

[Linux命令总结](Linux命令总结.docx)

mkdir 目录 -p 可以创建父目录

cp mv -i 交互式

-v 显示路径

重定向 >覆盖 >>追加

ls /bin > test.txt

查看文件 cat 文件名

合并文件 cat 1.txt 2.txt > 3.txt

管道 |

ls -la | more

软链接

ln -s 源文件 链接文件 不在同一目录，使用绝对路径

硬链接

ln 源文件 链接文件

文本搜索

grep 'a' 1.txt

-v 显示不包含匹配文本的所有行（相当于求反）

-n 显示匹配行号

-I 忽略大小写

^a ke$ [Ss]igna .

开头 结尾 多选 任意

查找文件

find . -name 文件名

解压缩

gzip

压缩用法：tar -zcvf 压缩包包名 文件1 文件2 ...

解压用法：tar -zxvf 压缩包包名 解压到指定目录：-C 目录

bz2

压缩用法： tar -jcvf 压缩包包名 文件

解压用法： tar -jxvf 压缩包包名

文件权限修改

chmod u/g/o/a +/-/= rwx 文件

chmod 751 文件

+ -a 保留权限

远程登陆

ssh 用户名@IP

文件传输

本地到远程

scp FileName RemoteUserName@RemoteHostIp:RemoteFile

scp -r FolderName RemoteUserName@RemoteHostIp:RemoteFolder

远程到本地

scp RemoteUserName@RemoteHostIp:RemoteFile FileName

scp -r RemoteUserName@RemoteHostIp:RemoteFolder FolderName

Ubuntu软件操作的相关命令

sudo apt-get update 更新源

sudo apt-get install package 安装包

sudo apt-get remove package 删除包

vim编辑

编辑模式 esc🡪 命令模式 🡨esc 末行模式

🡨i/o :🡪

常用命令：

yy: 复制 复制多行： 数字+yy

p: 粘贴 粘贴多次： 数字+p

dd：删除、剪切， 剪切或者删除多行：数字+dd

V: 选中多行

左下上右: H J K L

x: 从前往后删除一个字母

X: 从后往前删除一个字母

u: 撤销

shift + >: 往右缩进

shift + <: 往左缩进

.:重复上一次操作

G: 回到最后一行

gg: 回到第一行 1G

18G: 回到指定行

shift + 4:回到行末

shift + 6:回到行首

搜索: :/1 n:查看下一个搜索到的内容， N:查看上一个搜索到的内容

替换: :%s/1/2/g %: 整个文件， s:替换 g:全局

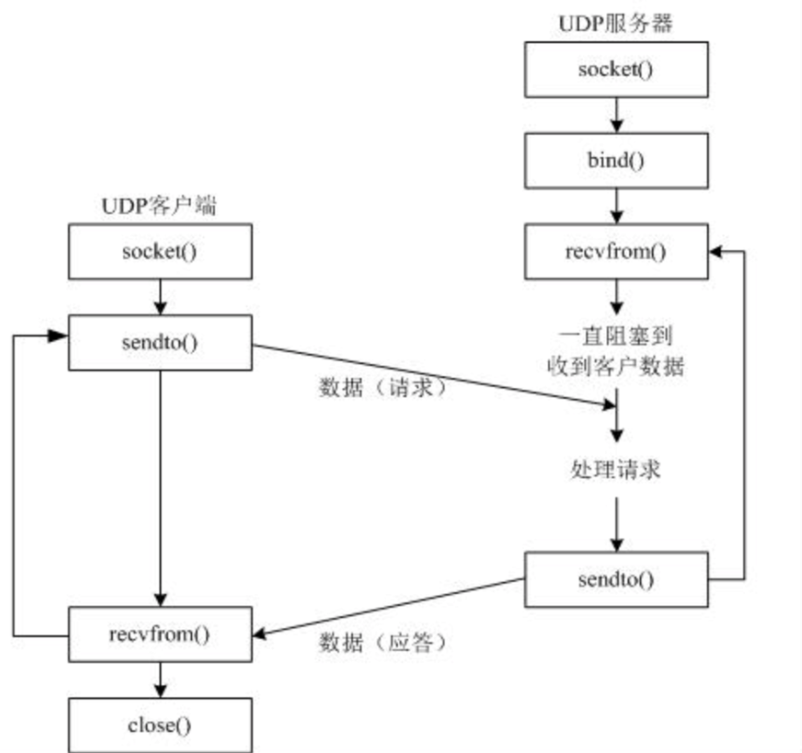
ctr+f: 下一页

ctr+b: 上一页

:wq== :x: 保存退出

netstat －an 查看程序所使用的端口号

sudo lsof -i [tcp/udp]:2425 查看端口号对应的应用程序



udp中socket相关内容

import socket

# 创建udp的套接字 AF\_INET ipv4 AF\_INET6 ipv6

#ip地址类型+传输协议 SOCK\_DGRAM udp SOCK\_STREAM tcp

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

#绑定本地的相关信息

udp\_socket.bind( (''”, 7788)) # ip地址和端口号，ip一般不用写，表示本机的任何一个ip

dest\_addr = ('192.168.1.103', 8080) # 注意是元组，ip是字符串，端口是数字

send\_data = input("请输入要发送的数据:")

#发送数据 编码格式utf-8或gbk+对方ip地址

udp\_socket.sendto(send\_data.encode('utf-8'), dest\_addr)

# 等待接收对方发送的数据

recv\_data , ip\_port= udp\_socket.recvfrom(1024) # 1024表示本次接收的最大字节数

# 显示对方发送的数据

print(recv\_data.decode('gbk')) # 显示解码数据

print(ip\_port) # ip和端口

# 设置socket的选项，允许发送广播消息

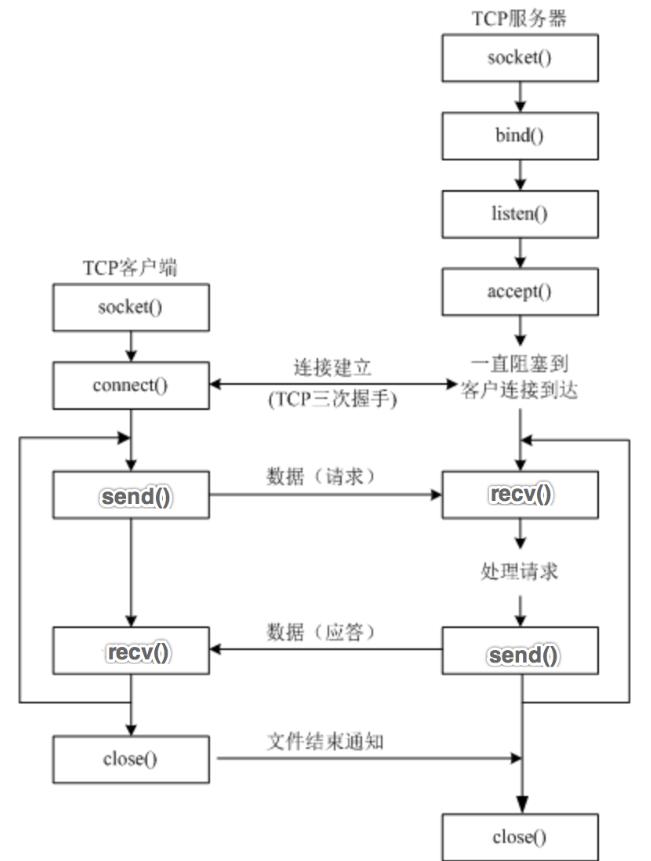
udp\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, True)

# 发送广播消息

udp\_socket.sendto("大家好".encode("gbk"), ("255.255.255.255", 9090))

# 不用的时候，关闭套接字

s.close()



tcp中socket相关内容

客户端client

import socket

#创建tcp客户端的socket

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

#设置服务器ip与port

server\_ipport=("192.168.16.65",8080)

#连接服务器

tcp\_client\_socket.connect(server\_ipport)

#发送数据给服务器

send\_data =input("请输入要发送的信息：")

tcp\_client\_socket.send(send\_data.encode("utf-8"))

#接受数据

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

print(recv\_data.decode("utf-8"))

tcp\_client\_socket.close()

服务器server

import socket

#创建tcp服务器的socket

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

#绑定本机ip

local\_add=("",8080)

tcp\_server\_socket.bind(local\_add)

#监听, tcp\_server\_socket改为了被动socket

#128表示最多允许128个客户端等待

tcp\_server\_socket.listen(128)

#等待服务,新建的client\_socket用于为客户端服务

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

print("链接成功",client\_ip)

#接受数据

recv\_data = client\_socket.recv(1024)

print(recv\_data.decode("utf-8"))

#发送数据

client\_socket.send("OK".encode("utf-8"))

#关闭

client\_socket.close()

多线程

import threading

#创建子线程

sing\_thread = threading.Thread(target=sing) #函数或方法名

#线程守护

sing\_thread.setDaemon(True)

#线程开始

sing\_threaf.start()

多线程共享全局变量

# 创建锁

lock= threading.Lock()

# 锁定

lock.acquire()

# 释放

lock.release()

进程多任务

import multiprocessing

#创建子进程

write\_process = multiprocessing.Process(target=write\_data)

#子进程开始

write\_process.start()

进程之间不共享全局变量

# 守护主进程

write\_process.daemon=True

#销毁子进程

write\_process. terminate()

进程间的通讯queue

#创建消息队列, 3:表示队列中最大消息个数

queue = multiprocessing.Queue(3)

#放入数据

queue.put(1)

queue.put("hello")

queue.put([3,5])

队列

queue.full() 是否满了bool

size = queue.qsize() 个数

value = queue.get() 值

进程池

# 创建进程池, 3:进程池中进程的最大个数

pool = multiprocessing.Pool(3)

#同步执行任务

#同步方式不需要主进程等待

pool.apply(work)

#异步执行任务

pool.apply\_async(work)

#异步执行需要让主进程等待进程池poor执行完成以后程序再退出

pool.close()

pool.join()

用来判断的方法

isinstance([5], Iterable) #是否可迭代

协程

import gevent

import time

from gevent import monkey

# 打补丁，让gevent框架识别耗时操作，比如：time.sleep，网络请求延时

monkey.patch\_all()

# 任务1

def work1(num):

for i in range(num):

print("work1....")

time.sleep(0.2)

# gevent.sleep(0.2)

# 任务1

def work2(num):

for i in range(num):

print("work2....")

time.sleep(0.2)

# gevent.sleep(0.2)

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

# 创建协程指定对应的任务

g1 = gevent.spawn(work1, 3)

g2 = gevent.spawn(work2, 3)

# 主线程等待协程执行完成以后程序再退出

#当前程序是一个死循环并且还能有耗时操作，就不需要加上join方法了,因为程序需要一直运行不会退出 (eg:while True:)

g1.join()

g2.join()

gevent.joinall([g1,g2])

[正则表达式](正则表达式/正则表达式30分钟入门教程%20-%20deerchao%20-%20博客园.html)

Import re

#匹配

result = re.match(正则表达式,要匹配的字符串)

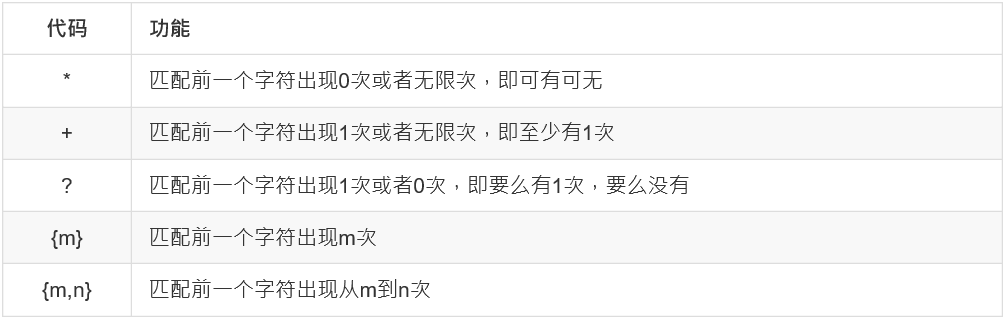
#提取数据

result.group()

匹配单个字符



匹配多个字符



{m,} 匹配前一个字符至少出现m次

\转义,eg:\\*

匹配开头结尾



[^abc123] 除了指定字符以外都匹配

匹配分组



多括号嵌套时,按左括号的顺序

re模块的高级用法

search查找

match\_obj = re.search("\d+", "水果有20个 其中苹果10个") 🡪20

findall 查找所有

result = re.findall("\d+", "苹果10个 鸭梨5个 总共15个水果") 🡪['10', '5', '15']

findall只找分组中内容, 修改都找 (?: )

sub替换

result = re.sub("\d+", "22", "评论数:10 赞数:20", count=1) 🡪评论数:22 赞数:20

result = re.sub("\d+", add, "阅读数:10") add函数

(def add(match\_obj):

value = match\_obj.group()

result = int(value) + 1

return str(result)

)

split 切割

result = re.split(",|:", my\_str, maxsplit=1) 🡪以,或:分割my\_str字符串

贪婪:默认贪婪

非贪婪: \*后加? ,?前不能匹配后面的字符串

R表示原始字符串,不需要转义” \\”

http请求报文

GET / HTTP/1.1 🡪请求行

Host: www.baidu.com 🡪IP地址,端口默认80

Connection: keep-alive 🡪长连接

Cache-Control: max-age=0 🡪缓存策略,不缓存

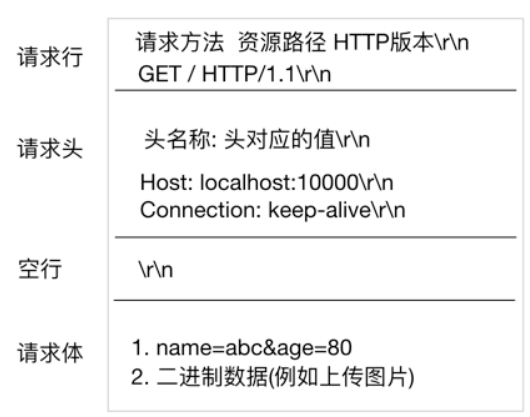
Upgrade-Insecure-Requests: 1 🡪使用https方式

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36 🡪浏览器类型

Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8 🡪接收的数据类型

Accept-Encoding: gzip, deflate, br 🡪压缩算法

Accept-Language: zh-CN,zh;q=0.9 🡪语言类型



GET没有请求体

POST有请求体

报文内容中\r\n 换行

http响应报文

HTTP/1.1 200 OK 🡪响应行 协议+状态号

Connection: Keep-Alive

Content-Encoding: gzip

Content-Type: text/html; charset=utf-8

Date: Wed, 14 Mar 2018 09:52:48 GMT 🡪服务器时间 0时区

Server: BWS/1.1

Content-Leng:10 🡪响应体长度

响应体

状态码:

200 成功

307 重定向

403 forbidden

404 not found

500 服务器出错

浅拷贝与深拷贝

浅拷贝拷贝引用,外层的拷贝

深拷贝拷贝数据,递归的拷贝

重新导入模块

from imp import reload

reload(xxx)

from xxx import y

y=z 不改变xxx中的数值

import xxx

xxx.y=z 改变xxx中的数值

property属性

使方法变为属性

方法一:

class P:

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

self.p1=p1

@property 创建

def func(self):

…

return self.p1

@ func.setter 设置

def func(self,value):

self.p1 = value

@ func.deleter 删除

def func(self):

del self.p1

p.func

p.func = 2

del p.func

方法二:

class Foo(object):

def get\_bar(self):

print("getter...")

return 'laowang'

def set\_bar(self, value):

"""必须两个参数"""

print("setter...")

return 'set value' + value

def del\_bar(self):

print("deleter...")

return 'laowang'

BAR = property(get\_bar, set\_bar, del\_bar, "description...")

obj = Foo()

obj.BAR # 自动调用get\_bar

obj.BAR = "alex" # 自动调用set\_bar方法

del obj.BAR # 自动调用del\_bar方法

with、上下文管理器

方法一:

class File():

def \_\_init\_\_(self, filename, mode):

self.filename = filename

self.mode = mode

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

print("entering")

self.f = open(self.filename, self.mode)

return self.f

def \_\_exit\_\_(self, \*args):

print("will exit")

self.f.close()

\_\_enter\_\_() 方法返回资源对象，这里就是你将要打开的那个文件对象，\_\_exit\_\_() 方法处理一些清除工作。

因为 File 类实现了上下文管理器，现在就可以使用 with 语句了。

with File('out.txt', 'w') as f:

print("writing")

f.write('hello, python')

方法二:

from contextlib import contextmanager

@contextmanager

def my\_open(path, mode):

f = open(path, mode)

yield f

f.close()

调用

with my\_open('out.txt', 'w') as f:

f.write("hello , the simplest context manager")

装饰器

通用装饰器

def set\_func(func):

def call\_func(\*agrs,\*\*kwargs):

#插入要添加的功能

return func(\*agrs,\*\*kwargs)

return call\_func

@set\_func

def test(num):

print(num)

test(100)

如果有多个装饰器,优先执行离函数最近的那个

@set\_func1

@set\_func2

def test(num):

…

test ⬄ set\_func1(set\_fun2(test))

带参数的装饰器

def fun\_arg(pre="hello"):

def fun(func):

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

print(pre)

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

return wrappedfunc

return fun

@fun\_arg("itcast")

def foo():

print("I am foo")