## 1linux-介绍-命令

### 1今日目标

每天一个小目标

### 1.2操作系统介绍

操作系统的作用：

对上提供软件运行支持，对下控制硬件的运行。

### 1.3操作系统分类

操作系统的分类：

桌面（windows）,服务器（Linux）,嵌入式（Linux）,移动设备操作系统（ios|android）

虚拟机使用软件模拟的与原系统彻底隔开的系统

### 1.4Linux系统的发展史

1,linux基于unix发展而来

2,linux内核：系统的心脏，包含：驱动，文件系统，进程系统

3;linux发行版：内核 + 一套软件 +手册

常见的发行版 centos redhit ubuntu

### 1.5Linux系统应用领域

linux主要在服务器领域

主要领域：服务器，嵌入式，个人桌面

### 1.6Ubuntu图形界面入门

### 1.7文件与目录

Linux 与windows文件存储方式不同，windows有多个根目录c:d:，Linux只有一个根目录/

Linux根目录： 文件--》其他--》根目录

根目录里常见的目录介绍：

/bin 二进制文件

/home 用户目录

/home/xx 用户家目录

/etc 系统配置目录

/root 超级管理员目录

### 1.8 Linux终端命令格式

终端是什么？

终端：软件的控制台，在终端中输入指令可以快速操作计算机

终端指令的格式

格式：命令 -[选项] [参数]

查看帮助

1,man 命令

回车 一行

空格 一页

F 下一屏

B 上一屏

q 退出

2,ls --help

### 1.9【重点】常用命令--显示文件和目录

·Pwd 显示当前所在的路径

·tree 以树状列表显示文件夹的内容

1. tree 当前目录
2. tree aaa 指定目录

·ls 以列表的形式显示文件夹的内容

1. ls -a 显示所有文件（包含隐藏文件）

隐藏文件在linux中以“.”开头

### 1.10【重点】常用命令--显示详细信息

·ls的两个选项

-l 以详细信息显示列表内容

-h 以更加人性化的方式显示文件大小（文件大小：k,mb）

### 1.11【重点】常用命令--切换目录

·cd指令，切换工作目录

·cd 路径 切换指定目录

·cd 回到家目录

·cd ~ 回到家目录

·cd .. 返回上一级

·cd - 返回上次的目录

### 1.12【重点】常用命令--创建文件和目录

·mkdir 创建目录 递归创建 添加-p选项

·touch 创建一个文件touch 文件名 多个文件touch 文件1 文件2 文件3

·gedit 用来打开一个文件进行编辑

1. 打开文件后，终端进入等待状态
2. 可以同时编辑多个文件

### 1.13【重点】常用命令--删除文件和目录2

·rm -> remove

删除文件或者目录

删除文件：rm 文件名

-i 以交互模式删除

-f 强制删除不提示

删除文件夹:rm -r 文件夹名， 递归删除目录中的内容

### 1.14【重点】常用命令--文件拷贝2

·cp -> copy

用来拷贝文件或者目录

拷贝文件:cp 源路径 目标路径

-i 交互模式拷贝

-f 强制覆盖不提示

-v 显示拷贝的过程

-a 拷贝原有属性

k拷贝目录：cp -r 源路径 目标路径

### 1.15【重点】常用命令--移动

·mv -> move

1,移动文件或者文件夹

mv 源路径 目标路径

-i 交互方式进行文件的移动

-f 强制覆盖不提死

-v 显示移动过程

注意：-----移动文件夹 不需要加-r

2,重命名文件或者文件夹

重命名:在一个目录中进行移动才能进行重命名

mv 旧文件名 新文件名

### 1.16常用命令--其他

·清屏 clear Ctrl + l

·which 查看某个指令所在的位置

·Tab: 自动补全

1. 没有歧义，自动补全
2. 有歧义（多个文件）按2次tab可以查看候选的文件名

·终止指令的执行 ctrl + c

### 1.17常用命令--日历日期

·cal 查看日历

-3 上月，本月，下月

-y 显示一年的日历

-j 以一年中的第XX天的格式来显示日历

·data 查看当前的年月日时分秒

格式化显示

data”+%Y” 输出年份

%m 月份

%d 天数

%H 小时

%M 分

%S 秒

%F 等价于 %Y-%m-%d

%T 等价于 %H:%M:%S

### 1.18常用命令--历史指令

·history 用来查看历史指令的信息

1. history 显示所有历史指令
2. history 数量 显示一定数量的历史指令
3. ！编号 执行某个历史指令

·历史指令保存在那 ~/.bash\_histroy文件中

### 1.19【重点】常用命令--文件查看cat

·cat 查看或者合并文件内容

1，查看文件

cat 文件名

-n 查看文件的时候，对每一行进行编号

-b 非空行进行编号

-s 连续2行以上的空行，只显示1行

1. 连接文件 把两个文件连接到一起输出

Cat 文件名1 文件名2

### 1.20【重点】常用命令--文件查看more

·more 用来分屏（分页）查看文件内容

选项：

+num 从第num行开始查看文件

-p 先清屏在查看文件

-s 连续2行以上内容 只显示一行

快捷键：

回车 查看一行

空格 查看下一屛

Ctrl +f/f 下一屏

Ctrl +b/b 上一屏

退出 q

### 1.21知识点总结

11

## 2linux--常见指令

### 2.1今日目标

### 2.2常用命令-数据流，管道

·> - 数据流

数据路分为输入流和输出流

输入流：从键盘或者文件中读取内容到内存中 input

输出流：从计算机内存中吧数据写进文件中或者显示到显示器上output

linux有三种流

标准的输入流 stdin

标准的输出流 stderr

标准的错误输出流 stdout

重定向：改变数据的流向（一般重定向到文件中）

·| -管道命令：一个命令的输出作为另一个的输入

指令1 | 指令2 \*指令1必须有输出

### 2.3【难点】命令建立连接

Linux连接文件有两种

·软连接: 相当于快捷方式 通过软连接可以修改文件内容

ln -s 源文件 连接文件

·硬链接： 一个文件有多个名字 通过硬连接可以修改文件内容

ln 源文件 连接文件

### 2.4【难点】常用命令建立连接2 （删除）

·删除软硬连接产看对源文件的影响

删除软硬连接对源文件没有影响

·删除源文件查看对软硬连接的影响

删除源文件软连接不可用

删除源文件，如果这个有多个硬链接，无影响

·区别：

·软连接介意指向一个不存在的文件，硬链接不可以

·可以对目录创建一个软连接，不可以对目录创建硬链接

### 2.5常用命令文件搜索

·在文本内部搜索grep

·grep内部搜索

grep “内容” 文件路径

-n 查看结果行数

-i 忽略大小写

-v 结果取反

·grep正则搜索

grep “^a” 文件路径 搜索以a开头的行

·在计算机中搜索find

Find 目标的目录 选项 条件

·按名称搜索 -name

find ./ -name text.txt

find ./ -name “\*.txt”

\* 任意一个或者多个字符

? 任意一个字符

[] 范围[12]

·按大小搜索 -size

Find ./ -size +30M

Find ./ -size -15M

Find ./ -size+15M -size-30M

### 2.6常用命令-归档与压缩（1）

·归档和解档

·tar -cvf 归档的文件名.tar 文件1 文件2

·tar -xvf 档案文件名

f选项必须放在最后

·归档+压缩 和 解压+解档

·tar -zcvf 归档的文件名.tar.gz 文件1 文件2

·解压解档

·tar -zxvf 归档的文件名.tar.gz

如果要指定解档的目录

tar -zxvf 归档的文件名.tar.gz. -C要解压档的目录

### 2.7常用命令归档压缩（2）

zip 和unzip 文件压缩加压

·zip 压缩

压缩目录：zip -r XXX.zip 目录

·unzip解压缩

Unzip XXX.zip

### 2.8文件的权限

·文件权限的构成

9个字母 分成三组（拥有者权限u，组权限g，其他用户权限o所有用户权限a

R 可读 w 可写 x可执行 （文件：文件可以直接运行,绿色，目录：表示这个目录可以打开） -没有权限

### 2.9权限的修改

·权限修改：chmod

1字母法

用户 ： u g o a

权限设置： + (增加) -（撤销） =（设置）

具体权限：r w x -

用法：chomd 用户 +具体权限 文件名

2数字法

r - 4 w -2 x -1 - -0

chomd 644 文件名

三位权限数字 拥有者权限 组权限 其他人权限‘

### 2.10常用命令 用户管理

·切换用户

临时 sudu 命令

永久 1 sudu -s 输入当前账户密码

2 su 用户名 输入用户名对应的密码

·passwd

Passwd 表示修改当前用户密码

Passwd XXX 表示修改xxx用户的密码

·exit

· exit 如果没有用户在栈中 直接退出终端

·如果多次 切换用户 退出到上一次登陆的用户

·who

·用来查看当前系统登陆了那字儿用户

-q 统计用户数

-u 显示最后一次操作时间

### 2.11 常用命令 关机重启

·shutdown

Shutdown 18.55 指定18.55关机

Shutdown +20 20分钟以后关机

Shutdown -h now 立刻关机

·重启reboot

Shutdown -r now 重新启动操作系统

### 2.12 ubuntu软件安装与卸载

·linux软件安装的三种方式

原代码包安装 deb包安装 apt-get方式

·apt-get方式安装软件

·配置软件源 修改 /ect/apt/source.lis

·更新软件源 sudo apt-get update

·安装软件 sudo apt-get inatall 软件包名称

·卸载软件 sudo apt-get-remove 软件包名称

### 2.13【重点】ssh远程登录

·服务器端安装ssh server

·客户端登陆

Ssh 服务器用户名@服务器地址

Ssh [demo@192.168.150.112](mailto:demo@192.168.150.112) --> 123

### 2.14【重点】远程拷贝

·scp的作用：可以上传或者下载文件

上传：

Scp 本地路径 服务器用户名@服务器ip:服务器路径

下载：

Scp 服务器用户名@服务器IP：服务器路径 本地路径

·如果下载目录

Scp -r

### 2.15【重点】编辑器vim介绍

·三种模式

命令模式（移动光标，复制删除），

输入模式（编辑文件），

末行模式（保存文件，查找替换）

2三行模式转换：

打开文件默认是命令模式 ----> a\i\o------>输入模式---->esc----->命令模式----->：----->末行模式

### 2.16【重点】编辑器Vim操作

·创建文件 vim 文件名 ------>i进入编辑模式-->编辑文件----->ESC到命令模式----->：进入末行模式------->wq保存并退出

·Vim进入输入模式

·i 光标前插入 I 行首插入

·a 光标后插入 A行尾插入

·o 光标下一行产生新一行 O光标上一行产生新一行

·进入命令模式

任何模式下按esc

### 2.17实践系统性能定式监控-介绍

Psutil

### 2.18实践系统性能定式监控-基础

### 2.19系统性能定时监控-升级版

## 3python开发环境及网络基础

### 3.1yagmail发送邮件

### 3.2实战系统性能定式监控 --邮件监控

·思路：三步1导入模块 import yagmail

2创建对象 yagmail .SMTP(user=”发件人”,password=”邮箱授权码”,host =”发件服务器”)

3发送邮件 obj.send(“收件人”,”主题”,”内容”)

### 3.3虚拟环境

·Python虚拟环境，允许安装不同版本的套件

·创建： mkvirtualenv 虚拟环境名称 默认python2.7

·进入：workon 虚拟环境名称

·查看所有环境：workon + 回车

·删除：1）先退出虚拟环境2）rmvirtualenv 虚拟环境名称

·退出:deactivate

·创建虚拟环境，指定python路径：mkvirtualenv-p /user/bin/python3.6 XXX

·在虚拟环境安装套件： pip install 套件名 ==版本

### 3.4网络通信概述

·什么是网络？

一些以共享资源为目的的计算机的集合

·学习网络的目的？

能够进行网络编程

网络编程：通过编写程序实现计算机之间内进行数据的传递

### 3.5【重点】ip地址

IP地址：用来在网络中标机一台电脑，是网络设备为网络中的每台计算机分配的一个唯一标识

·ipv4： 点分十进制 xxx.xxx.xxx.xxx 每一段0-255

·IP地址分类：

常用的是C类地址

·私有IP地址 172.16.xxx.xxx 192.168.xxx.xx

·本机地址：127.0.0.1

·ipv6：冒号分16进制

### 3.6IP地址查看

·虚拟机上网方式

·NAT 虚拟机与物理主机公用网络

·桥接模式 虚拟机能够获取局域网的IP地址

·ifconfig Linux查看IP地址

Config windows查看网络地址

·ping 检查某某个主机是否建立连接

### 3.7【重点】端口

·端口 可以认为是设备与外界通讯的一个出口

·分类：知名端口和动态端口

·知名端口：0-1023 固定不变 用户不能用

21 FTP服务

1. ssh服务
2. HTTP服务

·动态端口：1024-65535 程序可以绑定

·端口查看

·查看所有 ：netstat -an

·搜索某一个端口:netstat -an |grep:22

·查看某个端口，是哪个程序使用

### 3.8【重点】传输方式

·面向无连接

两台计算机通信时不需要建立连接（逻辑），就可以进行数据的收发，数据可能会丢

传输协议：Udp

·面向有连接

两台计算机通信时，需要先建立连接，再能通信

传输协议：rcP

### 3.9【重点】socket简介

·网络通信的基本单元，提供的方法可以实现数据的发送和接收

·创建套接字

1导入模块

Import socket

2创建套接字

#socket.socket(协议类型传输方式)

#参数一：

#socket.AF\_INIT 使用ipv4

#socket.AP\_STPMAM 使用ipv6

#参数二：

#socket.SOCK\_DGRAM 使用upd的传输方式

#socket.SOCK\_STREAM 使用tcp的传输方式

#upd\_socket = socket.socket(socket.AF\_INIT,socket.SOCK\_DGRAM)

3数据传输

4关闭套接字

Upd\_socket.close()

### 3.10【重点】udp网络程序--发送数据

udp不严格区分可会断和服务端

实战--发送数据

### 3.11【重点】udp网络程序--发送接收数据

实战--接受数据

### 3.12python3解码编码

·编码

字符串.encode() 默认utf-8字符集

·解码

字符串.decde() 默认utf-8字符集

·解码失败的处理

Decode(encoding=”字符集”,errors=”错误处理方式”)

错误处理方式：

1ignore 忽略模式

2strict 严格模式 报错 默认

### 3.13udp端口绑定----发送端

实战--端口绑定

核心用法udp\_socket.bind((“192.168.0.1”,8888))

#---参数是一个元组（“192.168.0.1”,8888）

### 3.14udp端口绑定--接收端

实战--端口绑定接收端

Udp\_socket.bind(“”,12345)

Ip地址净可能给默认值 好处如果当计算机有多个网卡时，不同网卡的数据都能被接收

发送端可以不绑定端口，接收端必须绑定端口

### 3.15udp广播

广播地址：xxx.xxx.xxx.255 或者255.255.255.255

思路：

#1导入模块

import socket

#2创建套接字

#                           IPV4               UDP

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

#3设置广播权限

            #--套接字默认不允许发送广播

            #--需要用setsockopt(套接字，属性，属性值)

            #--socket.SOL\_SOCKET-----当前套接字

            #---socket.SO\_BROADCAST-----发送广播

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

#4发送数据

udp\_socket.sendto("服务器即将关闭".encode("gbk"),("192.168.3.255",8080))

#5关闭套接字

udp\_socket.close()

### 3.16【难点】案例：udp聊天器

·send\_msg 发送信息

·recv\_msg 接受信息

·main() 主函数

### 3.17【难点】案例：udp聊天器二

·发送信息

实战-----聊天器二.py

## 4网络--udp/tcp

### 4.1今日目标

### 4.2【重点】TCP简介

tcp通信严格区分客户端和服务端

面向连接的可靠地基于字节的传输协议

tcp一面向连接：

通信双方需得进行一个通信连接双方都需要给这个连接分配必要的系统资源

Tcp二可靠传输：

1tcp采取应答机制

发送的每个报文段必须得到接收方的应答才认为tcp报文段传输成功

2超时重传

发送端发出一个报文段后就启动定时器 如果在规定时间内没有收到应答，

那么就会重新发送这个报文段

3错误校验···（顺序校验 去重）

TCP用一个校验函数来检验数据是否存在错误，在发送和接受时都要计算校验和

4流量控制和阻塞管理

流量控制用来避免主机发送过快而使接收方来不及完全收下。

3 tcp和udp的不同点

·面向连接

·有序数据传输

·重发对的数据包

·舍弃重复的数据包

·无差错的数据传输

·阻塞/流量控制

### 4.3【重点】tcp网络程序--客户端

·实现步骤：

1导入模块

2创建套接字

3建立连接

tcp\_client\_socket.connect(("服务器地址",端口号))

4发送数据

tcp\_client\_socket.send("约吗？".encode("gbk"))

5接收数据（recv\_data 是接收数据的二进制）

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

5关闭连接

### 4.4【重点】tcp网络程序--服务端

·1导入模块

2创建套接字

3绑定端口

4开启监听（把套接字由主动设置为被动模式）

tcp\_server\_socket.listen(128) 最大允许128个连接

5等待客户端连接

#-----开始接收客户端连接 程序会进入阻塞状态 如果有客户端连接  自动解除阻塞 往下执行

#recv\_data有两部分

#               第一部分返回一个新的套接字

#                第二部分客户端的ip地址和端口号

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

这个new\_client\_socket 是一个新的套接字 只是服务当前的客户端

6使用新的套接字接收客户端发的信息

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

7关闭新的套接字

new\_client\_socket.close()

8关闭服务器

### 4.5【重点】tcp客户端加强版

·可以循环接收多条信息

while True:

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

        if recv\_data:

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

            print("接受到[%s]的信息:%s" % (str(client\_ip\_port),recv\_text))

        else:

            print("客户端已经断开连接")

            break

·循环接收多个客户端连接

while True:

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

    print("新客户端上线了:%s" % str(client\_ip\_port))

    #6收发数据

    #----recv会让程序进一步阻塞   ，收到信息后解除阻塞

    while True:

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

        if recv\_data:

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

            print("接受到[%s]的信息:%s" % (str(client\_ip\_port),recv\_text))

        else:

            print("客户端已经断开连接")

            break

    #--------------这句话关闭表示不能再和当前的客户端通信

    new\_client\_socket.close()

#7关闭连接

#tcp\_server\_socket.close()表示程序不再接受新的客户端连接

tcp\_server\_socket.close()

### 4.6【重难点】tcp文件下载器客户端(1)

#1导入模块

#2创建套接字

#3建立连接

#4接受用户输入的文件名

#5发送文件名到服务端

#6创建文件，并且准备保存

#7接受服务器的数据，保存到本地（循环）

#8关闭套接字

### 4.7【重难点】tcp文件下载器服务端(2)

#1导入模块

#2创建套接字

#3绑定端口

#4设置监听，设置套接字由主动到被动

#5接受客户端连接

#6接受客户端发送的文件名

#7根据文件名读取文件内容

#8把读取的内容发送给客户端（循环）

#9关闭和当前客户的连接

#10，关闭服务器

### 4.8tcp的三次握手

名词: 标志位：SYN :连接请求

ACK：确认

FIN:关闭连接

序号：sep : 报文序号

ack: 确认号

·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

3必须是三次 如果少于三次，会造成系统资源的浪费

### 4.9tcp的四次挥手

·tcp四次挥手是用于断开连接

Tcp断开连接时发送的四个数据包，确保断开连接

1概念

2简单描述过程

3必须是四次 （新的连接收到老的数据包）

4主动发起断开的一方必须等待2个msl的时间

msl是任何报文段被丢弃前在网络内的最大时间

为什么要等待2个msl的时间？

当tcp的一段主动发起关闭，三次挥手后完成后发送第四次挥手时的ack包就进入这个状态，等待2mal时间的主要目的是:防止最后一个包对方没有收到，那么对方在超时后就会重新发送第三次挥手的包，此时主动关闭端可以再次发送一个ack应答包。

### 4.10ip地址和域名(http协议)

Htpp基于tcp的一个协议

Ip地址：网络中计算机的唯一标示 点分十进制

Ipv4 和 ipv6

域名:

·doman name

·因为网络中使用ip地址，但是ip地址不容易记忆，所以有了更好记忆的域名

·特殊域名 localhost

### 4.11DNS及浏览器请求服务器的过程

·浏览器输入网址-->先去本地DNS服务器查询---->去远程DNS服务器查询--->拿到地址建立tcp连接

·本地DNS是一个文件

Windows系统：c:/windows/systes/drivers/etc/hosts

linux系统：/etc/hosts

### 4.12HTTP协议概述

·HTTP协议：超文本传输协议，主要用于传输和发布HTML页面（网页）

··HTTP两部分构成：

·1请求协议

·2响应协议

协议是由协议项构成

协议项：1）协议名，2）协议值

### 4.13HTTP 协议格式查看

谷歌火狐

### 4.14【重点】HTTP协议请求报文格式

·请求报文格式

·请求行

请求方式 资源路径 协议及版本\r\n

·请求头

协议项

协议名：协议值\r\n

·请求空行

请求空行的作用：分割请求头和请求主体

·请求主体

请求主体（post请求才有）：浏览器要发送给服务端的内容

### 4.15HTTP【重点】响应报文的分析

·响应协议的

·响应行

·协议及版本 状态码 状态描述

常见转台码：

200 一切正常

404 客户其你去的资源不存在

302 重定向

·响应头

·协议项构成

·协议名：协议值

·响应空行

分割响应头和响应主体

·响应主体

服务器响应给客户端的数据

### 4.16长连接和短连接

·长连接

·一次连接多次数据传输 通信结束关闭连接

特点：要不连不上，一旦连接上，速度有保证，当瞬间访问压力比较大，服务器不可用

·短连接

·一次连接，一次传输，一次关闭

特点：会频繁的建立连接和关闭，当瞬间访问压力比较大，服务器反应慢

## 5基于TCP的Web服务器

### 5.1n今日目标

### 5.2【案例】模拟浏览器实现

#1导入模块

#2创建套接字

#3建立连接

连接域名 web服务器默认端口80

#4拼接请求协议

 #4.1请求行

request\_line = "GET / HTTP/1.1\r\n"

    #4.2请求头

request\_header = "Host:www.icoderi.com\r\n"

    #4.3请求空行

request\_blank = "\r\n"

    #整体拼接

request\_data = request\_line + request\_header + request\_blank

#5发送请求协议

Send(request\_data).encode() 默认字符串 比武二进制发送

#6接受服务器相应内容

#7保存内容

#7保存内容

        #7.1\r\n的位置

loc = recv\_text.find("\r\n\r\n")

        #7.2截取字符串

html\_data = recv\_text[loc+4:]

print(html\_data)

        #7.3保存内容到文件中

with open("index.html","w") as file:

    file.write(html\_data)

#8关闭连接

### 5.3【重点】返回固定数据

#1导入模块

#2创建套接字

#3设置地址重用

       #当前套接字          地址重用            地址重用tcp\_client\_socket.setsockopt(socket.SOL\_SOCKET,

socket.SO\_REUSEADDR,True)

#4绑定端口

#5设置监听   让套接字由主动变为被动

#6接受客户端的连接       定义函数request\_handler()

Def reques\_handler(new\_client\_sockent,ip\_port)

#7接受客户端发送的请求协议

Request\_Data = New\_cliet\_socket.recv(1024)

#8判断协议是否为空（客户端下线）

if not request\_data:

        print("[%s]客户端已经下线"% str(ip\_port))

        new\_client\_socket.close()

        return

#9拼接响应报文

·响应行

·响应头

·响应空行

·响应主体

#10发送响应报文 (记得encode转码)

#10关闭和当前客户端的连接

### 5.4【重点】返回固定页面

·改进 返回固定内容的代码

with open("G:/GitPro/pygame/LINUX系统/linux实战/5.0/HTML5箭头射击游戏代码/index.html","rb") as file:

        response\_body = file.read()

    #--响应全体拼接

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

·改进全体拼接

### 5.5【重点】返回指定界面

·核心思路，获取请求的资源路径

·解码请求协议

·得到请求行

·拆分请求行

·打开指定的文件

### 5.6【重点】返回指定页面存在问题

·访问的页面不存在

·对打开文件的代码进行一场捕获

try:

        with open("G:/GitPro/pygame/LINUX系统/linux实战/5.0/static/"+file\_path,"rb") as file:

            response\_body = file.read()

    except Exception as e:

        #重新修改响应行 404

        #相应信息为错误信息

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

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

        #将字符串转换为二进制

        response\_body = response\_body.encode("gbk")

·访问默认首页问题，直接访问地址不加路径报错

解决思路，如果不加路径 ，请求行中得到“\”,一次作为判断条件

 if file\_path == "/":

        file\_path = "index.html"

### 5.7【重点】使用面向对象的思想进行封装

·新建类WebSever

·创建对象方法\_\_init\_\_(),start()

·修改代码

1. 把套接字初始化的操作放在\_\_init\_\_()里面
2. 吧接受客户端连接的代码放到srart()里面
3. 把request\_hander()编程对象方法
4. 在main()函数中创建对象

对象为ws = WebSever()然后启动ws.start()

### 5.8【重点】服务端基础框架构建-1

·创建application文件夹

·在application文件夹创建了qpp模块

·app,py application 函数：

·修改request\_hendle()把核心代码放到app模块的application函数

·app.py 创建parse\_request()专门解析请求的路径

### 5.9【重点】服务端基础框架构建--2

·在application文件夹创建了utils模块

·在utils模块创建一个拼接响应报文的函数

·修改app模块application的返回响应协议部分

### 5.10【重点】给服务器加上命令行参数

### 5.11【重点】网游服务器

·拷贝游戏资源到 项目目录中

·在websever初始化方法中，配置一个字典

        #定义类的实例属性    字典为空

        self.projects\_dict = dict()

        #定义实例属性，保存要发布的项目的路径

        self.current\_dir = ""

        #

        self.projects\_dict["植物大战僵尸--普通版"] = "zwdzjs-vl"

        self.projects\_dict["植物大战僵尸--外挂版"] = "zwdzjs-v2"

        self.projects\_dict["保卫萝卜"] = "tafang"

        self.projects\_dict["读心术"] = "dxs"

        self.projects\_dict["2048"] = "2048"

        print(self.projects\_dict)

·初始化项目的方法

 #-定义一个初始化项目的方法

    def init\_\_porject(self):

        # 2.1 显示所有可以发布的游戏菜单

        # list(self.projects\_dict.keys()) 取出字典的key 并且转换为列表

        keys\_list = list(self.projects\_dict.keys())

        # 遍历显示所有的key

        # enumerate(keys\_list)

        # [(0,'植物大战僵尸v1'), (1, '植物大战僵尸v2') ...]

        for index, game\_name in enumerate(keys\_list):

            print("%d.%s" % (index,game\_name))

        # 2.2 接收用户的选择

        sel\_no = int(input("请选择要发布的游戏序号:\n"))

        # 2.3 根据用户的选择发布指定的项目 （保存用户选择的游戏对应的本地目录）

        #  根据用户的选择，得到游戏的名称（字典的key）

        key = keys\_list[sel\_no]

        #  根据字典的key 得到项目的具体路径

        self.current\_dir = self.projects\_dict[key]

该方法在wersever的初始化方法中调用

·修改request\_handler()的代码

·修改浏览器优先解析方式

修改utils里面的响应函数的响应头 ：

Response\_header += ”Content - Type:text/html\r\n”

## 6多任务--线程

### 6.1今日目标

### 6.2多任务的概念

·什么是多任务

同一时间有多个任务在执行，就是多任务

Python程序默认是单任务

### 6.3【重点】线程---基本使用

·主线程

程序启动后有一个默认的主线

作用：1创建子线程

2等其他子线程执行结束，做关闭操作

·子线程：程序的一个分支

·子线程创建：

1. 导入模块import threading
2. 创建线程对象 threading.Thread(target=执行的分支函数名)
3. 启动子线程 线程对象.start()

### 6.4【重点】线程名称和数量

·获取线程数量：获取活跃的线程数量

#---threading.enumerate()---当前活跃的线程对象列表

#长度len()

·获取线程名称

#threadding.current\_thread() 获取当前的线程对象，对象中含有名称

### 6.5【重点】线程数及顺序

·线程参数

·元组

Threading.Tread(target=XXX,args=(参数1,参数2,....))

元组中参数顺序与函数的参数顺序一致

·字典

threading.Thread(target=XXX,kwargs={"参数名:参数值,"参数名":参数值.......})

无序 需对应

·元组和字典

threading.Thread(target=sing,args=[参数1，参数2，参数3....]，kwargs={"参数名:参数值,"参数名":参数值.......})

·线程顺序

线程的执行顺序是无序的（线程有cpu调度执行，cpu根据系统的运行状态自行运行）

### 6.6【重点】守护线程

·线程守护：子线程和主线程的一种约定（挡住线程结束后，子线程也结束）

·设置方法子线程.setDAemon(True)

### 6.7【重点】并行和并发概念

·并发：任务数大于CPU核心数，这就是并发执行

·并行：任务书小于或等于CPU核心数，这就是并行

### 6.8【重难点】自定义线程类

·自定义线程类的步骤

·导入模块

·创建类并继承threading.Thread

Class MyThread(threading.Thread):

·重写父类的run方法

def run(self):

pass

·创建对象并且调用.start()

methread = MyThread()

methread.start()

·底层原理

Thread类

·run方法

·start()方法

Start()方法中调用run()方法

自定义线程的init方法问题

子类需要先调用父类的初始化方法，子类在初始化

### 6.9【重点】多线程共享全局变量

·多线程之间可以共享全局变量

### 6.10【难点】多线程--共享全局变量--问题

·存在的问题：当多个线程同时访问同一个资源，出现资源竞争的问题

·解决方案：优先让某个线程先执行

线程对象.join()

缺点：把多线程变成了单线程，影响整体性能

### 6.11同步

·同步：在多任务中，多个任务执行有先后顺序，一个执行完毕另外一个在执行

·异步：在多任务中，多个任务同时执行

·线程锁机制：当线程获取资源后，立刻进行锁定，资源调用完毕后再解锁，有效的保证同一时间只有一个线程在使用资源

### 6.12【重点】互斥锁

·互斥锁使用分三步

·创建一把锁

Lock\_1 = threading.Lock()

·锁定资源

Lock1.acquire()

·解锁资源

Lock1.release()

·互斥锁使用原则：尽可能少的锁定竞争资源

### 6.13死锁

·死锁：在多线程中，两个线程都占用一些资源，而且同一时间等待对方先释放资源，这种状态就是死锁状态

·锁用完后 及时释放

### 6.14【案例】多任务版的UDP聊天器

·创建一个子线程用来接受信息

 #创建子线程 ，单独接受用户信息

    t1 = threading.Thread(target = recv\_msg,args=(udp\_socket, ))

    #启动子线程

    t1.start()

    while True:

        sel\_num = int(input("请输入选项:\n"))

        if sel\_num ==1:

            print("你选择地是发送信息")

·删除接受信息菜单和判断

·接受多条信息

def recv\_msg(udp\_socket):

    """接受信息"""

    while True:

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

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

        print("接收到[%s]的消息:%s" % (str(ip\_post),recv\_text))

### 6.15【案例】多任务版UDP聊天器（二）

·让子线程守护主线程

#创建子线程 ，单独接受用户信息

    t1 = threading.Thread(target = recv\_msg,args=(udp\_socket, ))

    #子线程守护主线程

    t1.setDaemon(True)

    #启动子线程

    t1.start()

### 6.16TCP服务端框架

·TCP服务端

1）"""

#1导入模块

#2创建套接字

#3设置地址可以重用

#4绑定端口

#5设置套接字由主动变被动

#6接受客户端连接

#7接受客户端方的信息

#8解码进行输出

#9进行关闭和当前客户端的连接

"""

·支持多线程

思想：没来一个客户端，创建一个线程

while True:

    #6接受客户端连接

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

    print("欢迎新用户上线",ip\_port)

    #recv\_msage(new\_server\_socket,ip\_port)

    #创建线程

    thread\_recv\_msage = threading.Thread(target=recv\_msage,args=(new\_server\_socket,ip\_port))

    #设置线程守护

    thread\_recv\_msage.setDaemon(True)

    #线程运行

thread\_recv\_msage.start()

### 6.17【5-11案例】简单WEB服务器多任务版

## 7多任务--进程

### 7.1今日目标

### 7.2，进程以及状态

·进程：资源分配的基本单位，也是线程的容器

·进程的状态：

1. ：新建
2. ：就绪
3. 运行
4. 等待
5. 死亡

### 7.3【重点】进程--基本使用

·进程使用：

1. 导入模块

Import multiprocessing

1. 创建子进程对象

Process\_obj = Multiprocessing.Process(target=work1)

1. 子进程启动

### 7.4【重点】进程名称.pid

·获取进程名称

Multiprocessing.current\_process()

设置名称：

Process\_obj = Multiprocessing.Process(target=XXXX,name=名称）

·获取进程的编号

1 Multiprocessing.current\_process().pid

2 os.getpid()

·获取进程的父ID

Os.getppid()

·结束进程

Kill -9 进程编号

### 5【重点】进程--参数传递，全局变量问题

·进程的参数传递

1. args 元组
2. Kwargs 子典
3. 混合远足字典

·进程间全局共享变量问题

进程间不能共享全局变量

底层原理：子进程会复制主进程的资源到内部运行

### 7.6【重点】守护主进程

·进程守护：子进程和主进程的一张约定，当主进程结束时。子进程也随之结束

Process\_obj.daemon= True

·结束子进程

   #终止子线程执行    不是守护  就是死之前  杀死他

process\_obj.terminate()

### 7.7进程，线程对比

·进程和县城的对比

·进程，资源分配的基本单位，线程，CPU调度的基本单位

·进程运行需要独立的内存资源，线程运行需要的是必不可少的一点资源

·进程切换慢，线程切换更快

·线程不能独立运行，必须运行在进程中（进程提供资源）

·CPU密集型 进程优先， I/O 密集型 使用线程

·程序更稳定 进程， 线程相对不够稳定

·实际生活中， 多为组合使用

### 7.8【重点】消息队列--基本操作

·消息队列的目的：为了实现进程之间的通信

·队列的创建：

1导入模块 multiprocessing

2创建队列对象 queue = multiprocessing.Queue(5) #队列长度为5

·队列的操作

·放入值quese.put(值) #从队列尾部放入值

·取值 quese.get() -->从队列头部取值 get时 没参数

·XXX\_nowait方式

·放入值 put\_nowait() 特点：队列为满，同put()，队列已满，会报错，不等待

·取值 get\_nowait() 特点：队列未空，同get()，队列已空，会报错，不等待

### 7.9消息队列--常见判断

·常见的判断

·full() 判断是否已满

·empty() 判断是否已空

·取出队列中消息的数量

·qusize()

########此处有坑 没遇到

### 7.10【重点】Quese实现进程之间的通信

思路：利用队列在两个进程间进行传递，进而实现数据共享

·write\_queue(queue)

·read\_queue(queue)

·创建一个空队列

·把空队列作为参数，先传递给写队列，在传递给读队列

·join() 优先让你哥进程执行完成，另一个进程才能继续

### 7.11【重点】进程池

·进程池，是一个进程的容器，可以自动帮我们创建指定数量的进程，并且管理进程及工作

。创建方法

1导入模块 import multiprocessing

2创建进程池

polol = multiprocessing.Pool(3)

。进程池工作方式

·同步方式

pool.apply(函数名，（参数1，参数2.......）)

进程池中的进程，一个执行完毕后另外一个才能执行，多个进程有先后顺序

·异步方式

pool.apply\_async(函数名，（参数1，参数2........）)

进程池中的进程，多进程同时运行，没有先后顺序，效率提高

1. 进程池要close() 表示不再接受新的任务

pool.close()

1. 还要join( )表示要主进程等待进程池执行结束后在退出

pool.join()

### 7.12进程池中的QUEUE

进程池中的进程的通信

·获取方法

#创建一个进程池

    pool = multiprocessing.Pool(2)

    #创建进程池中的队列

    queue = multiprocessing.Manager().Queue(5)

异步方法：

 #异步执行

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

    result.wait()

    pool.apply\_async(read\_queue,(queue, ))

    #pool.close()表示不再接受新的任务

    #pool.join()表示主进程会等待进程池执行结束后再退出

    pool.close()

    pool.join()

### 7.13【案例】:文件夹copy器

思路：

"""

#1定义变量保存源文件夹的路径 和目标文件夹的路径

#2在目标路创建新的文件夹

#3获取源文件夹中所有的文件（列表）

#4遍历列表，得到所有的文件名

#5定义函数进行文件拷贝

文件拷贝：

参数：源文件夹路径    目标文件夹路径   文件名

1，拼接源文件和目标文件的具体路径

2，，打开源文件，创建目标文件

3，读取源文件的内容，写入到目标文件中（循环）

"""

#创建文件夹

Os.mkdir(路径)

获取文件夹中的内容

Os.listdir(路径)

### 7.14【重点】简单Web服务器多进程版

5.0--简单web服务器进程版本.py

### 7.15【重点】可迭代对象以及检测方法

·可迭代对象

#可遍历对象就是可迭代对象

#列表元祖字典字符串 都是可迭代对象

#数值100和自定义Myclass都是不可迭代的

#Myclass对象所属的类MyClass，如果包含了\_\_iter\_\_()，此时myclasss就是一个可迭代对象

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

·可迭代对象的检测：

#3检测一个对象是否可以迭代，用isinstance()  函数检测

### 7.16【重难点】迭代器及其使用方法

迭代器的作用：

1. 记录当前迭代的位置 2）配合next()获得可迭代对象的下一个元素值

·获取迭代器

Iter(可迭代对象)

·获取可迭代对象的值

Next(迭代器)

·for 循环的本质

for循环的本质：

1) 通过 iter(可迭代对象)   获得要遍历对象的迭代器

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

#自定义迭代器类

#1)，必须含有\_\_iter\_\_()方法

#2）,必须含有\_\_next\_\_（）方法

class MyIterator(object):

    def \_\_iter\_\_(self):

        pass

            #当next(迭代器时)，会自动调用该方法

    def \_\_next\_\_(self):

        pass

## 8多任务--协程

### 8.1今日目标

### 8.2【重难点】自定义迭代对象，迭代

#1MyList()类

    初始化方法\_\_init\_\_()

    对外提供迭代器\_\_iter\_\_()

    添加数据addIter()

2MyListIterator()

    初始化方法\_\_init\_\_()

    迭代器方法\_\_iter\_\_()

    获得下一个元素值\_\_next\_\_()

目标：

mylist = Mylist()

for value in mylist:

    print(value)

### 8.3【案例】斐波那契数列

·自定义了一个迭代器

·必须含有\_\_iter\_\_()方法

·还得有一个\_\_next\_\_()方法

·核心思想：

让a保存第一列的值

让b保存第二列的值

A=b

B=a+b

取a的值 ，得到斐波那契数列

### 8.4【重点】生成器

·生成器概念：生成器是一个特殊的迭代器（按照一定的规律生成数列）

·生成器创建方式

#生成器的创建一

1）列表推导式

data\_list2 = (x\*2 for x in range(10))

print(data\_list2)

#通过next获取下一个值

value = next(data\_list2)

print("------->",value)

#生成器的第二种创建方式

#使用yield创建了一个生成器

def text2():

    yield 10

n = text2()

yulue = next(n)

print("n------->",yulue)

### 8.5生成器案例：斐波那契数列

·思路：

"""

#创建一个生成器

    #目标：实现斐波那契数列

    #1)定义变量保存第一列和第二列的值

    #2）定义变量保存当前生成的位置

    #3）循环生成数据  条件是当前列数《总列数

    #4）保存a的值

    #5）修改a  \ b 的值

    #6）返回a的值   使用yield

#2定义变量 保存生成器

#next(生成器)得到下一个元素值

"""

·yield的作用

 #6）返回a的值   使用yield

                #yield的作用：

                #能够充当return的作用

                #能够保存程序的运行状态，并且暂停程序的执行

                #当next()的时候，可以继续唤醒程序从yield位置往下执行

        yield data

### 8.6生成器使用注意

·生成器中return的作用

可以结束生成器的运行

·生成器中send()

用法：生成器.send(传递给生成器的值)

传递fib.send(1)

接收XXX = yield data

### 8.7协程 -----yield

·能够使用yield关键字实现协程

协程：在不开辟新的线程的基础上 实现新的任务 写成是一个特殊的生成器

·手动实现一个协程:

·work1生成器

·work2生成器

·获取生成器 w1=work1 w2 = work2

·让协程运行循环起来

While True:

next(w1) next(w2)

### 8.8协程----greenlet

·Greenlet是python的C扩展，来源于Stackless Python， 旨在提供一个可供自动调度的微线程，即协程

·使用步骤

1. 导入模块 from greenlet import greenlet
2. 创建greenlet对象 g1 = greenlet(函数名)
3. 切换任务 g1.switch()

### 8.9【重点】协程----genvet

·gevent 也是第三方库，自动调度协程，自动识别程序中的耗时操作

·使用步骤：

1. 导入模块import gevent
2. 指派任务 g1 = gevent.spawn(函数名，参数1，参数2,......)
3. join()让主线程等待协程执行完毕后再退出

·Gevent 不能识别耗时操作的问题

·替换time.sleep( 0.5) ---->gevent.slee(0.5)

·打猴子补丁

1. 导入模块 from gevent import monkey
2. 破解所有 monkey.patch\_all()

猴子补丁的作用：

1. 在运行时替换方法，属性等
2. 在不修改第三方代码的情况下，增加原来不支持的功能
3. 在运行是为内存中的对象增加patch而不是在磁盘的源代码中增加

### 8.10进程，线程，协程对比

### 8.11【案例】：并发下载器---协程版

·案例实现思路：

#定义啊要保存的图片路径

#调用文件下载的函数，专门下载文件

#文件下载函数

1根据url地址请求网络资源

2在本地创建文件，准备保存

3读取获得到的网络资源 （循环）

4把读取道德资源写在本地

5做异常捕获

·打开网址 对应的资源

Urllib.request.urlopen(网址)

·joinall()批量把协程 添加join

Joinall(协程的列表)

#批量进行join()，需要一个参数协程的列表

    gevent.joinall([

            gevent.spawn(downlowd\_image,img\_url1,"1.gif"),

            gevent.spawn(downlowd\_image,img\_url2,"2.gif"),

            gevent.spawn(downlowd\_image,img\_url3,"3.gif")

    ])

### 8.12【案例】协程版web服务器

1导入模块gevent

2创建协程 指派任务gevent.spwan()

def start(self):

        #启动服务器

        print("web服务器已经启动，等待客户端连接中....")

        #6接受客户端的连接

        while True:

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

            print("新客户端上线：",ip\_port)

            #7接受数据  响应请求

            #导入模块gevent

            #使用gevent分派任务

            #要打补丁猴re子补丁

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

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

3打补丁

From gevent import monkey

Monkey.patch\_all()

## 9正则表达式

### 9.1今日目标

### 9.2【了解】正则表达式概述

·正则表达式：规则表达式（一套特殊的规则）

·正则表达式的作用：

·验证数据的有效性（查找）

·替换文本内容

·从字符串中提取子字符串（爬虫思想）

### 9.3【了解】测试工具介绍regexbuddy

·Regexbuddy 用来测试则很难规则表达式的正确性

·使用：

1. 选择版本
2. test选项卡

### 9.4【记忆】匹配单个字符

. 匹配任意单个字符

[] 列举 匹配[]里任意内容

[ab] 匹配a或者b

1. z]
2. Z]

[0-9]

\d 匹配所有数字

\D 匹配非数字

\s 匹配空格

\S 匹配非空格

\w 匹配字母 数字 下划线

\W匹配非数字 非字母 非下划线

### 9.5【记忆】匹配多个字符

\* 匹配前一个字符出现0次或无限次 ，即可有可无

+ 匹配前一个字符出现1次或无数次 ， 即至少有一次

? 匹配前一个字符出现1次或0次， 即要么一次要么没有

{m} 匹配前一个字符出现m次

{m,n} 匹配前一个字符出现 m到n次

### 9.6【记忆】匹配开头结尾

·^ 匹配以后一个字符开头

尖括号有两个作用：

1. 匹配以指定字符开头

^[a-zA-Z0-9\_]+\w 匹配以 小写字母 大写字母 下划线开头

1. 用于中括号内部 用于取反

[^he] 匹配 不含有 h 和 e 的字符

·$ 匹配以前一个字符结尾

\d$ 匹配以数字结尾

### 9.7【理解】re模块操作

·re模块的作用 ：python提供的用于正则操作的模块

·re的使用步骤：

1. 导入模块 import re
2. 使用match() 方法进行检测

#2t通过match   方法，验证正则

    #re.match(正则表达式，要验证的字符串)

    #match()方法如果匹配成功，返回match\_object对象

    #如果失败  返回None

result = re.match("\w{4,20}@163\.com$","hello@163.com")

1. 判断是否检测成功

#3加个判断  是否匹配成功

if result:

    print("匹配成功")

    print("匹配结果:", result.group())

else:

    print("匹配失败")

#4如果成功获取匹配的结果

1. 取出匹配的具体内容

Result.group() 获取匹配的具体内容

### 9.8【理解】re模块之分组“|”

·“|”多个正则满足任意一个都可以

^[0-9]?[0-9]$|^100$

### 9.9【理解】匹配分组之“()”

·分组，整体匹配

result = re.match("\w{4,20}@(163|126|qq|sina)\.com$","hello@qq.com")

·提取子字符串

默认第一个小括号里的内容为1

result = re.match("(\d{3,4})-(\d{7,8})","012-12345678")

if result:

    print("匹配成功",result.group())

    print("匹配的区号：",result.group(1))

    print("匹配的电话号码：",result.group(2))

### 9.10【理解，难】匹配分组之“\”

·引用分组

\1 表示引用第一组

#result = re.match("<([a-zA-Z0-9]+)>.\*</\\1>","<html>textdsds</html>")

result = re.match("<([a-zA-Z0-9]+)><([a-zA-Z0-9]+)>.\*</\\2></\\1>","<html>textdsds</html<html><h1>hihiiu</h1></html>

[\\表示转移第二个\让他成为一个普通的\](\\\\表示转移第二个\\让他成为一个普通的\\)

·分组起别名

·起别名

#?P<name1>给分组起别名

·引用别名

 #(?P=name1)引用别名为name1的分组

·整体代码

result = re.match("<(?P<name1>[a-zA-Z0-9]+)><(?P<name2>[a-zA-Z0-9]+)>.\*</(?P=name2)></(?P=name1)>","<html><h1>hihiiu</h1></html>")

### 9.11【理解】re模块的高级用法

·search()：在需要匹配的字符串中搜索要匹配的内容

#一，search的使用

#result = re.search("\d+","阅读次数：9999")

知道match和search()的区别

#使用math方法，正则表达式从需要检测开头进行匹配，成功返回matchobject对象，如果失败返回None

        #search在需要检测字符串以搜索的方法进行匹配，全文进行匹配，有则返回matchobject对象，如果失败返回None

·findall()：在需要匹配的字符串中查找所有满足大的内容，返回值是列表

#result = re.findall("\d+","阅读次数：9999,转发次数：6666，评论次数：8888"))

·sub(正则表达式，新的内容，要替换的字符串)，返回值是个列表

result = re.sub("\d+","10000","阅读次数：9999,转发次数：6666，评论次数：8888")

·split(正则表达式，待拆分的字符串)，返回值是一个列表

result = re.split("\d+,\*","阅读次数：9999,转发次数：6666，评论次数：8888")

### 9.12【理解】贪婪和非贪婪

贪婪：默认，在满足正则的情况下尽可能多的取内容

非贪婪：在满足正则的情况下尽可能少的取内容

贪婪转变为非贪婪

#把贪婪模式转换为非贪婪模式：?  在  + \* ? {}后面添加?    可以变成非贪婪

result = re.match("aaa(\d+?)","aaa12345667")

### 9.13【记忆】r的作用

r的作用：让正则中的\变为原生含义

### 9.14【应用】案例：正则获得电影下载链接

·思路：

1定义函数获取列表页的内容页地址get\_movie\_links()

1定义列表页的地址

2打开url地址，获取数据

3将数据进行解码得到html文本

4使用正则得到每一个内容页地址

4.1遍历，取出内容页地址

4.2拼接内容页地址

4.3打开内容页地址

4.4获取数据并读取

4.5解码内容页数据得到html文本

4.6使用正则获得下载链接

4.7定义字典保存键值

4.8返回字典

2主函数main()

1调用get\_movir\_links()，得到字典

2遍历字典，显示下载内容

## 10MySQL数据基本操作

### 10.2【了解】MySQL数据库概念及作用

·数据库概念：一些特殊格式的文件的集合

·数据库作用：用来存储各种数据

·数据库使用的优势：

·持久存储

·存取效率高

·支持可扩展

### 10.3【了解】数据库分类及特点

·关系型数据库

基于关系模型简历，用二维表进行数据存储的数据库

MySQL Oracle

·非关系型数据库

不是基于二维表，基于key-value方式存储

MongoDB Redis

### 10.4【了解】数据库管理系统DBMS

数据库管理系统：指的是管理数据库的一套软件

1. 客户端
2. 服务端
3. 数据库的文件

关系型数据库管理系统：RDBMS

SQL作用：客户端和服务端通信的特殊语言，含义：结构化查询语言

SQL语句主要分为：

DQL： 数据查询预言 如 select

DML： 数据操作语言 ，对数据进行增加，修改，删除，如insert,udpate,delete

TPL: 事务处理语言 ， 对事物进行处理，包括 begin， transaction,commit,rollback

DCL： 数据控制语言 ， 进行授权与权限回收，如grant,revoke

DDL: 数据定义语言，进行数据库表的管理等创建删除，如create，drop等

对于web程序员来讲，重点是数据的crud（增删改查），必须熟练使用DQL，DML，能够编写DDL完成数据库，表的操作，其他语言TPL DCL CCL了解即可

数据库的核心元素：数据库：数据库系统

数据表：二维表，行，列

记录：（二维表的一行，要求完整性）

字段：（二维表的列，要求一列的数据类型要求一致）

数据库保存的流程

·创建数据库

·创建数据表

·字段

·保存数据

### 10.5MySQL环境搭建

·MySQL 数据库管理系统的安装

Sudo apt-get install myaql-server

·MySQL 数据库的配置

配置文件：\*\*/etc/mysql/mysql.conf.d/mysqld.cnf\*\*

·MySQL 数据库的操作

·启动 sudo server mysql start

·检测 ps -ajx | grep mysql

·重启 sudo-server mysql restart

·停止 sudo server mysql stop

·客户端连接服务器

·图形化工具

·命令行连接

·可以看到密码：mysql -u用户名 -p密码

·不可看到密码：mysql -u 用户名 -p

·命令行退出

Exit

quit

### 10.6【了解】数据完整性

·实体完整性

主键约束：不能为空，也不能重复 primary key

唯一约束：能为空，但是不能重复 unique key

·域完整性约束

非空约束： 该字段信息不能为空 not null

默认约束：这个字段可以设置默认值 default

·参照完整性

外键约束： 建立表和表之间的关系foreign key

·用户定义完整性

字典的值必须是设定的范围之内

### 10.7【理解】客户端Navicat使用（一）

·建立连接

连接------>MySQL---->主机，端口，密码---->确定

·创建数据库

1. 链接
2. 右键 创建数据库 -->名称，字符集，排序规则

·创建数据表

1. 打开数据库
2. 右键创建表
3. 输入字段（光标移动）
4. 保存，输入表名

·修改数据表

1. 右键表名---->设计表---->修改字段---->保存

### 10.8【理解】客户端Navicat的使用

·打开表

双击表名，打开表

·添加数据

单击单元格可以添加内容，自动增长列，不需要写

·修改表数据

Ctrl+s 底部对勾

·删除数据

双击单元格修改内容 删除数据选中某行 点击底部减号

·导出，备份

导出：右键数据库，转储SQL文件，数据和结构

导入：新建数据库，右键运行SQL文件

### 10.9【记忆】MySQL数据类型

选择合适的类型保存数据1）节省空间，2）提升查询效率

·数值型

整型：

int tinfint.

浮点型:

Float double

表示的精度不同，float6位有效，double16位有效

Decimal浮点型 用来更加精确的保存小数

·字符串

Char 255个字符，固定大小

Varchar 65535字节，不固定（可变）

Text 64K

·枚举类型

·enum() 1-2个字节存储数据

·enum(“值1”,”值2”,”值3”,......)

·时间

·date 年月日

·datetime 年月日 时分秒

·time 时分秒

### 10.10【记忆】SQL--登陆和登出数据库

·登陆 mysql -u用户名 -p

·退出 Ctrl +d quit exit

·指令

查看数据库版本，select cersion();

查询当前时间，select now();

### 10.11【记忆】数据库操作

--01 mysql 数据库的操作

ctrl + a 回到开头

ctrl + e 回到结尾

ctrl + l 清屏

-- 连接数据库

--显示密码

mysql -uroot -pmysql

-- 不显示密码

mysql -uroot -p

mysql

-- 退出数据库

exit

quit

ctrl + d

-- sql语句最后需要有分号;结尾

-- 显示数据库版本 version

select version();

-- 显示当前时间

select now();

-- 查看当前使用的数据库

select database();

-- 查看所有数据库

show databases;

-- 创建数据库(python\_db)

-- create database 数据库名 charset=utf8;

create database python\_db;

-- 指定编码的数据库创建

create database python\_db1 charset=utf8;

-- 查看创建数据库的语句

-- show create database ....

show create database python\_db;

-- 使用数据库

-- use 数据库的名字

use python\_db1;

-- 删除数据库

-- drop database 数据库名;

drop database python\_db;

### 10.12【记忆】SQL--命令表结构的创建

--02 数据表的操作

-- 查看当前数据库中所有表

show tables;

-- 创建表

-- int unsigned 无符号整形

-- auto\_increment 表示自动增长

-- not null 表示不能为空

-- primary key 表示主键

-- default 默认值

-- create table 数据表名字 (字段 类型 约束[, 字段 类型 约束]);

-- 创建 classes 表(id、name)

create table classes(

id int unsigned primary key auto\_increment,

name varchar(10) not null,

num tinyint

);

-- 查看表结构

-- desc 数据表的名字;

desc classes;

-- 创建 students 表(id、name、age、high (decimal)、gender (enum)、cls\_id)

create table students(

id int unsigned primary key auto\_increment,

name varchar(20) not null,

age tinyint(1) ,

high decimal(3,2),

gender enum("男","女","妖"),

cls\_id int unsigned

);

-- 查看表的创建语句

-- show create table 表名字;

show create table students;

### 10.13【记忆】表结构的修改

-- 查看表结构

-- desc 数据表的名字;

desc classes;

-- 创建 students 表(id、name、age、high (decimal)、gender (enum)、cls\_id)

create table students(

id int unsigned primary key auto\_increment,

name varchar(20) not null,

age tinyint(1) ,

high decimal(3,2),

gender enum("男","女","妖"),

cls\_id int unsigned

);

-- 查看表的创建语句

-- show create table 表名字;

show create table students;

-- 修改表-添加字段 生日 datatime

-- alter table 表名 add 列名 类型;

alter table students add birthday datetime;

-- 修改表-修改字段：不重命名版

-- alter table 表名 modify 列名 类型及约束;

alter table students modify birth date not null;

-- 修改表-修改字段：重命名版

-- alter table 表名 change 原名 新名 类型及约束;

alter table students change birthday birth datetime;

-- 修改表-删除字段

-- alter table 表名 drop 列名;

alter table students drop birth;

-- 删除表

-- drop table 表名;

-- drop database 数据库;

drop table students;

### 10.14【记忆】SQL命令--表数据增删改查

--03 增删改查(curd)(重点--记忆)

-- 增加

+-------+------------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------+------------------+------+-----+---------+----------------+

| id | int(10) unsigned | NO | PRI | NULL | auto\_increment |

| name | varchar(10) | NO | | NULL | |

| num | tinyint(4) | YES | | NULL | |

+-------+------------------+------+-----+---------+----------------+

-- 全列插入

-- insert [into] 表名 values(...)

-- 主键字段 可以用 0 null default 来占位

-- 向classes表中插入 一个班级

insert into classes values(1, 'python20', 70);

insert into classes values(null, 'python19', 68);

-- 向students表插入 一个学生信息

| Field | Type | Null | Key | Default | Extra |

+--------+-------------------------+------+-----+---------+----------------+

| id | int(10) unsigned | NO | PRI | NULL | auto\_increment |

| name | varchar(20) | NO | | NULL | |

| age | tinyint(1) | YES | | NULL | |

| high | decimal(3,2) | YES | | NULL | |

| gender | enum('男','女','妖') | YES | | NULL | |

| cls\_id | int(10) unsigned | YES | | NULL | |

+--------+-------------------------+------+-----+---------+----------------+

-- 部分插入

-- insert into 表名(列1,...) values(值1,...)

insert into students values(null, '司马二狗', 18, 1.78, '妖', 1);

insert into students(id, name) values(null, '司马狗剩');

-- 多行插入

insert into students values(null, '欧阳铁娃', 18, 1.78, '妖', 1),(null, '诸葛铁锤', 18, 1.78, '妖', 1);

-- 修改

-- update 表名 set 列1=值1,列2=值2... where 条件;

-- 全部修改

update students set age = 38;

-- 按条件修改

update students set age = 88 where name = '司马狗剩';

update students set high = 1.2 , gender='男' where name='司马狗剩';

-- 查询基本使用

-- 查询所有列

-- select \* from 表名;

select \* from students;

---定条件查询

select \* from students where name='司马狗剩';

select \* from students where id = 2;

-- 查询指定列

-- select 列1,列2,... from 表名;

select id,name from students ;

select id,age from students;

-- 可以使用as为列或表指定别名

-- select 字段[as 别名] , 字段[as 别名] from 数据表;

select id as '编号', name as '姓名' from students;

-- 字段的顺序

select age,name from students;

-- 删除

-- 物理删除

-- delete from 表名 where 条件

delete from students where id = 2;

-- 逻辑删除

-- 用一个字段来表示 这条信息是否已经不能再使用了

-- 给students表添加一个 is\_delete 字段 bit 类型 默认为0

alter table students add is\_delete bit default 0;

-- bit 类型，智能保存 0 或者 1

-- is\_delete = 1 逻辑删除

update students set is\_delete = 1 where id = 3;

-- 数据库备份与恢复(了解)

-- mysqldump –uroot –p 数据库名 > python.sql;

-- mysql -uroot –p 新数据库名 < python.sql;

## 11MySQL数据进阶操作

### 11.1学习目标

### 11.2【应用】练习强化

·查询

·查询所有字段 Select \* from 表名;

·查询部分字段 select 字段1，字段2 from 表名;

·查询字段起别名

select 字段 as “别名” from 表名;

Select 字段 “别名” from 表名;

·给表其别名

select s.id,s.name from students as s；

·去掉重复distinct

Select distinct 字段名 from 表名;

### 11.3【记忆】where之比较运算

·>

·<

·=

·>=或者<=

·!=或者<>

### 11.4【记忆】where之逻辑运算

And 满足所有条件

Or 满足任意条件

Not 条件取反

### 11.5【记忆】where之模糊查询

·like表示模糊查询

·% 表示任意零个或多个

·\_ 表示任意一个字符

### 11.6【记忆】where之范围查询

·in 用于非连续的范围查询

相当于 多个值的or(或关系)

·between 起始值 and 终值

连续的范围 包含起始值和终值 是一种and关系

### 11.7【记忆】where之空值判断

Is null 判断值为空

Is not null 判断值不为空 注意错误(not is full)

### 11.9【记忆】聚合函数

·聚合函数：又称为组函数，对当前表做一个统计

常见的聚合函数：

Count(\*) 统计数量

Max(字段) 找出该字段的最大值

Min(字段) 中出该字段的最小值

Sum(字段) 对该字段进行求和

Avg(字段) 求该字段的平均值

·四舍五入的函数 round（数值,保留的小数位数）

-- 聚合函数

-- 总数

-- count

-- 计算班级学生的总数

select count(\*) from students;

select count(\*) '总人数' from students;

-- 查询男性有多少人，女性有多少人

select count(\*) from students where gender = '男';

select count(\*) from students where gender = '女';

-- 最大值

-- max

-- 查询最大的年龄

select max(age) from students;

select max(age) from students where gender='女';

-- 查询女性的最高 身高

select max(height) from students where gender='女';

-- 最小值

-- min

select min(height) from students where gender='女';

-- 求和

-- sum

-- 计算所有人的年龄总和

select sum(age) from students;

-- 平均值

-- avg

-- 计算平均年龄

select avg(age) from students;

-- 计算平均年龄 sum(age)/count(\*)

select sum(age)/count(\*) from students;

-- 四舍五入 round(123.23 , 1) 保留1位小数

-- 计算所有人的平均年龄，保留2位小数

select round(avg(age),2) from students;

-- 计算男性的平均身高 保留2位小数

select round(avg(height),2) from students where gender = '男';

### 11.10【记忆】group 分组查询

·group by 用于分组（按照某个特定的字段进行分类）

Group by 字段名

·group by + 聚合函数 ，分组统计计算

·group by + group\_coucat() 分组之后+内容链接称为一个字符串

·group by + with rollup 分组加小计

### 11.10. 【记忆】group分组

\* group by 用于分组（按照某个特定的字段进行分类）

> group by 字段名

\* group by + 聚合函数，分组统计/计算

\* group by + group\_concat() 分组 + 内容链接为一个字符串

\* group by + with rollup 分组 + 小计

```

-- group by

-- 按照性别分组,查询所有的性别

select gender from students group by gender;

select name,gender from students group by gender;错误

-- select name,gender from students group by gender;

-- 失败select \* from students group by gender;

-- 计算每种性别中的人数

select gender, count(\*) from students group by gender;

-- 计算每个年龄中的人数

select age, count(\*) from students group by age;

-- 查询 男、女性别中年龄的最大值

select gender, max(age) from students where gender='男' or gender='女' group by gender;

-- group\_concat(...)

-- 查询同种性别中的姓名

select gender, group\_concat(name) from students group by gender;

-- 查询每组性别的平均年龄

select gender, avg(age) from students group by gender;

-- 查询平均年龄超过30岁的性别，以及姓名 having avg(age) > 30(重点)

select gender, avg(age) from students group by gender having avg(age) > 30;

select gender, avg(age) av from students group by gender having av > 30;

-- 查询每种性别的平均年龄和名字

select gender, avg(age), group\_concat(name) from students group by gender;

-- 查询每种性别中的人数多于2个的性别和姓名（重点）

select gender, count(\*) c, group\_concat(name) from students group by gender having c > 2;

-- with rollup 汇总的作用(了解)

select gender, count(\*) from students group by gender with rollup;

·分组的字段要出现在select 的后面

### 11.8【记忆】order 排序

排序：Order by 排序字段一 排序规则 , 排序字段 排序规则 ........;

排序规则：

升序:asc 默认

升序：desc

select \* from students where age between 18 and 34 and gender = "女" order by height desc,age desc,id desc;

### 11.8. 【记忆】order排序

\* 排序：order by 排序字段1 排序规则, 字段2 规则2,.....

\* 排序规则：

\* 升序

> 默认 ， asc

\* 降序

> desc

```

-- order by 字段

-- asc从小到大排列，即升序

-- desc从大到小排序，即降序

-- 查询年龄在18到34岁之间的男性，按照年龄从小到大到排序(默认是asc升序)

select \* from students where age between 18 and 34 and gender='男' order by age;

-- 查询年龄在18到34岁之间的女性，身高从高到矮排序

select \* from students where age between 18 and 34 and gender='女' order by height desc;

-- order by 多个字段

-- 查询年龄在18到34岁之间的女性，身高从高到矮排序, 如果身高相同的情况下按照年龄从大到小排序

select \* from students where age between 18 and 34 and gender='女' order by height desc, age desc;

-- 查询年龄在18到34岁之间的女性，身高从高到矮排序, 如果身高相同的情况下按照年龄从大到小排序,

-- 如果年龄也相同那么按照id从大到小排序

select \* from students where age between 18 and 34 and gender='女' order by height desc, age desc, id desc;

### 11.11【记忆】limit限制记录

所谓limit 限制取出数据的数量 表中第一个数据的下标为0

·limit: 可以使用limit限制取出记录的数量，但是limit必须写在SQL语句的最后

·limit 起始位置 记录数

标准的SQL书写格式：

Select 字段1,字段2,.......

From 表名

[where 条件]

[Group by] 字段名

[order by 字段名 排序规则]

[having 条件]

[limit 起始位置,数量]

·分页 使用的关键字 是limit

核心公式 limit(第X页 - 1) \* 每页个数 , 每页个数

### 11.12【理解】连接

·内连接查询：查询的结果是两个表匹配到的数据，默认是笛卡尔积

·关键字 inner join

·select \* from 表一 inner join 表二 where/on 表一.字段 = 表二.字段

·外连接

·左外连接

主表 Left join 从表

左外连接另外一个表，在从表中没找到匹配，右侧补null

·右外连接

从表 right join 主表

右外连接另外一个表，在从表中没找到匹配，左侧补null

### 11.12. 【记忆】连接

\* 内连接：查询的结果为两个表匹配到的数据，默认是笛卡尔积

\* 关键字 inner join

\* select 字段 from 表1 inner join 表2 where/on 表1.字段 = 表2.字段

\* 外连接

\* 左外连接 主表 left join 从表

\* 左外连接另外一个表，在从表中没有找到匹配，右侧补 NULL

\* 右外连接 从表 right join 主表

\* 右外连接一个表，在从表中没有找到匹配，左侧补 NULL

-- inner join ... on

-- select ... from 表A inner join 表B;

select \* from students inner join classes;

-- 查询 有能够对应班级的学生以及班级信息

select \* from students inner join classes where students.cls\_id = classes.id;

-- 按照要求显示姓名、班级

select students.name, classes.name from students inner join classes where students.cls\_id = classes.id;

-- 给数据表起名字

select s.name, c.name from students s inner join classes c where s.cls\_id = c.id;

-- 内连接的另外一种写法

select s.name, c.name from students s, classes c where s.cls\_id = c.id;

-- 查询 有能够对应班级的学生以及班级信息，显示学生的所有信息 students.\*，只显示班级名称 classes.name.

select students.\*, classes.name from students inner join classes on students.cls\_id = classes.id;

-- 在以上的查询中，将班级名称显示在第1列

select classes.name,students.\* from students inner join classes on students.cls\_id = classes.id;

-- 查询 有能够对应班级的学生以及班级信息, 按照班级编号进行排序

-- select c.xxx s.xxx from students as s inner join clssses as c on .... order by ....;

select \* from students inner join classes on students.cls\_id = classes.id order by classes.id ;

-- 当时同一个班级的时候，按照学生的id进行从小到大排序

select \* from students inner join classes on students.cls\_id = classes.id order by classes.id, students.id ;

-- left join ....on

-- 查询每位学生对应的班级信息

select \* from students left join classes on students.cls\_id = classes.id ;

-- select \* from students right join classes on students.cls\_id = classes.id;

-- 查询没有对应班级信息的学生

-- select ... from xxx as s left join xxx as c on..... where .....

-- select ... from xxx as s left join xxx as c on..... having .....

select students.\* from students left join classes on students.cls\_id = classes.id where classes.id is null;

(注意)不建议使用 select \* from students left join classes on students.cls\_id=classes.id having classes.id is null;

-- right join on

-- 将数据表名字互换位置，用left join完成

select \* from students right join classes on students.cls\_id = classes.id;

```

### 11.13【记忆难】连接查询

### 11.14【了解】自连接

·自连接：一个特殊的内连接

\* 自连接： 特殊的内连接

```

-- 1、查询一共有多少个省

select count(\*) from areas where pid is null;

-- 2、查询省的名称为“山西省”的所有城市

select \* from areas city inner join areas province on city.pid = province.aid where province.atitle='山西省';

-- 3、查询省的名称为“广州市”的所有县

select \* from areas city inner join areas province on city.pid = province.aid where province.atitle='广州市';

```

### 11.15【记忆，难】子查询

·子查询：在一恶搞selct语句中，嵌入另一个select语句，那么嵌入的这个selsct语句叫做子查询语句，外部那个select语句叫做主查询。

·子查询：

·标量子查询： 子查询返回的是一行一列

·列子查询： 子查询返回的是一列多行

·行子查询： 子查询返回一行多列

·表查询： 子查询返回多行多列

- 子查询

-- 标量子查询: 子查询返回的结果是一个数据(一行一列)

-- 列子查询: 返回的结果是一列(一列多行)

-- 行子查询: 返回的结果是一行(一行多列)

-- 查询出高于平均身高的信息(height)

-- 1 查出平均身高

select avg(height) from students;

-- 2 查出高于平均身高的信息

select \* from students where height > (select avg(height) from students);

-- 查询学生的班级号能够对应的 学生名字

-- select name from students where cls\_id in (select id from classes);

-- 1 查出所有的班级id

select id from classes;

-- 2 查出能够对应上班级号的学生信息

select \* from students where id in (select id from classes);

```

## 12MySQL数据库编程

### 12.1--今日目标

### 12.2【应用】SQL演练

·表子查询：子查询返回一个表

select \* from goods

inner join

(select cate\_name, max(price) as max\_price from goods group by cate\_name) as max\_price\_goods

on goods.cate\_name = max\_price\_goods.cate\_name and goods.price = max\_price\_goods.max\_price;

### 12.3【应用】SQL操作实战

·解决的问题：goods表不再存储分类的名称，改成分类的id

·创建goods\_cates表

·插入数据：

-- 通过子查询插入数据到新的分类表中

insert into goods\_cates(name) (select cate\_name from goods group by cate\_name)

·更新 goods 表 cate\_name 为 goods\_cates的id:

-- 把 商品表 goods 中的 cate\_name 全部替换成 商品分类表 中的 商品id ( update (...)... set )

update goods inner join goods\_cates on goods.cate\_name = goods\_cates.name set goods.cate\_name = goods\_cates.id;

·修改cate\_name 为cate.id

-- 修改表结构 (alter table) 字段名字不同 change,把 cate\_name 改成 cate\_id int unsigned not null

alter table goods change cate\_name cate\_id int unsigned not null;

·创建 goods\_brands表

·操作同上

### 12.4【理解】外键使用

·外键：一个表的主键在另外一个表中出现，在另外一个表称为外键

作用：表间的数据插入，更新的时候的一种约束

·创建外键

·已经存在的表建立外键

Alter table 表名 add foreign key (当前表的字段) references 表明（字段）

·在创建新表的时候建立外键

-- 创建表的同时设置外键 (注意 goods\_cates 和 goods\_brands 两个表必须事先存在)

-- 创建goods\_test表,字段为(id, name, cate\_id,brand\_id )

-- 其中 cate\_id 外键关联 goods\_cates的id字段

-- 其中 brand\_id 外键关联 goods\_brands的id字段

create table goods\_test ( id int primary key auto\_increment, name varchar(150) not null, cate\_id int unsigned not null, brand\_id int unsigned not null,

foreign key (cate\_id) references goods\_cates(id),

foreign key (brand\_id) references goods\_brands(id)

);

·删除外键

·1查看外键名称

Show create table 表名;

·2删除外键

Alter table 表名 drop foreign key 外键名称;

### 12.5【记忆】视图的概念view

视图：虚拟表 仅仅支持查询 把一个复杂的SQL语句封装成一个虚表view

### 12.6【记忆】视图的使用view

·创建视图

cteate view 视图名 as selest ........

视图名命名一般以v\_ 开头

·查询视图

Show tables;

·使用视图

视图只能用来查询？

和使用普通表效果一致

·删除视图

Drop view 视图名

### 12.6【了解】事务概念及特性

·事物：事物transaction，指一个基本工作单元执行的一系列SQL语句的操作，要么完全执行，要么完全不执行

·作用，保证数据操作一战成

四大特征：ACID

1. A原子性，事物的过程不可分割
2. C一致性，双方的接过要一致
3. I 隔离性，同一时间只有一个事物在操作
4. D持久性，操作完成结果持久不变

### 12.7【记忆】事务使用

·步骤

1. 开启事务

Begin;

1. 操作数据库

Insert delete update

1. 确认修改

Commit;

1. 回滚

Rollback;

### 12.8【理解】验证事务的ACID

·commit

客户端A更新数据

客户端B查询

如果A不commit B看到的就是老的数据

·rollback

·验证隔离性：同一时间修改同一个字段

### 12.9【了解】数据库三范式

三范式：

1NF：列的原子性 字段是必须是原子的 不可分割

2NF：满足INF，必须要有主键，非主键字段必须完全依赖于主键，不可部分依赖

3NF：满足2NF,非主键字段必须直接依赖主键。不可传递依赖

### 12.10【了解】数据库设计

·顾客表

·订单表

·订单表详情

主义：

1. 创建外键关联，要引用的表必须先存在
2. 满足三范式

### 12.11【了解】E-R模型及表间关系

表间关系：

1一对一 ： 一个表中的一条数据能和另一个表的唯一一条数据对应（人和常住地址）

2一对多： 一个表中的一条数据能和另一个表的多条数据对应（一个班有多个学生）

3多对多： 一个表中的一条数据能和另一个表的多条数据对应，相反也成立。（学生和课程的关系）

如何保存多对多关系：建立中间表

### 12.12【应用】Python连接Mysql

·作用：使用python充当客户端连接数据库进行操作

步骤：

#1导入模块 import pymysql

#2建立连接对象   cunn = pymysql.connect()

                #参数host 主机

                #user  用户名

                #password  密码

               #port = 3306 默认

 #database   指定数据库

#3建立游标对象 cur = cunn.cursor()

#4使用游标对象执行 = cur.excute(“SQL语句”)

#增删改 影响的行数

#查询，总记录数

#5获取执行的结果

#取一条 cur.fetchone() #元祖

#取所有 cur.fetchall() #大元祖

#6打印输出内容

#for line in result\_list():

Print(line[0])

#7关闭游标对象

Cur.close()

#8关闭连接对象

Conn.close()

### 12.13【应用】python操作数据库CURD

思路：#1导入模块

#2创建连接对象

#3创建游标队形

#4使用游标对象执行SQL cur.execute()

#5提交 conn.commit()

#6获取执行结果（影响的行数）并打印

#7关闭游标

#8关闭连接

### 12.14【应用】SQL防注入

·防注入思路 ：

·在SQL中需要变化的地方，可以用占位符%s%d替换

sql = "select \* from students where name = %s order by id desc"

#SQL中科以出现多个占位符，后续列表元素个数要与之对应

·把参数封装到列表中

input\_name = input("请输入要查询的名称：")

params = [input\_name]

·把列表传递给execute(sql，列表)

result = cur.execute(sql,params)

## 13MySQL高级&装饰器基础

### 13.1【】今日目标

### 13.2【理解】索引

索引的作用：提升查询效率

索引的使用：1查看索引 show index from 表明

2建立索引 create index 索引名 on 表名（表中的字段名（字段长度））

#如果字段是字符串类型，需要制定长度

#如果字段不是字符串类型，可以不指定长度

3删除索引 drop index 索引名 on 表名；

2插入十万条数据

##1目标插入十万条数据

#2导入模块

#创建连接对象

#创建游标对象

#for循环插入十万条数据

#提交

#关闭游标对象

#关闭连接对象

3验证索引效果

1开启检测 set profiling = 1

2执行SQL语句

3查看每个SQL执行时间 show profiles;

4添加索引

5执行sql

6查看执行效率

### 13.3【记忆】用户管理

·创建用户

Create user “用户名”@”主机” identified by “密码”

·授权

权限：create drop select insert updata

所有权限：all privileges;

Grant 权限 on 数据库.表 to “用户名”@”主机名”

·查询权限

Show grans for “用户名”@”主机名”

·刷新权限

Flush privileges

### 13.4【记忆】用户管理2

·修改用户权限

Grant 权限 on 数据库.表 to “用户名”@”主机” with grant option;

·修改用户密码

知道密码修改新密码：

Alter user “用户名”@”主机” identified by “新密码”;

忘记密码修改新密码：

·1停止服务，跳过正常验证，设置密码为空

·2设置正常验证模式，重启服务，用空密码登陆，改成新密码

·删除用户

Drop user “用户名”@”主机”

### 13.5【应用】爬虫实战--爬取数据到数据库

思路："""

#一，定义函数获取列表页的影片信息获取内容页的地址get\_movie\_links()

#1定义列表页地址https://www.ygdy8.net/html/gndy/dyzz/index.html

#2打开列表页，获取数据

#3解码获取到的数据

#4使用正则得到所有影片内容页的地址

#二，定义主函数main()

#定义专门的函数，负责保存数据         add\_film()

            #1定义SQL  准备插入数据

            #2执行insert

            #3commit

#定义专门的函数，检测数据库是否存在相同的函数   film\_exist()

            #1根据影片名称和地址进行查询

            #2执行查询并获取查询的记录数

            #3如果获取到的记录数>0      return  true

            #4如果获取到的记录数 =0      return  false

#创建一个全局连接对象

#创建一个全局游标对象

#关闭游标对象

关闭连接

"""

### 13.6【应用】展示电影数据到网页中

·思路

"""

#1把WEB服务器返回固定内容，代码拷贝day5简单web服务器返回固定内容

#2把原本返回的固定内容改为从数据库动态读取

        #1导入pymysql

        #2连接数据库查询所有影片信息

        #3遍历查询的结果集 ，拼接相应的主体

"""

### 13.7【理解】闭包的概念

在一个外函数中定义了一个内函数，内函数里运用了外函数的临时变量，并且外函数的返回值是内函数的引用，就构成了一个闭包

·闭包构成条件

1. 存在函数的嵌套关系
2. 内层函数引用外城函数的临时变量
3. 外层函数 返回内层函数的引用

### 13.8【了解】闭包中的变量问题

·内层定义了和外层同名的变量

内层会优先使用内层定义的变量，即使定义的变量在内层的最后面

·解决方案：

当内层存在和外层同名变量，而且内层需要使用到外部变量，此时应该使用nonlocal 关键字进行约束

### 13.9【理解】装饰器入门

·装饰器的作用：不修改源代码的基础上，给函数增加新的功能

·装饰器的使用：

1. 存在闭包
2. 存在一个待扩展的函数

·写法：

@闭包的外层函数名

def 待装饰的函数():

Pass

### 13.10【应用】装饰器有参数的函数

·普通参数：

待装饰

@function\_out()

def login(num):

注意：

1. function\_in(num):
2. func(num)

·可变参数：

待装饰

@function\_out()

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

注意：

1. function\_in(\*args,\*\*kwargs):
2. func(\*args,\*\*kwargs)

### 13.11【应用】装饰有返回值的函数

·步骤

·待装饰的函数必须有返回值(return)

·闭包的内层函数 func() 改为return func()

### 13.12【了解】主从配置

## 14Python高级语法专题

### 14.1【今日目标】

### 14.2【理解】在原装饰器上设置外部变量

·作用：向装饰器内部传递参数 又被叫做装饰器工厂模式 flask 和Miniweb会用到

·格式:

def test(path):

    print(path)

    def function\_out(fucc):

        print("------function\_out-------",path)

        def function\_in():

            print("------开始验证------")

            fucc()

        return function\_in

    #test返回装饰器的引用  装饰器工厂模式

    return function\_out

@test("login.py")

#分解为两部

#先执行test("login.py")

#@第一布的结果

#@funciton\_out

#login = funcition\_out(login)

def login():

    print("开始登陆")

#@test("register.py")

#def register():

#    print("-----开始注册-------")

login()

#register()

### 14.3【理解】多重装饰器

·多重装饰器：给一个函数进行多次装饰

·装饰原则：就近原则，靠近待装饰函数的先装饰 随后一层一层装饰

### 14.4【了解】类装饰器

·作用：使用一个类为一个函数装饰

·格式：@类名 分解为@Test login = Test(login)

Login() ---------》去调用类的\_\_call\_\_方法

类的书写：

#1）类的\_\_init\_\_() ，init必须接受 装饰器传递的参数func

#2)类的\_\_call\_\_() , self.func()

待装饰的函数

·对象名() 调用对象的”\_\_call\_\_()”方法

class Test(object):

    def \_\_init\_\_(self,func):

        print("这是\_\_init\_\_方法")

        print("---------------func------",func)

        #self.func就是这个func的引用

        self.func = func

    def run(self):

        print("这个值正在疯跑")

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

        print("------call---------")

        self.func()

@Test

def login():

    print("开始登陆")

login()

### 14.5【了解】GIL锁

·监控资源的竞争情况htop

·资源的消耗情况

·单进程 1核

·多进程 多核

·多线程 多核但是很低 底层原因GIL锁问题

### 14.6【了解】GIL锁概念及影响

·GIL：全局解释器锁

·GIL跟python语言没有关系，Python底层解释器CPython的问题

·GIL锁释放情况：

1. 线程执行完毕释放
2. 遇到IO阻塞释放
3. Python3.0x使用计时器（执行时间到达阀值后，当前线程释放GIL）

### 14.7【理解】GIL锁解决方案

·GIL锁称为：“全局解释器锁”，是Cpython解释器的锁机制，也是历史遗留问题

·要提升多线程执行效率，解决方案：

1. 更换解释器 jpython pypython
2. 改为进程替换多线程
3. 子线程使用C语言实现

·必须知道的是：

·CPU密集型不太适合多线程

·I/O密集型适合多线程（GIL锁会释放）

### 14.8【理解】Python中可变和不可变

·可变：（mutable），创建后可以继续修改对象的内容（值）

字典，列表

·不可变：（unmutable），一旦创建就不可修改的对象（值）

数字，字符串，元祖

当内容发生修改，计算机重新分配一块内存空间

### 14.9【理解】简单可变类型拷贝

浅拷贝：

·产生新的空间

·源对象和副本对象指向同一个空间

·如果拷贝的是对象，子对象不会拷贝（不会产生新的空间）

深拷贝：

·会产生新的空间

·能够保持各自的独立性

·如果拷贝的是对象，子对象会拷贝（子对象也会产生新的空间）

·简单可变类型的拷贝总结：

简单可变类型的数据。不管是深浅拷贝，都会产生新的空间，而且保持自己的独立性

### 14.10【理解】复杂 可变类型的深浅拷贝

·复杂可变类型的总结：

·浅拷贝：拷贝的是顶层对象的值，子对象不会被拷贝（只是引用子对象）

·深拷贝：拷贝的是顶层对象和子对象，子对象会产生新的内存空间

### 14.11【理解】简单不可变类型拷贝

·简单不可变类型

·copy() 浅拷贝 ，副本和源对象相同，指向同一个空间

·deepcopy() 浅拷贝，副本和源对象相同，指向同一个空间

### 14.12【理解】复杂不可变类型拷贝

·复杂不可变数据类型深浅拷贝问题

·浅拷贝：直接引用

·深拷贝：看数据类型具体是可变的还是不可变的，若果数据时可变的，会产生新的空间，保持数据独立性。

### 14.13【理解】切片拷贝，字典拷贝

切片拷贝：浅拷贝，如果是简单可变类型，底层会产生新的空间

字典拷贝：浅拷贝

## 15Python语法高级专题

### 15.1【】今日目标

### 15.2【理解】import导入模块路径

·存在问题：当我们把模块文件放到工程文件夹的外部，发现无法正常引入

·原因：外部的文件夹的路径，没有在系统环境变量中

·解决方案：

1. 导入sys sys.path可以查看环境变量 返回值是一个列表
2. 把自己写的模块的路径 加入到这个列表里去

·方法sys.path.append(路径) 追加到末尾

·方法sys.path.insert(0,路径) 追加到指定位置

### 15.3【理解】import的reload加载问题

·存在的问题：import导入模块后，如果模块被修改，此时再次import不起作用

·import自动防止重复包含

·强制重新加载一次模块

·reload()模块

·使用方法：

·1)from imp import reload

·1)reload(要重新加载的模块)

### 15.4【理解】from .....import私有化问题

·私有化：模块中的一些变量不希望被其他模块导入，可以使用私有化解决

·私有化使用前提：必须使用from ... import \*

·在模块中，给变量前增加一个下划线”\_”

·这种私有化模式非常局限,如果使用其他模式导入，私有化将无效

·

### 15.5【记忆】import 和from...import的区别

·区别：

·写法的区别

·1）import 写法 模块名.变量/函数，类

·2）from ... import \* 写法 变量名/函数名

·底层的区别

·import 直接引用了源模块的变量/函数/类

·from .... import \* 拷贝源模块的 变量/函数/类 不一样

### 15.6【理解】可变参数的拆包问题

·可变参数传递 \*args \*\*kwargs 默认会封包过程

·如果想要这种参数继续传递下一个参数，传递的时候func(\*args,\*\*kwargs)

### 15.7【理解】单继承中的super（）

Surper()使用的时候,传递参数的时候，self不用传递

Surper()调用顺序，按照mro顺序来完成

Grandson.\_\_mro\_\_是一个元祖

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

### 15.8【理解】多继承和mro顺序

多继承中surper()执行顺序严格执行mro顺序表

Mro顺序表：

·类名.mro()

·类名.\_\_mro\_\_

在多继承中，当在类中使用surper()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，不建议使用类名调用父类德尔方法，会出现父类被调用两次的问题

### 15.9【记忆】@property装饰方法

·@property特点：让我们通过 对象.方法名 的方式去调用方法

·语法格式

@property

def test(self):

pass

·@property装饰方法，只能有一个参数self

### 15.10【记忆】@property其他使用方式

·经典类：@property一种方式

·新式类：

·@property

goods.price 获取价格的方法

[·@XXX.setter](mailto:·@XXX.setter)

goods.price = XXX

[·@XXX.deleter](mailto:·@XXX.deleter)

del.goods.price [----》@XXXX.delete](mailto:----》@XXXX.delete) 装饰的方法

### 15.11【了解】property作为类属性

·定义property 对象的类属性

xxx = property(参数一，参数二，参数三，参数四)

        #property 对象

        #property （第一个参数，第二个参数，第三个参数，第四个参数）

        #第一个参数，当我们foo.BAR，自动调用第一个参数的方法

        #第二个参数，当我们foo.BAR = 100,会自动调用第二个参数的方法

        #第三个参数，当我们del.foo.BAR,会自动调用第三个参数的方法

        #第四个参数，当我们Foo.BAR.\_\_doc\_\_,会自动获取第四个参数的内容 一个字符串

·使用：

·对象.XXX 获取值

·对象.xxx = 100 设置值

·del 对象.xxx 调用第三个参数方法

·类.xxx.\_\_doc\_\_ 获取第四个参数的内容

### 15.12【记忆】魔法属性及方法

·魔术属性

·\_\_doc\_\_ 获取魔术信息

·获取类的注释信息 类名.\_\_doc\_\_

·索取方法的描述信息 对象名.方法.\_\_doc\_\_

·\_\_module\_\_ 获取所属模块 直接运行该文件 获取的是\_\_main\_\_

·对象名.\_\_module\_\_

·\_\_class\_\_ 获取对象所属的类

·对象名.\_\_class\_\_

·魔术方法

·\_\_init\_\_ 初始化方法

·类名() 直接调用

·\_\_del\_\_ 删除对象的时候会调用

·del 对象名 直接调用

### 15.13【记忆】魔术属性及方法2

·魔术属性 \_\_dict\_\_ 获取对象获取类的信息

·获取对象信息 对象名.\_\_dict\_\_ 获取对象的实例属性的信息

·获取类信息 类名.\_\_dict\_\_ 获取类的模块，对象方法，类的描述......

·魔术方法

·\_\_call\_\_() 当使用对象名加括号调用\_\_call\_\_方法

·\_\_str\_\_() 当打印对象的时候会调用 print(obj) \_\_str\_\_方法一定要return字符串

·用字典的格式操作对象的方法

·\_\_getitem\_\_(item) 对象[“xxx”]

·\_\_setitem\_\_(key,item) 对象[“XXX”] = XXX

·\_\_delitem\_\_(key) del 对象[“XXX”]

### 15.14【理解】with管理上下文方式一

·上下文：以with open 来说，打开文件在上文，关闭文件在下文

·上下文管理器：

·\_\_enter\_\_ 上文方式

·\_\_exit\_\_ 下文方式

·自定义一个满足上下文管理器的类

class MyFile(object):

    def \_\_enter\_\_(self):

        print("进入上文")

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

        return self.file

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

        print("进入下文")

        self.file.close()

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

        self.file\_name = file\_name

        self.file\_model = file\_model

if \_\_name\_\_ == "\_\_main\_\_":

    with MyFile("G:/GitPro/pygame/LINUX系统/linux实战/15.0/1.txt","r") as file:

        file\_data = file.read()

        print(file\_data)

### 15.15【理解】with管理上下文方式二

·通过装饰器@contextmanger实现上下文管理

·装饰器

·待装饰的函数

myopen() 分拆成上文和下文 ，使用yield分拆

#装饰器不修改源代码   增加新的功能

from contextlib import contextmanager

#装饰器装饰函数的步骤

@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("G:/GitPro/pygame/LINUX系统/linux实战/15.0/1.txt","r") as file:

    file\_data = file.read()

    print(file\_data)

·装饰的过程

1导入模块

2进行装饰