My_Note

操作系统

操作系统直接运行在计算机上的系统软件，同时与软硬件打交道

Linux

- /: 根目录
- /bin: 可执行二进制文件的目录
- /etc: 系统配置文件存放的目录、
- /home: 用户家目录

Linux内核是操作系统控制**E**硬件的核心，且常见的**Linux**操作系统都是基于**Linux**内核开发的

Linux命令

基本命令

查看目录命令

命令	说明
`ls`	查看当前路径下目录信息
`tree`	以树状方式显示目录
`pwd`	查看当前目录路径
`clear`	清楚终端内容
`ctrl+shift+"+"`	放大窗口字体
`ctrl+"-"`	缩小窗口字体

切换目录命令：目录名后面要加**/**

```
## 切换到桌面
cd Desktop/
```

cd的命令和说明

命令	说明
cd目录	切换到指定目录
cd~	切换到当前用户的主目录
`cd..`	切换到上一级目录
`cd.`	切换到当前目录
`cd-`	切换到上一次目录

创建和删除文件

命令	说明
torch 文件名	创建指定文件
mkdir 目录名	创建目录(文件夹)
rm 目录名	删除指定文件
rmdir 目录名	删除空目录

== `rm` 也可以删除目录，但需要在命令后加-r==

复制、移动文件

命令	说明
cp	复制(拷贝)文件，拷贝目录
mv	移动文件、移动目录、重命名

```
## 复制hello并命名为hello1
cp hello hello1
## 对目录所有东西进行拷贝
cp a a_cp -r
## 把当前目录的hello文件移动到a目录下
mv ./hello ./a
## 把hello文件重命名为hi
mv hello hi
```

`ls` 命令选项

命令选项	说明
-l	以列表方式显示，默认是字节
-h	智能的显示文件大小
-a	显示隐藏文件和隐藏目录

`mkdir` 命令选项

命令选项	说明
-p	创建所依赖的文件夹(多级目录)

```
## 创建多级目录
mkdir aa/bb/cc -p
```

`rm` 命令选项

命令选项	说明
-i	交互式提示(删除前会给你提示)

-r	递归删除目录及其内容
-f	强制删除

`cp` 命令选项

命令选项	说明
-i	交互式提示
-r	递归拷贝目录及其内容
-v	显示拷贝后的路径描述

`mv` 命令选项

命令选项	说明
-i	交互式提示
-v	显示移动后的路径描述

```
## 复制目录hello为hello1
mv hello/ hello1 -i
```

重定向命令

(把终端执行命令的结果保存到目标文件**)**

命令	说明
>	如果文件存在，会覆盖原有文件内容，相当于文件□作中的'w'模式
>>	如果文件存在，会追加写入文件末尾，相当于文件□作中的'a'模式

```
## 创建文件
touch a.txt
## 写入文件
ls > a.txt
## 追加
ls >> a.txt
```

查看文件内容命令

命令	说明
cat	查看小型文件
more	查看大型文件
|	管道，一个命令的输出可以通过管道做为另一个命令的输入，相当于一个容器

more命令使用

□作键	说明
回车	显示下一行信息
b	显示上一屏信息
f	显示下一屏信息
q	退出

```
touch a.txt
## 查看文件内容
cat a.txt
## 查看大型文件内容
more huge.txt
## 管道命令，以more命令输出tree
tree /bin | more
```

软链接

(方便文件的访问□作)

命令	说明
`ln -s`	创建软链接

```
## 创建目录
mkdir A/B/C -p
## 进入C
cd A/B/C
## 创建文件
touch hello.py
## 回到桌面
cd Desktop/
## 创建软链接
## 要注意的是相对路劲下的软链接不能随意移动，否则会损坏
ln -s ./A/B/C/hello.py hello_s1.py
ln -s /home/python/Desktop/A/B/C/hello.py hello_s2.py
```

搜索文件内容命令

命令	说明	选项	说明
		-v	显示不包含匹配文本的所有行
`grep`	查找/搜索文件内容	-n	显示匹配行号
		-i	忽略大小写

```
## 在a.txt里面查找hello
## 显示匹配行号
```

```
grep hello a.txt -n
```

查找文件命令

命令	说明
`find`	根据文件名查找文件

通配符	说明
*	代表0个或多个任意字符
?	代表任意一个字符

```
## 查找
find . -name "2.txt"
## 模糊查找
find . -name "2*"
```

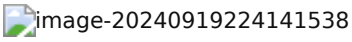
压缩和解压缩命令

压缩格式	说明
`.gz`	压缩包后缀
`.bz2`	压缩包后缀

==说明：需要用tar命令来压缩和解压缩==

tar命令选项	说明
`-c`	创建打包文件
`-v`	显示解包或打包的详细信息
`-f`	指定文件名称，必须放到所有选项后面
`-z`	压缩(`.gz`)
`-j`	压缩(`.bz2`)
`-x`	解压缩
`-C`	解压缩到指定目录

```
## 打包文件，并不会压缩
tar -cvf 1.tar *.txt
## 压缩文件并打包
tar -zcvf 1.tar.gz *.txt
## 解压缩
tar -xvf 1.tar.gz
```



文件权限命令

image-20240919224141538

文件类型说明

- -表示普通文件
- d表示文件夹

文件权限说明

- 由三个三元组字符组成
- 第一个三元字符组代表文件所有者用户的权限
- 第二个代表文件用户组的权限
- 第三个代表其他用户的权限
- r表示可读，w表示可写
- x表示可执行，-表示没有权限

==root用户不受上述限制==

修改文件权限

命令	说明
chmod	修改文件权限

chmod u/g/o/a+/-/=rwx 文件名

角色	说明
u	user代表该文件的所有者
g	gruop代表用户组
o	other代表其他用户
a	all代表所有用户

操作符

操作符	说明
+	增加权限
-	撤销权限
=	设置权限

权限说明

权限	说明
r	可读
w	可写
x	可执行

-	无任何权限

```
## 新建文件
torch a.py
## 修改文件权限
chmod u + rwx a.py
## 让所有用户不能阅读此文件
chmod a - r a.py
```

获取管理员权限

sudo命令的使用

命令	说明
sudo -s	切换到root用户，获取管理员权限
sudo	某个命令的执行需要获取管理员权限可以在执行命令前面加上sudo

```
## □换到root用户
sudo -s
## 使用sudo执行命令
sudo cat a.py
```

whoami命令的使用

命令	说明
whoami	查看当前用户
who	查看所有用户

exit命令使用

命令	说明
exit	退出登录用户

which命令使用

命令	说明
which	查看命令位置

passwd使用

命令	说明
passwd	修改用户密码，不指定用户默认修改当前登录用户密码

关机和重启命令使用

命令	说明
shutdown -h now	立刻关机

reboot	重启

软件安装

两种方法

安装方式	说明	命令
离线安装	deb文件格式安装	sudo dpkg -i deb 安装包
在线安装	apt-get方式安装	sudo apt-get install 安装包

软件卸载

两种方法

卸载方式	说明	命令
离线安装包卸载	deb文件格式卸载	sudo dpkg -r 安装包名
在线安装包卸载	apt-get方式卸载	sudo apt-get remove 安装包名

vim介绍

工作模式：

- 命令模式：通过i进入编辑模式，用：进入末行模式
- 编辑模式：编写代码，用esc退出到命令模式
- 末行模式：进行代码保存等操作，用esc推出到命令模式
    - :w：保存
    - :wq：保存退出
    - :x：保存退出
    - :q!：强制退出

==注意==：vim打开文件进入的是命令模式

vim的常用命令

多任务

指的是在同一时间内执行多个任务

- 并发：在一段时间内去交替执行多个任务，即让各个任务轮流执行，但轮流执行的速度过快，表面上看我们会感觉这些软件在同时执行
    - 任务数量大于CPU的核心数
- 并行：在一段时间内真正的同时一起执行多个任务，给CPU的每个内核安排一个执行的任务
    - 任务数量小于或等于CPU的核心数

进程

进程是操作系统进行资源分配和调度运行的基本单位,通俗解释：一个正在运行的程序就是一个和进程

多进程的作用

程序运行会默认创建一个进程，为主进程

程序运行之后又创建了一个进程，这个新创建的进程称为子进程

多进程完成多任务

进程的创建步骤

1. 导入进程包
    - `import multiprocessing`
2. 通过进程类创建进程对象
    - `进程对象=multiprocessing.Process()`
    -

参数名	说明
target	执行的目标任务名，这里指的是函数名(方法名)
name	进程名，不用设置
group	进程组，目前使用None

3. 启动进程执行任务
    - `进程对象.start()`

代码实现

```python
import multiprocessing
import time
def coding():
    for i in range(3):
        print('coding')
        time.sleep(0.2)


def music():
    for i in range(3):
        print('music')
        time.sleep(0.2)

if __name__ = '__main__':
    # 通过进程类创建进程对象
    coding_process = multiprocessing.Process(target = coding)
    music_process = multiprocessing.Process(target = music)
    coding_process.start()
    music_process.start()
```

进程执行带有参数的任务

参数名	说明
args	以元组的方式给执行任务传参
kwargs	以字典方式给执行任务传参

1. 元组方式传参：元组方式传参一定要和参数的顺序保持一致
2. 字典方式传参：字典方式传参字典中的key一定要和参数名保持一致

```
import multiprocessing
import time
def coding(num,str):
    print(str)
    for i in range(3):
        print('coding')
        time.sleep(0.2)
def music(count):
    for i in range(3):
        print('music')
        time.sleep(0.2)
if __name__ = '__main__':
    # 通过进程类创建进程对象
    # 元组形式传参
    coding_process = multiprocessing.Process(target = coding,args=(3,'你好'))
    # 字典形式传参
    music_process = multiprocessing.Process(target = music,kwargs={'count':3})
    coding_process.start()
    music_process.start()
```

获取进程编号

==作用==：根据进程编号来区分不同的进程

1. 获取当前进程编号
    - **`getpid()方法`**
2. 获取当前父进程编号

- **`getppid()方法`**

==需要导入os模块！！==

```
import multiprocessing
import os
import time
def coding():
    print("coding>>>%d" % os.getpid())
    print("父进程编号为%d" % os.getppid())
    for i in range(3):
        print('coding')
        time.sleep(0.2)


def music():
    print("music>>>%d" % os.getpid())
    print("父进程编号为%d" % os.getppid())
    for i in range(3):
        print('music')
```

```
        time.sleep(0.2)

if __name__ = '__main__':
    print(os.getpid())
    # 通过进程类创建进程对象
    coding_process = multiprocessing.Process(target = coding)
    music_process = multiprocessing.Process(target = music)
    coding_process.start()
    music_process.start()
```

进程间不共享全局变量

实际上创建一个子进程就是把主进程的资源进行拷贝产生了一个新的进程，两个进程里面有相同的全局变量名，但是地址却是不一样的

```
import multiprocessing
import os
import time
my_list = []
def write():
    for i in range(3):
        my_list.append(o)
        print('add:',i)
    print(my_list)


def read():
    print(my_list)

if __name__ = '__main__':
    write_process = multiprocessing.Process(target = write)
    read_process = multiprocessing.Process(target = read)
    write_process.start()
    read_process.start()
```

主进程和子进程的结束顺序

为了保证子进程的正常运行，主进程会等待所有的子进程执行完成以后销毁，可以设置守护主进程来保证主进程结束后子进程才销毁

- 设置守护主进程：`子进程对象.deamon = True`
- 销毁子进程：`子进程对象.terminate()`

设置守护主进程

```
import multiprocessing
import time
def work():
    print('工作中')



if __name__ = '__main__':
    # 通过进程类创建进程对象
```

```
    work_process = multiprocessing.Process(target = work)
    # 设置守护主进程
    work_process.daemon = True
    work_process.start()
    print('主进程执行完成')
```

销毁主进程

```
import multiprocessing
import time
def work():
    print('工作中')



if __name__ = '__main__':
    # 通过进程类创建进程对象
    work_process = multiprocessing.Process(target = work)
    work_process.start()
    # 销毁主进程
    work_process.terminate()
    print('主进程执行完成')
```

线程

1. 进程是==分配资源的最小单位==，线程是==程序执行的最小单位==

2. 线程自己不占用系统资源，其可同属一个进程并==共享进程的所有资源==

多线程完成多任务

1. 导入线程模块
   - `import threading`
2. 通过线程类创建线程对象
   - `线程对象 = threading.Thread(target = 任务名)`
   -

参数名	说明
target	执行的目标任务名，这里指的是函数名(方法名)
name	线程名，一般不用设置
group	线程组，目前使用None

3. 启动线程执行任务

- 线程对象.**start()**

```
import threading
import time
def coding():
    for i in range(3):
        print('coding')
        time.sleep(0.2)


def music():
    for i in range(3):
        print('music')
        time.sleep(0.2)

if __name__ = '__main__':
    # 通过进程类创建进程对象
    coding_thread = threading.Thread(target = coding)
    music_thread = threading.Thread(target = music)
    coding_thread.start()
    music_thread.start()
```

线程执行带有参数的任务

参数名	说明
args	以元组的方式给执行任务传参
kwargs	以字典方式给执行任务传参

1. 元组方式传参：元组方式传参一定要和参数的顺序保持一致
2. 字典方式传参：字典方式传参字典中的key一定要和参数名保持一致

```
import threading
import time
def coding(num):
    for i in range(num):
        print('coding')
        time.sleep(0.2)


def music(count):
    for i in range(count):
        print('music')
        time.sleep(0.2)

if __name__ = '__main__':
    # 通过进程类创建进程对象
    coding_thread = threading.Thread(target = coding,args = (3,))
    music_thread = threading.Thread(target = music,kwargs={'count':3})
```

```
    coding_thread.start()
    music_thread.start()
```

主线程和子线程的结束顺序

主线程会等待所有的子线程执行结束后再结束

设置守护主线程

1. `threading_Thread(target = work,daemon = True)`
2. 线程对象**.setDaemon(True)**

```
import threading
import time
def work():
    print('工作中')



if __name__ = '__main__':
    # 设置守护主线程
    work_thread = threading_Thread(target = work,daemon = True)
    # 设置守护主线程
    work_thread.setDaemon(True)
    work_thread.start()

    print('主线程执行完成')
```

线程间的执行顺序

==线程间的执行是无序的==

获取当前线程信息: `current = threading.current_thread()`

线程间共享全局变量

```
import thereading
import time

my_list = list()

def write():
    for i in range(3):
        print('add:',i)
        my_list.append(i)
    print(my_list)


def read():
    print('read:',my_list)

if __name__ == '__main__':
    write_thread = threading.Thread(target = write)
```

```
    read_thread = threading.Thread(target = read)
    write_thread.start()
    time.sleep(1)
    read_thread.start()
```

线程间资源竞争问题

线程间的资源竞争（或竞争条件）是指多个线程在访问共享资源时，因执行顺序的不确定性而导致的错误

==解决方法：互斥锁==

互斥锁的使用

互斥锁:对共享数据进行锁定，保证同一时刻只有一个线程去操作

==注意：==互斥锁是多个线程一起去抢，抢到锁的线程先执行，没有抢到的线程进行等待，等锁使用完释放后，其它等待的进程再去抢这个锁

1. 互斥锁的创建
    - `mutex = threading.Lock()`
2. 上锁
    - `mutex.acquire()`
3. 释放锁
    - `mutex.release()`

```
import threading

g_num = 0

def sum_1():
    # 上锁
    mutex.acquire()
    for i in range(100000):
        # 声明全局变量
        global g_num
        g_num += 1
    # 解锁
    mutex.release()
    print('g_num1:',g_num)



def sum_2():
    # 上锁
    mutex.acquire()
    for i in range(100000):
        # 声明全局变量
        global g_num
        g_num += 1
    # 解锁
    mutex.release()
    print('g_num2:',g_num)
```

```
if __name__ == '__main__':
    # 创建锁
    mutex = threading.Lock()
    sum1_thread = threading.Thread(target = sum_1)
    sum2_thread = threading.Thread(target = sum_2)


    sum1_thread.start()
    sum2_thread .start()
```

死锁

一直等待对方释放锁的场景就是死锁

死锁的结果:

会造成应用程序的停止响应，不能执行其他任务

阻塞

使用 `.join()` 方法，会使线程状态进入阻塞，在主线程中调用子线程**.join()**，主线程会等待子线程执行完后再执行

```
import time
import threading

def work():
    print('work')
    time.sleep(1)

work_thread = threading.Thread(target = work)
work_thread.start()
work_thread.join()

print('主线程执行结束')
```

进程和线程

- 关系
    1. 线程是依附在进程里面的，没有进程就没有线程
    2. 一个进程默认提供一条线程，进程可以创建多个线程
- 区别
    1. 进程之间不共享全局变量
    2. 线程之间共享全局变量，但要注意资源竞争
    3. 创建进程的资源开销更大
    4. 进程是□作系统资源分配的基本单位，线程是CPU调度的基本单位
    5. 线程不能够独立执行，必须依附在进程中

网络

实现资源共享和信息传递的虚拟平台，**socket**和**web**开发等等有涉及

IP地址

IP地址是分配给==网络设备上网使用的字符标签==，它能够==标识网络中唯一的一台设备==

通过**IP**地址找到网络中唯一一台设备，然后与其通信

ifconfig和ping命令

命令名	说明
ifconfig	查看网卡信息
ping	检查网络是否正常

```
## 在Linux中查看
## 会出现两个ip地址，一个是本机的(即local_post)，一个是在网络上的
ifconfig
## 查看本机和百度的网络是否畅通
ping baidu.com
```

端口和端口号

==只有IP地址是无法确定把数据给到哪个进程的！！！==

每运行一个程序都会有一个端口**(**传输数据的通道**)**，想要给对应的程序发送数据，找到对应的端口即可

- 端口是传输数据通道，是数据传输的必经之路
- 端口号是用来管理区分不同端口的一个号码

端口号的分类

- 知名端口号
  - 指的是众所周知的端口号，范围从==0到1023==，21端口号分配个FTP(文件传输协议)，25端口分配个SMTP(简单邮件传输协议)服务，80端口分配给HTTP服务
- 动态端口号
  - 范围从==1024到65535==，在开发的时候可以在这个范围内设置端口号
  - 提示**:**当运行一个程序默认会有一个端口号，程序退出时端口号被释放

socket和TCP

- socket是程序之间进行数据交换的接口，通过socket，开发者可以实现客户端和服务器之间的网络通信
- TCP是一种传输控制传输协议，它是一种 面向连接的，可靠的，基于字节流的传输层通信协议
  - **TCP**通信步骤
    1. 创建连接
    2. 传输数据
    3. 关闭连接

编码转换

函数名	说明
encode	编码，将字符串转换为字节码
decode	解码，将字节码转换为字符串

提示：

`encode()` 和 `decode()` 可以接受参数，encoding指的是在编解码过程中使用的编码方案

- `bytes.decode(encoding = 'utf-8')`
- `str.encode(encoding = 'utf-8')`

TCP服务端

- socket服务端编程 1. 创建socket对象 2. 绑定socket对象到指定ip和地址 3. 服务端开始监听端口 4. 接收客户端连接，获得连接对象 5. 客户端连接后，通过recv方法，接收客户端发送的消息 6. 通过conn(客户端当前连接对象)，调用send方法可以回复消息 7. conn和socket_server调用close方法关闭连接

==监听：==是网络通信的核心机制之一，使得服务器可以在特定端口和ip等待客户端理解，并且能和多个客户端建立连接

服务端**socket**类的参数和方法说明：

参数名	说明
AddressFamily	IP地址类型，分为
Type	传输协议类型

开发服务端用到的函数

方法名	说明
bind	绑定ip地址和端口号
send	发送数据
accept	等到接受客户端的连接请求
recv	接收数据
listen	设置监听，参数为可排队等待连接的最大数量

```
import socket
## 1.建立服务端套接字 只负责监听
socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
## 设置端口复用
socket.setsocket(socket.SOL_SOCKET,socket.SO_REUSEADDR,True)
## 2.绑定ip地址和端口号
## 如果bind中的ip地址为"",表示其使用的是本机的ip地址
socket.bind(('IP地址',端口号))
## 3.设置监听
socket.listen(128)
## 4.等待客户端的连接请求 accept会阻塞等待 返回一个用以和客户端通讯的socket，客户端的地址
conn_socket ,ip = socket.accept()
print('客户端地址：',ip)
## 5.接收数据 用的是和客户端通讯的socket
data = conn_socket.recv()
print(data.decode())
## 6.发送数据
conn_socket.send('你好吗').encode(encoding = 'utf-8')
## 7.关闭套接字
conn_socket.close()
socket.close()
```

TCP客户端

- 客户端编程
    1. 创建socket对象
    2. 连接到服务端
    3. 发送消息
    4. 接受返回消息
    5. 关闭连接

客户端**socket**类的参数和方法说明:

参数名	说明
AddressFamily	IP地址类型，分为
Type	传输协议类型

开发客户端用到的函数

方法名	说明
connect	和服务端套接字建立连接
send	发送数据
recv	接收数据，参数为接收的字节数
close	关闭连接

```
import socket
## 1.创建客户端套接字
client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
## 2.连接服务端
client.connect(('ip地址','端口号'))
## 3.发送数据
clent.send('你好吗').encode(encoding = 'utf-8')
## 4.接收数据,recv会阻塞，直到数据到来
data = client.recv(1024)
print(data.decode())
## 5.关闭客户端连接
client.close()
```

相关注意点

1. 当TCP客户端程序想要和TCP服务端程序进行通信时必须要先建立连接
2. **TCP**服务端必须绑定端口号，否则客户端找不到这个TCP服务端程序
3. listen后的套接字是被动套接字，只负责接受新的客户端的连接请求，不能收发消息
4. 当客户端和服务端连接成功后，服务端会产生一个==新的套接字==用于和客户端通讯
5. 关闭accept套接字意味着和这个客户端通信完毕
6. 当客户端套接字调用**close**后，服务端的**recv**会解阻塞，返回的数据长度为**0**，服务端可以根据返回数据的长度来判断客户端是否已经下线。客户端也是同理的。

网址

称**url**，又称统一资源定位符

URL的组成

- 协议部分: https:// 、http://、ftp://
- 域名: IP地址的别名，它是用点进行分割使用英文字母和数字组成的名字，DNS可以将这个别名解析成IP地址
- 资源路径部分: 该网页在服务器的路径

HTTP协议的介绍

- 通过**HTTP**协议来规定浏览器和**web**服务器之间通讯的数据格式，称为超文本传输协议
- 传输**HTTP**协议格式的数据是基于**TCP**传输协议的，发送数据之前要先建立连接
- **TCP**传输协议是用来保证网络中传输的数据的安全性的**,HTTP**协议是用来规定这些数据的具体格式的

HTTP的请求报文

- GET方式请求的报文: 获取Web服务器数据，==携带的参数会在url显示出来！！！（不安全）==
- POST方式请求的报文: 向Web服务器提交数据

说明:

- POST请求方式有由带参数的请求体构成，在向浏览器发送POST请求的时候需要携带参数和对应的value

HTTP的响应报文

- 响应行: HTTP协议版本，状态码等
- 响应头: 服务器名称等
- 空行
- 响应体: 响应给客户端的数据

HTTP状态码

状态码	说明
200	服务器成功处理请求
400	错误的请求，请求地址和参数有误
404	请求资源不存在
500	服务器内部代码出现错误

搭建python自带的静态Web服务器

使用pycharm搭建

```
import http
```

开发自己的静态Web服务器

开发步骤:

1. 编写一个TCP服务端程序
2. 获取浏览器发送的HTTP请求报文数据
3. 读取固定页面数据，把页面数据组装成HTTP响应报文数据发送给浏览器
4. HTTP响应报文数据发送完后，关闭服务于客户端的套接字

```python
import socket
socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
## 设置端口复用
socket.setsocket(socket.SOL_SOCKET,socket.SO_REUSEADDR,True)
## 绑定端口和ip
socket.bind(("",8080))
## 设置监听
socket.listen(128)
## 接收返回参数
conn,ip = socket.accept()
## 接收请求
## data里面有用户的请求信息，其中包含了请求的文件，我们可以根据这个信息返回指定页面
data = conn.recv(1024).decode()
request_data = data.spilt(" ")
request_path = request_data[1]
## 判断是否访问主页，直接访问ip会让request_path变为'/'
if request_path == '/':
    request_path = '/index.html'
## 打开文件并组成响应报文返回
try:
    with open ("/static" + request_path,"rb") as f:
        file_data = f.read()
except Exception as e:
    # 返回404错误数据
    response_line = "HTTP/1.1 404 Not Found\r\n"
    response_header = "Server:pwb\r\n"
    response_body = "404 Not Found HAHA"
    response = (response_line + response_header + response_body).encode()
    conn.send(response)
else:
    response_line = "HTTP/1.1 200 OK \r\n"
    response_header = "Server:pwb\r\n"
    response_body = file_data
    # 转换成二进制数据便于HTTP传输
    response = (response_line + response_header).encode() + response_body
    conn.send(response)
finally:
    # 关闭套接字
    conn.close()
```

一个简单的异常捕获代码

```python
def divide_numbers(a, b):
    try:
        result = a / b  # 可能会引发 ZeroDivisionError
    except ZeroDivisionError:
        print("错误：不能将数字除以零！")
    except TypeError:
        print("错误：输入的类型不正确！")
    else:
        print(f"结果是：{result}")  # 如果没有异常，打印结果
```

```
    finally:
        print("执行结束。")  # 无论如何都会执行

## 示例调用
divide_numbers(10, 2)  # 正常情况
divide_numbers(10, 0)  # 除以零的情况
divide_numbers(10, "a")  # 类型错误的情况
```

多任务版本

使用多线程

- 当客户端和服务端连接成功，创建子线程，使用子线程专门处理客户端的请求

```
import socket
import threading
def handle(conn):
    # 接收请求
    # data里面有用户的请求信息，其中包含了请求的文件，我们可以根据这个信息返回指定页面
    data = conn.recv(1024).decode()
    request_data = data.spilt(" ")
    request_path = request_data[1]
    # 关闭客户端时客户端会返回一个空字符给服务端，这时候服务端无法解析，需要在这里进行判断
    if len(request_path) == 1:
        conn.close()
        return
    # 判断是否访问主页，直接访问ip会让request_path变为'/'
    if request_path == '/':
        request_path = '/index.html'
    # 打开文件并组成响应报文返回
    try:
        with open ("/static" + request_path,"rb") as f:
            file_data = f.read()
    except Exception as e:
        # 返回404错误数据
        response_line = "HTTP/1.1 404 Not Found\r\n"
        response_header = "Server:pwb\r\n"
        response_body = "404 Not Found HAHA"
        response = (response_line + response_header + response_body).encode()
        conn.send(response)
    else:
        response_line = "HTTP/1.1 200 OK \r\n"
        response_header = "Server:pwb\r\n"
        response_body = file_data
        # 转换成二进制数据便于HTTP传输
        response = (response_line + response_header).encode() + response_body
        conn.send(response)
    finally:
        # 关闭套接字
        conn.close()

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

```
## 设置端口复用
socket.setsocket(socket.SOL_SOCKET,socket.SO_REUSEADDR,True)
## 绑定端口和ip
socket.bind(("",8080))
## 设置监听
socket.listen(128)
while True:
    # 接收返回参数
    conn,ip = socket.accept()
    sub_thread = threading.Thread(target = handle,args=(conn,))
    sub_thread.start()
```

面向对象开发

把□作抽象到一个类中

```
import socket
import threading

class HttpWebServer():
    def __init__(self):
        self.socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        # 设置端口复用
        self.socket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,True)
        # 绑定端口和ip
        self.socket.bind(("",8080))
        # 设置监听
        self.socket.listen(128)
    def handle(self,conn):
        # 接收请求
        # data里面有用户的请求信息，其中包含了请求的文件，我们可以根据这个信息返回指定页面
        data = conn.recv(1024).decode()
        request_data = data.spilt(" ")
        request_path = request_data[1]
        # 关闭客户端时客户端会返回一个空字符给服务端，这时候服务端无法解析，需要在这里进行判断
        if len(request_path) == 1:
            conn.close()
            return
        # 判断是否访问主页,直接访问ip会让request_path变为'/'
        if request_path == '/':
            request_path = '/index.html'
        # 打开文件并组成响应报文返回
        try:
            with open ("/static" + request_path,"rb") as f:
                file_data = f.read()
        except Exception as e:
            # 返回404错误数据
            response_line = "HTTP/1.1 404 Not Found\r\n"
            response_header = "Server:pwb\r\n"
            response_body = "404 Not Found HAHA"
            response = (response_line + response_header + response_body).encode()
            conn.send(response)
```

```
        else:
            response_line = "HTTP/1.1 200 OK \r\n"
            response_header = "Server:pwb\r\n"
            response_body = file_data
            # 转换成二进制数据便于HTTP传输
            response = (response_line + response_header).encode() + response_body
            conn.send(response)
        finally:
            # 关闭套接字
            conn.close()
    def start(self):
        while True:
            # 接收返回参数
            conn,ip = self.socket.accept()
            sub_thread = threading.Thread(target = self.handle,args=(conn,))
            sub_thread.start()
my_web_driver = HttpWebServer()
my_web_driver.start()
```

命令行启动动态绑定端口号

```
import socket
import threading
## 用于读取终端信息
import sys

class HttpWebServer():
    def __init__(self,port):
        self.socket = socket.sockt(socket.AF_INET,socket.SOCK_STREAM)
        # 设置端口复用
        self.socket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,True)
        # 绑定端口和ip
        self.socket.bind(("",port))
        # 设置监听
        self.socket.listen(128)
    def handle(self,conn):
        # 接收请求
        # data里面有用户的请求信息，其中包含了请求的文件，我们可以根据这个信息返回指定页面
        data = conn.recv(1024).decode()
        request_data = data.spilt(" ")
        request_path = request_data[1]
        # 关闭客户端时客户端会返回一个空字符给服务端，这时候服务端无法解析，需要在这里进行判断
        if len(request_path) == 1:
            conn.close()
            return
        # 判断是否访问主页,直接访问ip会让request_path变为'/'
        if request_path == '/':
            request_path = '/index.html'
        # 打开文件并组成响应报文返回
        try:
            with open ("/static" + request_path,"rb") as f:
```

```
                file_data = f.read()
        except Exception as e:
            # 返回404错误数据
            response_line = "HTTP/1.1 404 Not Found\r\n"
            response_header = "Server:pwb\r\n"
            response_body = "404 Not Found HAHA"
            response = (response_line + response_header + response_body).encode()
            conn.send(response)
        else:
            response_line = "HTTP/1.1 200 OK \r\n"
            response_header = "Server:pwb\r\n"
            response_body = file_data
            # 转换成二进制数据便于HTTP传输
            response = (response_line + response_header).encode() + response_body
            conn.send(response)
        finally:
            # 关闭套接字
            conn.close()
    def start(self):
        while True:
            # 接收返回参数
            conn,ip = self.socket.accept()
            sub_thread = threading.Thread(target = self.handle,args=(conn,))
            sub_thread.start()
def main():
    # 终端返回的信息，是一个列表
    print(sys.argv)
    if len(sys.argv) != 2:
        print('格式错误')
        return
    if not sys.argv[1].isdigit():
        print('格式错误')
        return
    port = sys.argv[1]
    my_web_driver = HttpWebServer(port)
    my_web_driver.start()
if __name__ = '__main__':
    main()
```

函数参数

- 函数名存放的是函数所在空间的地址
- 函数名也可以像普通的变量赋值
- 函数名也可作为参数传递

```
def fun1():
    print('hello')

def fun(fun1):
    fun1()
```

```
fun(fun1) # 打印出hello
```

闭包

- 闭包的作用：可以保存函数内的变量，不会随着函数调用完而销毁
- 定义：函数嵌套前提下，内部函数使用了外部函数的变量，并且外部函数返回了内部函数，我们把使用外部函数变量的内部函数称为闭包

```
def fun1(num1):
    # 闭包
    def fun2(num2):
        num = num1 + num2
        print(num)
    # 返回内部函数
    return fun2;
## 创建闭包实例
f = fun1(10)
f(1) # 打印11
f(2) # 打印12
```

闭包内修改外部变量

使用**nonlocal**关键字：声明外部变量

```
def fun1(num1):
    # 闭包
    def fun2(num2):
        # 声明外部变量 使得fun2可以num1
        nonlocal num1
        num1 = num2 + 10
    print(num1)
    fun2(10)
    print(num1)
    # 返回内部函数
    return fun2;
## 调用
fun1(10)
## 分别打印10、20
```

with语句的使用

即使出现异常也可以自动关闭文件！！！

```
## 以with方法打开文件
with open('1.txt',r) as f:
    data = f.read()
    print(data)
```

生成器

根据程序设计者制定的规则循环生成数据，条件不成立时结束生成

生成器推导式

- **next**函数获取生成器中的下一个值

```
## 创建生成器
data = (i * i for i in range(100))
## 读取数据
print(next(data)) # 输出0
print(next(data)) # 输出1
for i in data:
    print(i)
```

- yield关键字
    - 代码执行到yiled会暂停，然后把结果返回出去，下次启动生成器会在暂停的位置继续往下执行
    - 数据生成完成再去调用生成器会抛出异常

```
def num():
    for i in range(10):
        print('开始生成')
        yield i
        print('生成完成')


g = num()
print(next(g)) # 打印开始生成 1
print(next(g)) # 打印生成完成 开始生成 1
```

- 生成斐波那契数列

```
def fn(num):
    a = 0
    b = 1
    index = 0
    while index < num:
        result = a
        a,b = b,a+b
        yield result
        index += 1

f = fn(5)
print(next(f)) # 打印0
print(next(f)) # 打印1
print(next(f)) # 打印1
```

深拷贝和浅拷贝

浅拷贝

- 对于可变类型

  - **copy**函数是浅拷贝**,**只对可变类型的第一层对象进行拷贝，会开辟新的内存空间进行存储，不会拷贝对象内部的子对象
- 对于不可变类型
  - 不会给拷贝的对象开辟新的内存空间，对于深拷贝和浅拷贝都是一样的

image-20241017200908328

只会拷贝地址而不会拷贝**a**和**b**的值

```
import copy
## 可变类型
a = [1,2,3,4]
b = [3,4,5,6]
c = [a,b]
d = copy.copy(c)
## 地址不一样
print(id(c))
print(id(0))
## 三者完全一样
print(id(a))
print(id(c[0]))
print(id(d[0]))

## 不可变类型
a = (1,2,3)
b = (4,5)
c = (a,b)
d = copy.copy(c)
## 地址一样
print(id(c))
print(id(d))
```

深拷贝

- 调用**deepcopy**函数对可变类型的每一层对象进行拷贝
- 深拷贝和浅拷贝的区别

  image-20241017202624486

```
import copy
## 可变类型
a = [1,2,3,4]
b = [3,4,5,6]
c = [a,b]
d = copy.deepcopy(c)
## 地址不一样
print(id(c))
print(id(d))
## 深层地址也不一样
```

```
print(id(c[0]))
print(id(d[0]))
```

logging(日志)

日志等级可以分为5个，从低到高分别是

- DEBUG: 调试bug时使用
- INFO: 程序正常运行时使用
- WARNING: 程序未按预期运行时使用，但不是错误
- ERROR: 程序出错误时使用
- CRITICAL: 特别严重的问题导致程序不能再继续运行时使用

- 默认为**WARNING**等级，当在**WARNING**或在**WARNING**之上等级才记录的日志信息

- logging日志等级和输出格式的设置
  - `logging.basicConfig(level = logging.DEBUG,format = '%(asctime)s - %(filename)s[line:%(lineo)d] - %(levelname)s:%(message)s', filename = 'log.txt',filemode = 'w')`
    - level表示日志等级
    - format为日志输出格式
      - %(asctime)s: 打印日志的时间
      - %(message)s: 打印日志信息
      - %(filename)s: 打印当前执行程序名
      - %(lineo)d: 打印日志当前行号
      - (levelname)s: 打印日志级别名称
    - filename: 日志文件名
    - filemode: 打开文件的方式

```
import logging
logging.basicConfig(level = logging.DEBUG,format = '%(asctime)s - %
(filename)s[line:%(lineo)d] - %(levelname)s:%(message)s', filename =
'log.txt',filemode = 'w')`

logging.info('你好啊')
```

web框架开发