huorong/Sysdiag/bin"

dest\_dir = "C:/Users/Lenovo/Desktop/test"

# 2.在目标文件创建新的文件夹

try:

os.mkdir(dest\_dir)

except FileExistsError:

print("您已经创建过了")

# 3.获取源文件夹的所有文件

file\_list = os.listdir(source\_dir)

print(file\_list)

pool = multiprocessing.Pool(10)

# 4.遍历列表，得到所有文件名

for file\_name in file\_list:

# 5.定义函数进行文件拷贝

list =pool.apply\_async(copy\_work,(source\_dir,dest\_dir,file\_name))

pool.close()

pool.join()

- 最近看的有点快了，得沉淀下了

—————————————————————————

>>简单Web服务器多进程版（第六十七课）

- 当进程调用了一个对象时，最好是直接关闭一次，黑马二部分7.14

def start(self):

while True:

new\_client\_socket, ip\_port = self.tcp\_server\_socket.accept()

s = multiprocessing.Process(target=self.request\_handler ,args=(new\_client\_socket,ip\_port))

s.start()

new\_client\_socket.close()

new\_client\_socket.close()

—————————————————————————

>>可迭代对象及检测方法（第六十八课）

- 知道什么是可迭代对象

- 能够使用 instance( ) 检测对象是否可迭代

**迭代是访问集合元素的一种方式，迭代器是一个可以记住遍历位置的对象**

**迭代器对象从集合的第一个元素开始访问，知道所有元素被访问完结束，**

**迭代器只能往前不会后退**

**- instance可以去查询是否可以迭代(非递归)，可以就返回 True 不行就 False**

- from collections import Iterable

**- instance(待检测对象，iterable)**

**- iterable = 检测前者是不是后者的子类**

例子：

from collections.abc import Iterable

ret = isinstance([1,2,3], Iterable)

print(ret)

结论：

**1.可遍历对象就是可迭代对象**

**2.列表，元组，字典，字符串，都是可迭代对象**

**3.100 和 自定义 myclass 默认都是不可迭代的**

**4.myclass 对象所属的类 Myclass 如果包含了 \_\_iter\_\_( ) 方法，此时 myclass**

**就是一个可迭代对象**

**5.可迭代对象的本质：对象所属的类中包含了 \_\_iter\_\_( ) 方法**

**6.检测一个对象是否可以迭代，用 isinstance( ) 函数检测**

—————————————————————————

>>迭代器及其使用方法（第六十九课）

- 知道使用 iter() 函数可以获得可迭代对象的迭代器

- 知道使用 next() 函数可以获得迭代器数据

**1.iter() 函数与 next() 函数**

**iter( 可迭代对象 ) ——》 获取对象的迭代器**

**next( 迭代器 ) ——》获取下一个元素**

——————————————————————

from collections.abc import Iterable

a = [1,2,3]

ret = isinstance(a, Iterable)

print(next(iter(a)))

输出：1

总结:

1.记录遍历的位置

2.提供给下一个元素的值(配合 next( ) 函数)

3.for循环本质：

1.通过 iter(要遍历的对象) 获取要遍历的对象的迭代器

2.next(迭代器) 获取下一个元素

3.帮我们捕获了 StopIteration 异常

自定义迭代器类，满足两点

**1.必须含有 \_\_iter\_\_()**

**2.必须含有 \_\_next\_\_()**

—————————————————————————

>>自定义迭代对象，迭代器（第七十课）

class My\_List(object):

def \_\_init\_\_(self):

self.list = []

def \_\_iter\_\_(self):

my\_list\_script = My\_List\_Script(self.list)

return my\_list\_script

def add\_ok(self, name):

self.list.append(name)

class My\_List\_Script(object):

def \_\_init\_\_(self,list):

self.my\_list = 0

self.list = list

def \_\_iter\_\_(self):

pass

def \_\_next\_\_(self):

if self.my\_list < len(self.list):

s = self.list[self.my\_list]

self.my\_list += 1

return s

else:

raise StopIteration

if \_\_name\_\_ == '\_\_main\_\_':

mylist = My\_List()

mylist.add\_ok("Just Monika")

mylist.add\_ok("Just Yuri")

mylist.add\_ok("Just Sayori")

mylist.add\_ok("Just Natsuki")

for i in mylist:

print(i)

—————————————————————————

>>斐波那契数列（第七十一课）

- 找到自身的问题了......

- 迭代器本身就是一个迭代器

- a 保存第一列的值，b保存第二列的值

**- a,b=b,a+b**

class List():

def \_\_init\_\_(self,num):

self.a = 0

self.b = 1

self.num = num

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

a\_num = self.a

a\_num += 1

if a\_num < self.num:

self.a,self.b = self.b,self.a+self.b

return self.a

else:

raise StopIteration

if \_\_name\_\_ == '\_\_main\_\_':

list = List(1000)

for i in list:

print(i)

—————————————————————————

>>生成器（第七十二课）

- 生成器就是特殊的迭代器

- 能够用两种方法创建生成器

1.列表推导式

2.函数中使用 yield

- 列表推导式 (扩展)

- data\_list = [i\*2 for i in range(10)]

- data\_list = (i\*2 for i in range(10))

- 函数生成器

def test():

yield 10

n = test()

print(n)

**yield = 1.能够充当 return 的作用**

**2.能够保存程序的状态**

**3.下一次从 yield 位置向下执行**

**4.停止后，yield 会自动保存当前的所有变量属性**

—————————————————————————

>>生成器 - 使用注意（第七十三课）

- 知道使用 send 方法能够启动生成器，并传递参数

- 能够说出生成器中 return 的作用

**1.生成器中使用 return 的问题**

**- 生成器中使用 return 语法上没有问题，但是如果执行到 return 之后**

**生成器会停止迭代，抛出停止迭代的异常**

**2.send() - 传参数**

例子：

def fibnacci(n):

a = 1

b = 1

current\_index = 0

while current\_index < n :

data = a

a, b = b, a+b

current\_index += 1

num = yield data

if num == 1:

return "停止执行吧"

else:

raise StopIteration

if \_\_name\_\_ == '\_\_main\_\_':

fib = fibnacci(5)

value = next(fib)

print(value)

try:

value = next(fib)

print(value)

value = next(fib)

print(value)

value = fib.send(1)

print(value)

except Exception as c:

print(c)

—————————————————————————

>>协程（第七十四课）

- 知道使用 yield 关键字可以实现协程

- 在不开辟新线程的基础上，实现多个任务

- 线程是一个特殊的生成器

应用场景：

**- 当程序中存在大量不需要 CPU 操作时(IO),适用于协程**

**- 在一个线程中的某个函数，可以在任何地方保存当前函数的一些临时变量等信息，**

**然后换到另外一个函数执行，**

—————————————————————————

>>协程-greenlet（第七十五课）

- 知道使用 greenlet 可以实现协程

from greenlet import greenlet

1.greenlet 是一个第三方模块，

- 导入模块 from greenlet import greenlet

- 创建 greenlet 对象，返回对象

- 切换任务 - g1.switch()

代码：

from greenlet import greenlet

import time

def work1():

print("work1即将开始执行")

while True:

print("正在执行work1")

time.sleep(0.5)

g2.switch()

def work2():

print("work2即将开始执行")

while True:

print("正在执行work2")

time.sleep(0.5)

g1.switch()

if \_\_name\_\_ == '\_\_main\_\_':

g1 = greenlet(work1)

g2 = greenlet(work2)

g1.switch()

—————————————————————————

>>协程-gevent（第七十六课）

import gevent

**- gevent.spawn(函数名，参数1，参数2) - 返回参数**

**- 参数.join - 会让程序进入阻塞状态**

- gevent内置了sleep状态，他认为time.sleep(0.5)不算阻塞

- gevent.sleep(0.5)

**- 默认情况下，time.sleep() 不能被 gevent 识别为耗时操作**

1.把 time.sleep() ——》gevent.sleep()

**2.给 gevent 打补丁(让 gevent 识别 gevent.sleep())**

**打补丁：**

- 在不修改程序源代码的情况下，为程序增加新的功能

如何打补丁？

**1.导入模块 monkey 模块 from gevent import monkey**

**2.破解 monkey.patch\_all()**

**- 一般放在开头**

**- gevent.getcurrent( ) - 查看当前执行任务的协程**

**个人理解：好像只有耗时操作时，gevent才会切换**

代码：

import gevent

def work1():

print("work1即将开始执行")

while True:

print("正在执行work1")

gevent.sleep(0.5)

def work2():

print("work2即将开始执行")

while True:

print("正在执行work2")

gevent.sleep(0.5)

if \_\_name\_\_ == '\_\_main\_\_':

g1 = gevent.spawn(work1)

g2 = gevent.spawn(work2)

g1.join()

g2.join()

—————————————————————————

>>进程-线程-协程对比（第七十七课）

- 能够说出进程，线程，协程对比

**高效率方式：进程+协程**

—————————————————————————

>>案例 - 并发下载器（第七十八课）

**- urllib.request.urlopen( )**

**- 打开网址并返回对应内容(二进制流)**

**- gevent.joinall([])**

**- 批量 .join**

**gevent.joinall([**

**gevent.spawn(download\_img, img\_url1, "name.jpg"),**

**gevent.spawn(download\_img, img\_url2, "name.jpg1"),**

**gevent.spawn(download\_img, img\_url4, "name.jpg2")**

**])**

import gevent

from gevent import monkey

monkey.patch\_all()

import urllib.request

def download\_img(imgUrl,file\_name):

try:

responce\_data = urllib.request.urlopen(imgUrl)

with open(file\_name, "wb") as file:

while True:

file\_data = responce\_data.read(1024)

if file\_data:

file.write(file\_data)

else:

break

except Exception as e:

print("文件下载失败",file\_name,e)

else:

print("文件下载成功",file\_name)

def main():

img\_url1 = "https://tse1-mm.cn.bing.net/th/id/R-C.daf3f6e4f1a457618e3e2d4ab0c66383?rik=5oaQZeiTwVy8Gg&riu=http%3a%2f%2fdaocheng.mjjq.com%2fpic%2f051219-cdc%2f1135030215\_1005447294.jpg&ehk=C1tyN1ZHLkGCvDz8r1YxlDFWbZSVEslwN3VzNSohoLs%3d&risl=&pid=ImgRaw&r=0"

img\_url2 = "https://tse1-mm.cn.bing.net/th/id/OIP-C.nRlAFygdctTCHmIWN7GxRwHaEK?pid=ImgDet&rs=1"

img\_url4 = "https://tse1-mm.cn.bing.net/th/id/R-C.2200f336cb7cc8f21798e7d9fe9b24f5?rik=6HjanVulzGkvpA&riu=http%3a%2f%2fwww.desktx.com%2fd%2ffile%2fwallpaper%2fscenery%2f20170107%2f080145c3a7460e7fa0369052a11467db.jpg&ehk=ffhPqQk04pZ18gOXu3OrPfkEIoQW046d4EfbUzji9yI%3d&risl=&pid=ImgRaw&r=0"

gevent.joinall([

gevent.spawn(download\_img, img\_url1, "name.jpg"),

gevent.spawn(download\_img, img\_url2, "name1.jpg"),

gevent.spawn(download\_img, img\_url4, "name2.jpg")

])

if \_\_name\_\_ == '\_\_main\_\_':

main()

—————————————————————————

>>正则表达式概述（第七十九课）

- 一套特殊的规则

这里不写了，我先学的 JS 正则

—————————————————————————

>>匹配单个字符（第八十课）

- 知道正则表达式能够匹配单个字符

字符 功能

. 匹配任意1个字符(除了\n)

[ ] 匹配[ ] 中列举的字符

\d 匹配数字，即0 - 9

\D 匹配非数字，即不是数字

\s 匹配空白，即空格，\t-tab键 \n-换行

\S 匹配非空白

\w 匹配单词字符，即a-z,A-Z,0-9,\_

\W 匹配非单词字符

—————————————————————————

>>匹配多个字符（第八十一课）

**字符 功能**

**\* 表示前一个字符出现0次或无限次**

**+ 表示前一个字符出现1次或无限次**

**？ 表示前一个字符出现0次或1次**

**{m} 表示前一个字符，连续出现m次**

**{m,n} -m一定小于n 表示前一个字符，连续出现最少m次，最多n次**

—————————————————————————

>>匹配开头结尾（第八十二课）

**字符 功能**

**^ 匹配字符串开头，注意^[4-7]和[^4-7]的区别**

**$ 匹配字符串结尾**

注意^[4-7]和[^4-7]的区别：

- 放在[]外面是以它开头，

- 放在[]里面是取反，不包含[]里的内容

—————————————————————————

>>re模块操作（第八十三课）

- 知道 re.match 方法的作用

- 知道 group 方法的作用

**re.match(pattern,string,falgs)**

**- 从开头就搜索，不是就返回 None**

**1.pattern 正则模型**

**2.string 要匹配的字符串**

**3.falgs 匹配模式**

**match()方法一旦匹配成功，返回一个match object对象，而match object有一下方法**

**1.group( ) 返回被 RE 匹配的字符串**

**2.start( ) 返回匹配开始的位置**

**3.end( ) 返回匹配结果的位置**

**4.span( ) 返回一个元组包含匹配(开始,结束)的位置**

例子：

import re

result = re.match("itcast", "itcast.com",)

if result:

print(result.group())

—————————————————————————

>>匹配分组"|"（第八十四课）

**字符 功能**

**| 匹配左右任意一个表达式 - 可以理解为 正则 或 正则**

**(ab) 将括号中字符作为一个分组**

**\num 引用分组 num 匹配到的字符串**

**(?P<name>) 分组起别名**

**(?P=name) 引用别名为 name 分组匹配到的字符串**

—————————————————————————

>>匹配分组"( )"（第八十五课）

**- 知道正则中 ( ) 的作用**

**1 2**

**\w(4,20)@(163|126|qq)\.com**

**( ) - 作为一个整体去检查**

检查：hello@163.com

2 - ( 163 )

—————————————————————————

>>知道(?P<name>)起名子（第八十六课）

\ - 引用 - 引用( 数字 )的东西

例子：

<([A-z0-9]+)>.\*<\1>

注意转义

2.给它起名字

- ?P<name1> 给i分组起别名，别名为 name1

- ?P=name1 引用别名为 name1 的分组

match("<(?P<nam1>[正则语法]+")

引用: </(?P=name1)>

—————————————————————————

>>re模块的高级用法（第八十七课）

- 知道 re 模块 **search**  方法的作用

- 知道 re 模块  **sub** 方法的作用

- 知道 re 模块  **split**  方法的作用

- 知道 re 模块 **findall**  方法的作用

1.search 搜索匹配

- re.search 函数会在字符串内查找模式匹配，只要找到第一个匹配然后返回

如果字符串没有匹配，则返回 None

- search 就是搜索字符串中有没有满足的东西

- match 从需要检查的字符串的开头去匹配

格式: re.search(pattern,string,flags=0)

2.findall 查找所有，返回列表

3.sub 字符串替换

- 按照正则，查找字符串并且替换为指定的内容

- sub("正则表达式","新的内容","要替换的字符串")

- 返回替换后的字符串

4.split("正则表达式","待拆分的字符串")

按照正则拆分字符串，返回值是一个列表

—————————————————————————

>>贪婪和非贪婪（第八十八课）

- 知道贪婪匹配的特点

**贪婪：在规则正确时，尽可能的多匹配字符**

**非贪婪：在规则正确时，尝试匹配尽可能少的字符**

转变方式：在 +\*?( ) 后面添加 ？可以转变

—————————————————————————

>>r的作用（第八十九课）

1.让正则中的 \ 不在具有转义功能

用法:

result = re.match(**r**"正则")

—————————————————————————

>>批量获取电影下载链接（第九十课）

- 我的第一个爬虫

import requests

import re

url = "C:/Users/Lenovo/Desktop/电影.txt"

response\_list\_text = url

with open("C:/Users/Lenovo/Desktop/dianyingtiantang.txt","w")as test\_file:

with open(url,"r",encoding="UTF-8") as file:

while True:

s = file.read()

if s:

text = re.findall(r"<a href=\"(.\*)\" title=\"(.\*)\">(.\*)</a>",s)

text\_new = re.sub(",","\\n",str(text))

print(text\_new)

—————————————————————————

## 以下内容转为MySQL 所以重开一个word文档

—————————————————————————

>> Python 连接 MySQL（第九十一课）

- 能够说出 python 操作数据库的步骤

- 能够 **connect** 方法创建连接对象

- 能够使用连接对象的 **cursor( )** 方法 创建游标对象

- 能够使用游标对象的 **exeute( )** 方法 执行MySQL语句

- 能够使用游标对象的 **close( )** 方法 关闭游标对象

- 能够使用连接对象的 **close( )** 方法 关闭连接对象

用于建立于数据库的连接调用 pymysql 中的 conneat() 方法

**conn = connect( 参数列表 )**

参数 host : 连接 mysql 的主机, 如果本机是 localhost

参数 port : 连接 mysql 主机的端口,默认是 3306 ( 可以不写 )

参数 databases : 数据库的名称

参数 user : 连接的用户名

参数 password : 连接的密码

参数 charset : 通信采用的编码方式,推荐使用 utf8( 可以不写,默认utf8 )

关闭连接 conn.close( )

提交数据 conn.commit( )

撤销数据 conn.rollback( )

**通过连接获取游标 cur = conn.cursor( ) 返回 Cursor 对象,用于执行 sql 语句并获得结果**

获取结果集中的一条

cur.fetchone() 返回一个元组 形如(1,"沢渡雫",18)

获取结果集中的所有

cur.fetchall() 执行查询时,获取结果集的所有行,一行构成一个元组

再将这些元组装入一个元组中返回

( (1,"桃夜歌",18 )( 2,"美咕噜",18 ) )

写入SQL语句的方法

cur.execute() - 参数就是 SQL 返回值是被影响的行数

例子：

import pymysql

conn = pymysql.connect(host='localhost', user='root', password='sxhmzz840845135', database='jing\_dong')

cur = conn.cursor()

result = cur.execute("select \* from goods order by id desc")

print(result)

result\_list = cur.fetchall()

# print(result\_list)

for line in result\_list:

print(line)

cur.close()

conn.close()

—————————————————————————

>> Python 操作数据库（第九十二课）

- 能够使用游标对象去操作数据库进行增删改查

例子：

import pymysql

conn = pymysql.connect(host='localhost', user='root', password='sxhmzz840845135', database='jing\_dong')

cur = conn.cursor()

sql = "insert into goods values(null, '老王牌拖拉机' , 1,1,9998,1,1)"

ret = cur.execute(sql)

conn.commit()

print("影响了",ret,"行")

cur.close()

conn.close()

—————————————————————————

>> SQL防注入（第九十三课）

- 能够说出如何避免 SQL 注入问题

**- 1.需要变化的地方可以使用占位符 %s %d**

**- 把参数封装到列表中**

**- 把列表传递给 execute(sql,列表)**

input\_name = input("请输入名字")

input\_list = [input\_name]

sql = "select \* from goods where name = %s"

result = cur.execute(sql , input\_list)

—————————————————————————

>> 爬取数据并保存到数据库中（第九十四课）

- 能够创建案例所需要的数据库，数据表

- 能够将采集的电影信息保存到数据库中

- 自己写的代码：

import requests

import re

import pymysql

import os

import time

url\_01 = "https://qingju.ga/Topping/"

xiazaiweizhi = "C:/Users/Lenovo/Desktop/在青桔爬取的文件"

url = "C:/Users/Lenovo/Desktop/在青桔爬取的文件/requests.txt"

result = requests.get(url=url\_01)

result\_text = result.text

try:

os.mkdir(xiazaiweizhi)

except FileExistsError:

pass

with open(url,"w", encoding="UTF-8") as file:

file.write(result\_text)

time.sleep(0.5)

class Except\_center():

def \_\_init\_\_(self,url):

self.url = url

self.conn = pymysql.connect(host='localhost', user='root', password='sxhmzz840845135', database='galgame')

self.result\_update = self.conn.cursor()

self.sql = "select game\_name from game;"

self.sql\_01 = 'create table test(fenqu varchar(20) not null,game\_name varchar(50) not null,lianjie varchar(100) not null)'

self.sql\_03 = 'select test.fenqu,test.game\_name,test.lianjie from test left join game on test.game\_name = game.game\_name where game.game\_name is null'

self.list\_01 = []

self.n = 0

self.b = 0

def find(self):

self.result\_update.execute(self.sql)

for i in self.result\_update.fetchall():

for a in i:

self.list\_01.append(a)

with open(url, "r", encoding="UTF-8") as files:

s = files.read()

if files:

try:

self.result\_update.execute(self.sql\_01)

self.conn.commit()

print("test表以成功创建")

except:

pass

text = re.findall(

r'<strong>(.+?)</strong></font><br><font color=#FF0000 size=3><strong>.+?</strong></font></p>|<td align="center"><a href="(https://.+?)">(.+?)<',

s)

for qu, lianjie, game\_name in text:

if qu and not lianjie and not game\_name:

qu\_name = qu

continue

sql\_02 = "insert into test values('%s','%s','%s')" % (qu\_name, game\_name, lianjie)

self.result\_update.execute(sql\_02)

self.conn.commit()

self.result\_update.execute(self.sql\_03)

self.group = []

for q in self.result\_update.fetchall():

self.group.append(q)

def delete\_test(self):

sql\_04\_delete\_test = 'drop table test'

self.result\_update.execute(sql\_04\_delete\_test)

self.conn.commit()

print("test文件清除成功")

self.result\_update.close()

self.conn.close()

class Modification(Except\_center):

def \_\_init\_\_(self,except\_center):

super().\_\_init\_\_(self)

self.except\_center = except\_center.group

self.sql = 'select id,name from fenqu;'

self.result\_update.execute(self.sql)

self.center = []

def update(self):

for i in self.result\_update.fetchall():

for self.fenqu, self.name, self.url in self.except\_center:

if i[1] == self.fenqu:

self.fenqu = i[0]

sql = "insert into game values(null,'%d','%s','%s');" % (self.fenqu, self.name, self.url)

self.result\_update.execute(sql)

self.conn.commit()

print("最新内容为:")

print(self.fenqu, self.name, self.url)

except\_center = Except\_center(url)

except\_center.find()

modification = Modification(except\_center)

modification.update()

except\_center.delete\_test()

print("数据库更新完成")

—————————————————————————

>> 展示游戏数据到网页中（第九十五课）

- 能够调试和运行

- 能够将采集的游戏数据展示出来

import socket

import pymysql

def request\_handler(new\_client\_socket, ip\_port):

"""接收信息，并且做出响应"""

**# 7、接收客户端浏览器发送的请求协议**

request\_data = new\_client\_socket.recv(1024)

# print(request\_data)

**# 8、判断协议是否为空**

if not request\_data:

print("%s客户端已经下线!" % str(ip\_port))

new\_client\_socket.close()

return

**# 9、拼接响应的报文**

# 9.1 响应行

response\_line = "HTTP/1.1 200 OK\r\n"

# 9.2 响应头

response\_header = "Server:Python20WS/2.1\r\n"

response\_header = "Content-type:text/html;charset=utf-8\r\n"

# 9.3 响应空行

response\_blank = "\r\n"

# 9.4 响应的主体

response\_body = ""

conn=pymysql.connect(host='localhost',user='root', password='sxhmzz840845135', database='galgame')

cur = conn.cursor()

SQL = "select \* from v\_galgame order by 编号 desc;"

cur.execute(SQL)

result\_list = cur.fetchall()

for row in result\_list:

response\_body += "%s 游戏分区%s &nbsp 游戏名字%s &nbsp 下载链接:&nbsp<a href='%s'>%s</a><br>" %(row[0], row[1], row[2], row[3], row[3])

cur.close()

conn.close()

response\_data = response\_line + response\_header + response\_blank + response\_body

# 10、发送响应报文

new\_client\_socket.send(response\_data.encode())

# 11、关闭当前连接

new\_client\_socket.close()

def main():

"""主函数"""

# 1、导入模块

# 2、创建套接字

tcp\_server\_socket=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

# 3、设置地址重用

当前套接字 地址重用 值True tcp\_server\_socket.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR, True)

# 4、绑定端口

tcp\_server\_socket.bind(("", 8888))

# 5、设置监听，让套接字由主动变为被动接收

tcp\_server\_socket.listen(128)

# 6、接受客户端连接 定义函数 request\_handler()

while True:

new\_client\_socket, ip\_port = tcp\_server\_socket.accept()

print("新客户来了:", ip\_port)

# 调用功能函数处理请求并且响应

request\_handler(new\_client\_socket, ip\_port)

# 11、关闭操作

# tcp\_server\_socket.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

—————————————————————————

>> 闭包的概念及基本使用（第九十六课）

- 能够说出函数名赋值给一个变量的作用

- 能够说出闭包的特点

**1.闭包的概念:在一个外函数中定义了内函数,内函数运用了外函数的临时变量**

**并且外函数的返回值时内函数的引用，这样就构成了闭包**

**2.闭包构成的条件:**

**- 存在函数的嵌套关系**

**- 内层函数引用了外层函数的临时变量**

**- 外层函数返回内层函数的引用 ( 地址 )**

例子：

def test\_out(num):

print("我是 test\_out 中的值", num)

def test\_in(num\_in):

print("在 test\_in 中调用 test\_out 的值", num)

print("我是 test\_in 中的值", num\_in)

return test\_in

give\_me\_test\_in = test\_out(100)

give\_me\_test\_in(20)

——————————————————

我是 test\_out 中的值 100

在 test\_in 中调用 test\_out 的值 100

我是 test\_in 中的值 20

—————————————————————————

>> 闭包中变量问题（第九十七课）

def function\_out(num):

def function\_in():

# 造成报错的原因是,它认为内层函数已经定义了 num 变量,优先使用内层

# 如果已经定义且需要使用外层变量的话

nonlocal num

**# nonlocal 不使用内层函数的变量,而是使用函数的变量**

print("function\_in num", num)

# num = 88

num = 88

print(num)

return function\_in

ret = function\_out(10)

ret()

—————————————————————————

>> 装饰器入门（第九十八课）

- 能够说出装饰器用在什么场景下

- 能够说出装饰器函数相较于闭包的明显特点

- 能够说出装饰器的语法格式

**概念: 在不改变函数代码的情况下,给函数添加新的功能**

———————————————————————————————

#首先创建一个函数,并写一个( 传参数的位置 )

def function\_out(func):

#在函数的基础上定义另一个函数,并返回函数地址,形成闭包

# 内部函数可以去使用外部参数,或者自己去定义一个

def function\_in():

#错开去写

print("——————开始验证——————")

func()

#返回内部函数地址,此时 function\_out = function\_in

return function\_in

def login():

print("开始登陆")

"""

函数地址的重新定义, function\_out = function\_in

所以加上括号实则就是在调用 function\_in 内部的东西

function\_out(login)就是将 login 的地址传入了 func 里

注意function\_out = function\_in

也就是说现在是 function\_in地址(login地址)

login地址重用 = function\_in地址(login地址)

"""

login = function\_out(login)

login()

——————————————

def function\_out(func):

def function\_in():

print("开始验证")

func()

return function\_in

@function\_out

def login():

print("开始登陆")

login()

—————————————————————————

>> 装饰有参数的函数（第九十九课）

def function\_out(func):

def function\_in(num):

print("开始验证")

func(num)

return function\_in

@function\_out

def login(num):

print("开始登陆————", num)

# login = function\_out(login)

login(2)

————————————————————

def function\_out(func):

def function\_in(\*args,\*\*kwargs):

print("开始验证")

func(\*args,\*\*kwargs)

return function\_in

@function\_out

def login(\*args,\*\*kwargs):

print("开始登陆————", args)

print("开始登陆了————",kwargs)

# login = function\_out(login)

login(2,a=3)

注意 print 不能输出多值元组/字典

**return新认知:谁调用的它，它就把数值返回到哪并将控制权一并返回**

—————————————————————————

>> 主从配置（第一百课）

- 能够说出主从配置的应用场景

- 相当于备份服务器

BVItJ411v7nz 13.12

—————————————————————————

>> 在原装饰器上设置外部变量（第一百零一课）

def test(path):

print(path)

def function\_out(func):

def function\_in():

print("开始验证")

func()

return function\_in

return function\_out

@test("login.py")

#@test("login.py") 分为两部：

#1.test("login.py") --> function\_out的引用

#2.@第一步引用的结果

# n = test("login.py")

#@n

#和二为一

def login():

print("开始登陆")

login()

—————————————————————————

>> 多重装饰器（第一百零二课）

- 能够知道多个装饰器装饰一个函数的语法格式

**- 多重装饰器:给一个函数进行多次装饰**

**- 装饰原则:就近原则**

def makeBlod(func):

def function\_in():

return "<b>"+func()+"</b>"

return function\_in

def makeItalic(func):

def function\_in():

return "<i>"+func()+"</i>"

return function\_in

@makeItalic

@makeBlod

def test():

return "helloword-1"

print(test())

—————————————————————————

>> 类装饰器（第一百零三课）

- 了解类装饰器的使用

———————————— 基础

class Test(object):

def \_\_init\_\_(self):

print("---init---方法")

def run(self):

print("正在疯跑")

def \_\_call\_\_(self, \*args, \*\*kwargs):

print("---call---方法")

test = Test()

test.run()

test()

————————————

class Test(object):

def \_\_init\_\_(self,func):

print("---init---方法")

print(func)

self.func = func

def run(self):

print("正在疯跑")

def \_\_call\_\_(self, \*args, \*\*kwargs):

print("开始验证")

self.func()

@Test

#login = Test(login)

def login():

print("开始登陆")

login()

**个人理解:login()=test(),所以会调用 \_\_call\_\_()方法**

—————————————————————————

>> GIL锁的引入（第一百零四课）

- 能够使用 htop 查看系统状态

- 等同于 windows 下的任务管理器

- 能够通过观察不同的代码分析 CPU 占用情况

GIL全局解释锁:保证同一时间,只有一个线程使用CPU

—————————————————————————

>> GIL概锁解决方案（第一百零五课）

- 换解释器( pypy,Jpy ) - 不推荐

- 换进程( 会造成资源浪费 ) - 酌情而定

**- 编写 C 语言(绕过 GIL 锁,并且不用换解释器) - 推荐个der~啊？我还去学C？**

—————————————————————————

>> Python 中可变与不可变（第一百零六课）

- 这里的可不可变指的是内存地址是否变更

**变量 - 保存数字 - 不可变的**

**字典 - 可变**

**列表 - 可变**

—————————————————————————

>> 简单可变类型的拷贝（第一百零七课）

浅拷贝：**引用 ( 地址 ) 拷贝**:并**没有产生新的空间,**如果拷贝的是对象，原对象和 copy 对象，

都指向同一个内存空间，**只拷贝父对象，不会拷贝对象内部的子对象**

- 浅拷贝，产生新的空间，能够保证数据的独立性 <——可变类型

copy.copy( 变量 ) 会进行拷贝

深拷贝：**会产生新的空间**，如果拷贝的是对象，**原对象和 copy 对象都指向不同的内存空间，**

**会拷贝对象及其子对象( 会产生新的空间 )**

- 深拷贝，产生新的空间，能够保证数据的独立性 <——可变类型

copy.deepcopy( 变量名 )

作用:

1.减少内存的使用

2.以后在做数据的清洗，修改或者入库的时候，对原数据进行复制一份

**以防数据修改之后，找不到原数据**

—————————————————————————

>> 复杂可变类型的拷贝（第一百零八课）

- 知道区别

**- 浅拷贝只复制最顶层的数据子 ( 对象只是作为引用 )**

**- 深拷贝是开辟新的空间并复制**

—————————————————————————

>> 简单不可变类型拷贝（第一百零九课）

**- 知道简单不可变类型都是浅拷贝**

**不可变类型:不管深拷贝还是浅拷贝，都不会开辟新的空间**

**而是直接引用了被拷贝的数据的地址**

等价于 tuple2 = tuple1

—————————————————————————

>> 复杂不可变类型拷贝（第一百一十课）

**浅拷贝：和上一节课一个效果**

**深拷贝：创建了一个新的空间**

浅拷贝：只关心最外的数据类型是什么，如果是不可变的数据类型，直接引用

没有办法保证数据的独立性

深拷贝：这个数据是否有可变的数据类型，如果有，他就会开辟多个空间存储数据和地址，达到保证数据独立性的作用

—————————————————————————

>> 切片拷贝，字典拷贝（第一百一十一课）

- 知道什么是切片拷贝

- 知道什么是字典拷贝

**切片拷贝：是一种浅拷贝，副本对象和原对象指向同一个空间**

**变量名[ 从什么地方开始 : 到什么地方结束 ]**

list1 = [1,2,3,4,5,6]

list2 = list1[0:2]

print(list2)

**字典拷贝：也是一种浅拷贝，副本对象和原对象指向同一个空间**

—————————————————————————

>> import 导入模块路径问题（第一百一十二课）

- 知道如何导入指定目录下的模块

**找不到桌面上 module 模块的原因是 Py 没去找**

import sys

print(sys.path)

sys.path.append("路径")

现在就可以用了

—————————————————————————

>> import 的 reload 加载问题（第一百一十三课）

from imp import reload 导入系统模块

reload( 模块名 ) 可以重新加载模块

—————————————————————————

>> from...import 的私有化问题（第一百一十四课）

- 知道 \_变量名 使用 from xxx import \* 不能被导入

name = "莫妮卡"

age = 18

此时在 from...import \* 的情况下都可以被访问

name = "莫妮卡"

\_age = 18

此时在 from...import \* 的情况下 age/\_age 访问不到

类中定义私有化属性

self.\_\_name = xxx

—————————————————————————

>> import，from...import 区别（第一百一十五课）

import 是引用

from...import 是拷贝

—————————————————————————

>> 可变参数的拆包问题（第一百一十六课）

**拆包：对于函数中多个返回数据，去掉元组，列表或者字典，直接获取里面数据的过程**

**可变参数 \*args \*\*kwargs 默认会封包过程**

如果想要这种单数继续传递到下一个函数

**传递的时候 func( \*args,\*\*kwargs )**

—————————————————————————

>> 单继承中super( )（第一百一十七课）

**- super( ) 使用的时候，传递参数的时候，self 不用传递**

**- super( ) 调用顺序，按照 mro 顺序来完成**

类名.\_\_mro\_\_是个元组

当在类中使用 super( ) 在 mro 列表中找到当前类的下一个元素，调用该元素的方法

—————————————————————————

>> 多继承和 MRO 顺序（第一百一十八课）

- 知道多继承的特性

- 能够说出 MRO 顺序的作用

- 能够说出 super 的作用

**多继承中 super() 执行顺序，严格执行 MRO 顺序表**

MRO顺序表

- 类名.mro( )

- 类名.\_\_mro\_\_

注意：

当在类中使用 super( ) 在 MRO 列表中找到当前类的下一个元素，调用该元素的方法

多继承中，不建议使用类名直接调用父类的方法

—————————————————————————

>> @properpy 装饰方法（第一百一十九课）

- 能够说出使用 properpy 的好处

- 只能有一个 self 参数

- 调用时无需使用 ( )

class Foo:

def \_\_init\_\_(self):

self.num = 100

@property

def prop(self):

return self.num

f = Foo()

#print(f.prop()) --> print(f.prop)

print(f.prop)

—————————————————————————

>> limit 分页算法（第一百二十课）

class Pager():

def \_\_init\_\_(self, current\_page):

self.current\_page = current\_page

self.per\_items = 10

@property

def start(self):

val = (self.current\_page - 1) \* self.per\_items+1

return val

@property

def end(self):

val = self.current\_page \* self.per\_items

return val

p = Pager(2)

print(p.start)

print(p.end)

—————————————————————————

>> property 其他使用方式（第一百二十一课）

- 知道 @xxx.setter 的作用

- 知道 @xxx.delete 的作用

**- Python 中的类具有 经典类 和 新式类 ，新式类的属性比经典类的属性丰富**

( 如果类继承object，那么该类是新式类，**Python3中默认所有类为新式类** )

class Goods(object):

def \_\_init\_\_(self):

self.org\_price = 1000

self.discount = 0.7

@property

def price(self):

return self.org\_price \* self.discount

@price.setter

def price(self, val):

print("调用",val)

if val > 0:

self.org\_price = val

@price.deleter

def price(self):

print("调用123")

goods = Goods()

print(goods.price)

# goods.price == goods.price( 500 )

goods.price = 500

print(goods.price)

# del goods.price == @price.deleter 装饰的方法

del goods.price

—————————————————————————

>> property 其他使用方式 - 总结

@property goods.price

@xxx.setter goods.price = xxx

@xxx.deleter del goods.price == @price.deleter 装饰的方法

—————————————————————————

>> property 作为类属性（第一百二十二课）

- 类属性，在类中定义值为 property 对象的类属性

class Foo(object):

def get\_bar(self):

return "monika"

BAR = property(get\_bar)

foo = Foo()

print(foo.BAR)

property对象有四个参数

- property( 一，二，三，四 )

- 1.当我们通过 foo.BAR 时，自动调用第一个参数的方法

- 2.当我们通过 foo.BAR = 100 时，自动调用第二个参数的方法

- 3.当我们通过 del foo.BAR 时，自动调用第三个参数的方法

- 4.当我们通过 Foo.BAR.\_\_doc\_\_ 时，自动获取第四个参数的内容 ( 注意是类名 )

class Goods(object):

def \_\_init\_\_(self):

self.org\_price = 1000

self.discount = 0.7

def get\_price(self):

print("get\_price 方法")

return self.org\_price \* self.discount

def set\_price(self, val):

print("set\_price方法", val)

if val > 0:

self.org\_price = val

print("-"\*50)

def del\_price(self):

print("del\_price方法")

print("-" \* 50)

BAR = property(get\_price, set\_price, del\_price, "Monika is very Beautiful")

goods = Goods()

print(goods.BAR)

goods.BAR = 500

print(goods.BAR)

del goods.BAR

print(Goods.BAR.\_\_doc\_\_)

—————————————————————————

>> 魔法属性和方法 - 1（第一百二十三课）

- 知道 \_\_doc\_\_ 魔术方法的作用

- 知道 \_\_module\_\_ , \_\_class\_\_ 魔术方法的作用

- 知道 \_\_init\_\_ 魔术方法的作用

- 知道 \_\_del\_\_ 魔术方法的作用

1.\_\_doc\_\_ 表示类的描述信息

类名.\_\_doc\_\_

对象.test.\_\_doc\_\_

例子：

class Goods(object):

"""这是一个商品类"""

def monika(self):

"""这是一只莫妮卡玩偶"""

goods = Goods()

print(Goods/goods.\_\_doc\_\_)

print(goods/Goods.monika.\_\_doc\_\_)

2. \_\_module\_\_ 和 \_\_class\_\_

- \_\_module\_\_ ：表示当前操作的对象在哪个模块

- print( goods.\_\_module\_\_ )

- \_\_class\_\_ ：表示当前操作对象的类是什么

- print( goods.\_\_class\_\_ )

—————————————————————————

>> 魔法属性和方法 - 2（第一百二十四课）

- 知道 \_\_call\_\_ 魔术方法的作用

- 知道 \_\_dict\_\_ 魔术属性的作用

- 知道 \_\_str\_\_ 魔术方法的作用

- 知道 \_\_getitem\_\_ , \_\_setitem\_\_ , \_\_delitem\_\_ 魔术方法的作用

1.\_\_call\_\_ : 对象后面加括号，触发执行

class Goods(object):

def \_\_call\_\_(self, \*args, \*\*kwargs):

print("call方法调用")

goods = Goods()

goods()

2.\_\_str\_\_ : 打印对象，触发执行

class Goods(object):

"""这是一个商品类"""

def \_\_str\_\_(self):

return "I LOVE MONIKA"

goods = Goods()

print(goods)

3.\_\_dict\_\_ : 类或对象中的所有属性，返回字典

class Goods(object):

def \_\_init\_\_(self):

self.org\_price = 100

self.discount = 0.7

goods = Goods()

print(goods.\_\_dict\_\_)

输出：{'org\_price': 100, 'discount': 0.7}

print(Goods.\_\_dict\_\_)

输出类的所有信息

4. \_\_getitem\_\_ 和 \_\_setitem\_\_ 与 \_\_delitem\_\_

- 用于索引操作，如字典，以上分别表示获取，设置，删除数据

- 1.getitem

class Goods(object):

def \_\_getitem\_\_(self, item):

print("key =",item )

goods = Goods()

goods['a'] 调用 \_\_getitem\_\_ 方法

输出：key = a

2.setitem

class Goods(object):

def \_\_setitem\_\_(self, key, value):

print("key = %s value = %s" %( key,value ) )

goods = Goods()

goods['a'] = 10 调用 \_\_setitem\_\_ 方法

输出：key = a value = 10

3.delitem

class Goods(object):

def \_\_delitem\_\_(self, key):

print("要删除key = ",key)

goods = Goods()

del goods['a'] 调用 \_\_delitem\_\_ 方法

输出：要删除key = a

—————————————————————————

>> with 管理上下文方式 一（第一百二十五课）

- 能够说出 with 的好处

- 能够自定义符合 with 规范的上下文管理类

**类里必须有两个方法 \_\_enter\_\_ 与 \_\_exit\_\_**

class Myfile(object):

def \_\_enter\_\_(self):

self.file = open(self.file\_name, self.file\_modle)

return self.file

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

self.file.close()

def \_\_init\_\_(self,file\_name,file\_modle):

self.file\_name = file\_name

self.file\_modle = file\_modle

with Myfile("C:/Users/Lenovo/Desktop/JustMonika.txt","r") as file:

file\_data = file.read()

print(file\_data)

—————————————————————————

>> with 管理上下文方式 二（第一百二十六课）

- 能够使用装饰器实现 with 上下文管理器

- 导入 from contextlib import contextmanager

@contextmanager

def myopen(file\_name,file\_model):

print("进入上文")

file = open(file\_name, file\_model)

yield file

print("进入下文")

file.close()

with myopen("C:/Users/Lenovo/Desktop/JustMonika.txt","r") as file:

file\_data = file.read()

print(file\_data)

—————————————————————————

>> miniWeb框架构建基本构建（第一百二十七课）

def application(current\_dir, request\_data, ip\_port):

# 调用 parse\_request函数，解析请求协议，返回请求的资源路径

file\_path = parse\_request(request\_data, ip\_port)

# 定义变量保存资源路径

resource\_path = current\_dir + file\_path

response\_data = None

#改进动态显示

if file\_path.endswith(".py"):

if file\_path == "/index.py":

response\_body = "This Monika time &nbsp @/index.py %s&nbsp&nbsp" %time.ctime()

response\_data = utils.creat\_http\_response("200 OK", response\_body.encode())

elif file\_path == "/center.py":

response\_body = "This Monika time &nbsp @/center.py %s&nbsp&nbsp" %time.ctime()

response\_data = utils.creat\_http\_response("200 OK", response\_body.encode())

elif file\_path == "/gettime.py":

response\_body = "This Monika time &nbsp @/gettime.py %s&nbsp&nbsp" %time.ctime()

response\_data = utils.creat\_http\_response("200 OK", response\_body.encode())

else:

response\_body = "<h1>Fuck You!</h1>"

response\_data = utils.creat\_http\_response("404 Not Found", response\_body.encode())

else:

try:

# 通过 with open 读取文件

with open(resource\_path, "rb") as file:

# 把读取的文件内容返回给客户端

response\_body = file.read()

# 调用 utils 模块的 create\_http\_response 函数，拼接响应协议

response\_data = utils.creat\_http\_response("200 OK", response\_body)

except Exception as e:

# 2）响应的内容为错误

response\_body = "Error! (%s)" % str(e)

# 3）把内容转换为字节码

response\_body = response\_body.encode()

response\_data = utils.creat\_http\_response("404 Not Found!", response\_body)

return response\_data

—————————————————————————

>> 路由列表 ( django )（第一百二十八课）

创建两个文件，对应网页链接去查找函数

——————————————————

urls:

from application import funs

字典:

{

'/index.py':index,

'/center.py':center

'/gettime.py':gettime

}

——————————————————

funs.py

"""

定义若干函数实现功能

1.index

2.center

3.gettime

"""

——————————————————

app2.py

#改进动态显示

if file\_path.endswith(".py"):

if file\_path in urls.route\_dict:

func = urls.route\_dict[file\_path]

response\_body = func()

response\_data = utils.creat\_http\_response("200 OK", response\_body.encode())

else:

response\_body = "<h1>Fuck You!</h1>"

response\_data = utils.creat\_http\_response("404 Not Found", response\_body.encode())

—————————————————————————

>> 装饰器路由 ( flask )（第一百二十九课）

- 能够使用装饰器路由自动添加路径数据

funs:

from application import urls

def route(path):

def function\_out(func):

urls.route\_dict[path] = func

# print("ok by", path)

def function\_in():

return func()

return function\_in

return function\_out

@route("/index.py")

#route("/index.py") ----> function\_out 引用

#@function\_out

#index = function\_out( index )

#index() ----> function\_in

def index():

"""处理 index.py 请求的信息"""

return "This is index show"

@route("/center.py")

def center():

"""处理 center.py 请求的信息"""

return "This is center show"

@route("/gettime.py")

def gettime():

"""处理 gettime.py 请求的信息"""

return "This is gettime show"

————————————————————

urls:

route\_dict = {

}

—————————————————————————

>> 模板替换（第一百三十课）

- 能够说出模板替换的具体步骤

funs:

import re

from application import urls

def route(path):

def function\_out(func):

urls.route\_dict[path] = func

# print("ok by", path)

def function\_in():

return func()

return function\_in

return function\_out

@route("/index.py")

#route("/index.py") ----> function\_out 引用

#@function\_out

#index = function\_out( index )

#index() ----> function\_in

def index():

"""处理 index.py 请求的信息"""

with open("templates/index.html", encoding="UTF-8") as file:

content = file.read()

content = re.sub("{%content%}","Hello World", content)

return content

@route("/center.py")

def center():

"""处理 center.py 请求的信息"""

with open("templates/center.html", encoding="UTF-8") as file:

content = file.read()

content = re.sub("{%content%}", "Hello World", content)

return content

@route("/gettime.py")

def gettime():

"""处理 gettime.py 请求的信息"""

return "This is gettime show"

—————————————————————————

>> 首页数据加载（第一百三十一课）

funs:

import re

from application import urls

import pymysql

def route(path):

def function\_out(func):

urls.route\_dict[path] = func

# print("ok by", path)

def function\_in():

return func()

return function\_in

return function\_out

@route("/index.py")

#route("/index.py") ----> function\_out 引用

#@function\_out

#index = function\_out( index )

#index() ----> function\_in

def index():

"""处理 index.py 请求的信息"""

with open("templates/index.html", encoding="UTF-8") as file:

content = file.read()

conn = pymysql.connect(host= "localhost", user="root", password="sxhmzz840845135", database="stock\_db")

cur = conn.cursor()

cur.execute("select \* from info")

data\_from\_mysql = str(cur.fetchall())

cur.close()

conn.close()

content = re.sub("{%content%}", data\_from\_mysql, content)

return content

@route("/center.py")

def center():

"""处理 center.py 请求的信息"""

with open("templates/center.html", encoding="UTF-8") as file:

content = file.read()

content = re.sub("{%content%}", "Hello World", content)

return content

@route("/gettime.py")

def gettime():

"""处理 gettime.py 请求的信息"""

return "This is gettime show"

—————————————————————————

>> 首页数据加载 - 2（第一百三十二课）

funs:

import re

from application import urls

import pymysql

def route(path):

def function\_out(func):

urls.route\_dict[path] = func

# print("ok by", path)

def function\_in():

return func()

return function\_in

return function\_out

@route("/index.py")

#route("/index.py") ----> function\_out 引用

#@function\_out

#index = function\_out( index )

#index() ----> function\_in

def index():

"""处理 index.py 请求的信息"""

with open("templates/index.html", encoding="UTF-8") as file:

content = file.read()

conn = pymysql.connect(host= "localhost", user="root", password="sxhmzz840845135", database="stock\_db")

cur = conn.cursor()

cur.execute("select \* from info")

# data\_from\_mysql = str(cur.fetchall())

data\_from\_mysql = ""

for line in cur.fetchall():

str = """

<tr>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td><input type="button" value="添加" id="toAdd" name="toAdd" systemidvaule="000007"></td>

</tr>

""" % line

data\_from\_mysql += str

cur.close()

conn.close()

content = re.sub("{%content%}", data\_from\_mysql, content)

return content

@route("/center.py")

def center():

"""处理 center.py 请求的信息"""

with open("templates/center.html", encoding="UTF-8") as file:

content = file.read()

content = re.sub("{%content%}", "Hello World", content)

return content

@route("/gettime.py")

def gettime():

"""处理 gettime.py 请求的信息"""

return "This is gettime show"

—————————————————————————

>> 首页数据加载 - 3（第一百三十三课）

funs:

conn = pymysql.connect(host="localhost", database="stock\_db", user="root", password="sxhmzz840845135")

# 3) 创建游标对象

cur = conn.cursor()

# 4）通过游标执行查询

cur.execute("select \* from info")

# 5) 获取查询的结果

# ((,,,,,,),(),())

# data\_from\_myql = str(cur.fetchall())

data\_from\_mysql = ""

# 5.1 遍历元组（得到每一行信息）

# 5.1 拼接html格式的字符串

for line in cur.fetchall():

str = """

<tr>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td>%s</td>

<td><input type="button" value="添加" id="toAdd" name="toAdd" systemidvaule="000007"></td>

</tr>

""" % line

data\_from\_mysql += str

# 6) 关闭操作

# 游标 连接

cur.close()

conn.close()

—————————————————————————

>> END?（第一百三十四课）