# gitlab

# 步骤一：通过docker搭建gitlab服务器

[root@gitlabserver ~]# ls

anaconda-ks.cfg

docker-engine-1.12.1-1.el7.centos.x86\_64.rpm #两个安装包需要单独传送,安装有先后docker-engine-selinux-1.12.1-1.el7.centos.noarch.rpm

~]# yum -y install docker-engine-selinux-1.12.1-1.el7.centos.noarch.rpm

~]# yum -y install docker-engine-1.12.1-1.el7.centos.x86\_64.rpm

~]# yum -y install docker-engine #不需要安装

1. 上传gitlab\_zh.tar镜像文件，至gitlib服务器,导入中文版gitlab镜像

[root@gitlabserver images]# docker load < gitlab\_zh.tar #进入有gitlab镜像的目录否则要绝对路径

a94e0d5a7c40: Loading layer [================>] 116.5MB/116.5MB

88888b9b1b5b: Loading layer [==================>] 15.87kB/15.87kB

52f389ea437e: Loading layer [=================>] 14.85kB/14.85kB

52a7ea2bb533: Loading layer [=================>] 5.632kB/5.632kB

db584c622b50: Loading layer [=================>] 3.072kB/3.072kB

62786ff6a243: Loading layer [==================>] 75.85MB/75.85MB

71bc04f4b7c7: Loading layer [==================>] 2.048kB/2.048kB

26e083d332d8: Loading layer [==================>] 2.048kB/2.048kB

2c02e58e96b8: Loading layer [==================>] 2.048kB/2.048kB

589c7a23de2a: Loading layer [==================>] 15.87kB/15.87kB

44474d2cdcd1: Loading layer [=================>] 1.359GB/1.359GB

41c94e16b901: Loading layer [=================>] 16.78MB/16.78MB

04cafa6a1534: Loading layer [=================>] 160MB/160MB

Loaded image: gitlab\_zh:latest

1. 将物理主机ssh端口改为2022后，输入如下命令起动容器：

[root@gitlabserver images]# vim /etc/ssh/sshd\_config #改ssh端口

[root@gitlabserver images]# docker run -d -h gitlab --name gitlab -p 443:443 -p 80:80 -p 22:22 --restart always -v /srv/gitlab/config:/etc/gitlab -v /srv/gitlab/logs:/var/log/gitlab -v /srv/gitlab/data gitlab\_zh:latest

b9dc65e0def51a4d09d2a597b2b929490e972a34f3de993439d2f7cc22039b77

此时端口成功启动

[root@gitlabserver images]# docker ps #查看确保启动无误

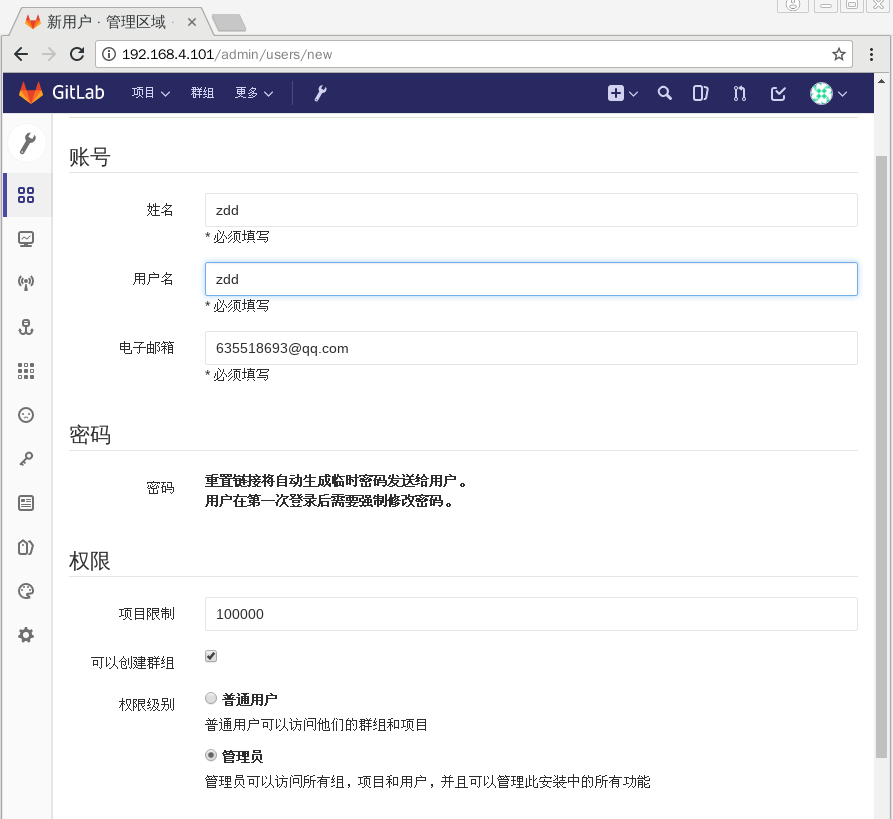
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

a9e217dfe4b9 gitlab\_zh:latest "/assets/wrapper" 2 minutes ago Up 6 seconds (health: starting) 0.0.0.0:22->22/tcp, 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp gitlab

## 步骤二:初始化git,以及新建群组

初次登录,重置root密码为:q12345678 zdd:q12345678





# 函数调用集

**import random**

from random import choice #取随机字符

from random import randint #取随机数

**import sys** #位置参数

copy(sys.argv[1], sys.argv[2]) #,argv是sys模块中的数列，将位置参数存到该列表中

**import string** #字符集

print(string.digits) # 输出包含数字0~9的字符串

print(string.ascii\_letters) # 包含所有字母(大写或小写)的字符串

print(string.ascii\_lowercase) # 包含所有小写字母的字符串

print(string.ascii\_uppercase) # 包含所有大写字母的字符串

print(string.punctuation) # 所有特殊字符

**import shutil** #shell工具,可调用shell工具,详情请看下面shutil专题

**import keyword**

keyword.iskeyword #所有关键字集合

if keyword.iskeyword(idt): #判断idt变量值是否是关键字，如果是则执行下面return

**import os** #文件访问系统,详情请看下面os模块专题

**import subprocess** #调用系统命令,详情请看下面subprocess专题

**import getpass** #

passwd = getpass.getpass('密码:') #定义输入的密码,且是不明文显示,可作为输入窗口

**import time** #时间函数,详情请看线面time模块专题

**import pickle** #文件读写模块,详情请看下面pickle专题

**import hashlib** #md5算法

**import re** #正则匹配

from collections import Counter #Counter函数统计指定内容的出现次数,并排序

**import socket**

**urllib模块** #处理URL,爬取网站

# 列表

## @@斐波那契数列

# fib = [0, 1]

# for i in range(8):

# fib.append(fib[-1] + fib[-2]) # 将列表中最后两项之和，追加到列表中

# print(fib)

###############################

fib = [0, 1]

n = int(input('长度: '))

for i in range(n - 2):

fib.append(fib[-1] + fib[-2])

print(fib)

## @@猜数字游戏

import random

all\_choice = ['石头', '剪刀', '布']

win\_list = [['石头', '剪刀'], ['剪刀', '布'], ['布', '石头']]

prompt = '''(0) 石头

(1) 剪刀

(2) 布

请选择(0/1/2): '''

pwin = 0

cwin = 0

while pwin < 2 and cwin < 2: # 人机都没赢够2次，继续循环，三局两胜

computer = random.choice(all\_choice)

ind = int(input(prompt)) # 将用户输入的字符转换为数字

player = all\_choice[ind] # 通过数字下标，取出列表中的字符串

print('你的选择是:%s, 计算机出拳: %s' % (player, computer))

if player == computer:

print('\033[32;1m平局\033[0m')

elif [player, computer] in win\_list:

pwin += 1

print('\033[31;1mYou WIN!!!\033[0m')

else:

cwin += 1

print('\033[31;1mYou LOSE!!!\033[0m')

## @@九九乘法表

# for i in range(3): # i 控制打印几行

# # 一行内打印3个hello后，打印回车

# for j in range(3): # 内层循环控制一行内打印几次hello

# print('hello', end=' ') # print默认在结尾打印\n，将结尾修改为空格

# print()

########################

# for i in range(3): # i 控制打印几行

# for j in range(i + 1): # 内层循环控制一行内打印几次hello

# print('hello', end=' ') # print默认在结尾打印\n，将结尾修改为空格

# print()

########################

for i in range(1, 10):

for j in range(1, i + 1):

print('%sx%s=%s' % (j, i, i \* j), end=' ')

print()

~

# 遍历

chs = 'hello'

alist = ['bob', 'alice']

atuple = (10, 20)

adict = {'name': 'tom', 'age': 23}

for ch in chs:

print(ch)

for name in alist:

print(name)

for i in atuple:

print(i)

for key in adict:

print('%s: %s' % (key, adict[key]))

# while循环

# 列表解析

## 通过for循环

# print([10 + 5 for i in range(5)])

#输出：[15, 15, 15, 15, 15]，#10+5这个计算5次，for循环了5次

# print([10 + i for i in range(5)])

#输出：[10, 11, 12, 13, 14] #10+i,i来自于后面的for循环

# print([10 + i for i in range(2,5)])

#输出：[12, 13, 14] #10+i,i来自于后面的for循环

## 通过if过滤，

满足if条件的才参与10+i的运算

# print([10 + i for i in range(1,11) if i % 2 == 0]) #输出[12, 14, 16, 18, 20]

# print([10 + i for i in range(1,11) if i % 2 == 1]) #输出[11, 13, 15, 17, 19]

# print(['192.168.1.%s' % i for i in range(1,10) if i > 8]) #输出['192.168.1.9']

# a = ['192.168.1.%s' % i for i in range(1,10) if i > 8]

# print(a) #输出['192.168.1.9']

# 算术运算符

Python算术运算符

以下假设变量： a=10，b=20：

运算符 描述 实例

+ 加 - 两个对象相加 a + b 输出结果 30

- 减 - 得到负数或是一个数减去另一个数 a - b 输出结果 -10

\* 乘 - 两个数相乘或是返回一个被重复若干次的字符串 a \* b 输出结果 200

/ 除 - x除以y b / a 输出结果 2

% 取模 - 返回除法的余数 b % a 输出结果 0

\*\* 幂 - 返回x的y次幂 a\*\*b 为10的20次方， 输出结果 100000000000000000000

// 取整除 - 返回商的整数部分（向下取整）

>>> 9//2

4

>>> -9//2

-5

>>> 40-3\*\*2+11//3\*\*2\*8

39

# 文件对象：

文件操作三个步骤：打开、读写、关闭

* r,以读方式打开(文件不存在则报错)
* w,以写方式打开(文件存在则清空,不存在则创建)
* a,以追加模式打开,(必要时创建新文件)
* b,以二进制模式打开,以bytes方式打开(直接显示为2进制的方式)
* r+/w+/a+,以读写模式打开

## 1.read,默认读取文件的全部内容

# f = open('/tmp/passwd1') #打开文件

# data =

# #读取文件

# print(data) #输出

# #读取文件时,文件指针会向后移动,读取全部内容后文件指针已经到结尾,再读就没有内容了

# # data = f.read()

# # print(data)

# data2 = f.read()

# print(data2) #data2是没有数据的,因为第一个data = f.read()全部读取完毕了

# data = f.close() #关闭

# f = open('/tmp/passwd2')

# data = f.read(4) #取前面四个字符

# print(data)

# data2 = f.read(4) #接着上面再读取四个字符

# print(data2)

# f.close() #关闭f 打开的文件

# print(f.read(4)) #报错,已经被关闭ValueError: I/O operation on closed file.

## 2.readline,读一行####

# f = open('/tmp/passwd1')

# a = f.readline() #读取第一行

# print(a)

# b = f.readline() #接着上面读取第二行

# print(b)

# f = open('/tmp/passwd1').readline() #不建议使用,无法读取第二行

# print(f)

## 3.readlines 读取所有行,放到列表中,

# f = open('/tmp/passwd2')

# a = f.readlines()

# print(f) #输出一个列表,每个value就是文件中的一行

## 4.通过for循环进行遍历,常用,需要记住

# f = open('/tmp/passwd2')

# for line in f:

# print(line,end='')

# f.close() #关闭

## 5.读取非文本文件

# f = open('/tmp/windows.jpg')

# a = f.read(4) #报错,默认情况下,文件被认为是文本文件,python试图将取出的4字节转换成文字,图片无法转换成文字

# f.close()

# f = open('/tmp/windows.jpg','rb') #以rb的方式打开,

# print(f.read(4)) #输出b'\xff\xd8\xff\xe0' 16进制数,python将2进制以16进制方式进行显示

# print(f.read(4)) #输出b'\x00\x10JF' #接着上面的显示

# f.close()

# f = open('/tmp/windows.jpg','rb') #以rb的方式打开,

# print(f.read(8)) #输出b'\xff\xd8\xff\xe0\x00\x10JF'

# f.close()

## 6,写入

# f = open('/tmp/passwd1','w') #以写的方式打开

# a = f.write('hello world!\n') #没有则创建.有的话就清空再写入,

# b = f.write('hello world!\n')

# # c = f.writelines('hello world!\n') #???????????

# print(a,b,)

#[root@room9pc01 tmp]# cat passwd1 #查看实际文件内容

#hello world!

#hello world!.

# f = open('/tmp/passwd1','w')

# a = f.writelines(['2nd line.', '3rd line.\n'])

# print(a)

# f.close() #关闭,也写入数据

###[root@room9pc01 tmp]# cat passwd1 #查看实际文件内容

###2nd line.3rd line.

## 7.with子句:语句结束,文件自动关闭

# with open('/tmp/passwd1') as f:

# print( f.readline() ) #显示文件的内容:2nd line.3rd line.

#

# # print(f.readline()) #继续执行.报错,file已closed

## 8.文件指针###以后用不着(了解)

# f = open('/tmp/passwd','rb') #以可读与二进制方式打开

# print(f.tell()) #输出0,

# f.read(5) #读取5个字节后

# print(f.tell()) #查看输出5,指针偏移5个字节

#####

###seek函数用与移动文件指针,有两个参数.第二个参数表示相对位置.

###第二个参数(0表示开头,1表示当前位置.2表示结尾),第一个参数是相对于第二个参数的偏移量

# f = open('/tmp/passwd1','rb')

# print(f.tell()) #当前指针位置0

# print(f.read(5)) #输出b'2nd l' #文件原文2nd line.3rd line.

# print(f.seek(3,1)) #输出8,,1表示当前位置.3表示偏移5个字节

# print(f.readline()) #输出b'.3rd line.\n' #文件原文2nd line.3rd line.

...

## @@cp功能命令

src\_fname = '/bin/ls' #定义源文件

dst\_fname = '/tmp/ls' #定义目标文件

src\_fobj = open(src\_fname,'rb') #以rb(读和二进制)方式打开，源文件

dst\_fobj = open(dst\_fname, 'wb') #=以写和二进制方式打开目标文件，没有则创建

while 'true': #无限制读取

data = src\_fobj.read(4096) #因为读取数据是放入内存,所以每次读取只读取4字节

# if data == b'': #判断方式1,data数据等于空,则表示数据取完,执行break

# if len(data) == 0: #判断方式2,统计data变量的数据长度,等于0,表示数据取完,执行break

if not data: #判断方式3,如果data变量是空字符串,则False.

break #终止所有循环

dst\_fobj.write(data) #将data变量值写入dst\_fobj,参考上面写入练习

src\_fobj.close()

dst\_fobj.close()

###验证,电脑上操作如下############

[root@room9pc01 tmp]# md5sum ls #哈希值验证两个文件是否一致

918cb545b3458e1bf18b712b36af304f ls

[root@room9pc01 tmp]# md5sum /bin/ls

918cb545b3458e1bf18b712b36af304f /bin/ls

## @@cp模块（函数方式）

import sys

def copy(src\_fname, dst\_fname): #定义函数,并定义两个参数

src\_fobj = open(src\_fname,'rb') #打开源文件,以rb方式

dst\_fobj = open(dst\_fname, 'wb')

while 'true': #无限制读取

data = src\_fobj.read(4096) #因为读取数据是放入内存,所以每次读取只读取4字节

if not data: #判断方式3,如果data变量是空字符串,则False.

break #终止所有循环

dst\_fobj.write(data) #将data变量值写入dst\_fobj,参考上面写入练习

src\_fobj.close()

dst\_fobj.close()

copy(sys.argv[1], sys.argv[2]) # argv是sys模块中的数列，将位置参数存到该列表中

##执行以及验证结果

[root@room9pc01 day02]# ls

day02.py day03.py home-day02.py test.py

[root@room9pc01 day02]# python3 day02.py home-day02.py test2.py

[root@room9pc01 day02]# ls #将home-day02.py复制到test2.py

day02.py day03.py home-day02.py test2.py test.py

# 函数

## 调用、默认值

####斐波那契数列

def gen\_fib(n=10): #定义名为gen\_fib函数，默认参数n=10

#number = int(input('请输入要计算的斐波那契数列范围:'))在这不再手动输入,在函数gen\_fib(n)中调用n参数

alist = [0,1] #定义最初始的列表,占用两个了

for i in range(n-2): #这里接受参数n.输入了10 ,实际运行要减去2个,

alist.append(alist[-1] + alist[-2]) #将列表中最后一个数加上倒数第二个的和,添加到列表中

# print(alist) #输入5 得到结果[0, 1, 1, 2, 3] 函数中不用这个输出,否者固定在函数中输出

return alist #将最后得到的alist 列表结果返回给函数gen\_fib(),函数结果需要return返回

mylist = gen\_fib() #调用函数执行,未传参,将会以默认参数n=10执行

print(mylist)

## 函数的位置参数

###cp功能模块

import sys

def copy(src\_fname, dst\_fname): #定义函数,并定义两个参数

src\_fobj = open(src\_fname,'rb') #打开源文件,以rb方式

dst\_fobj = open(dst\_fname, 'wb')

while 'true': #无限制读取

data = src\_fobj.read(4096) #因为读取数据是放入内存,所以每次读取只读取4字节

if not data: #判断方式3,如果data变量是空字符串,则False.

break #终止所有循环

dst\_fobj.write(data) #将data变量值写入dst\_fobj,参考上面写入练习

src\_fobj.close()

dst\_fobj.close()

copy(sys.argv[1], sys.argv[2]) # argv是sys模块中的数列，将位置参数存到该列表中

import导入模块

1文件:是python从物理上组织代码的形式,

2模块:是python从逻辑上组织代码的形式,

3 以.py 我为结尾的python文件都是一个模块

4 模块文件名字去掉后面的扩展名（.py）即为模块

## import 导入模块1

hi = 'hello zdd'

def pstar(n=30):

print('$' \* n)

###实际执行如下#########

[root@room9pc01 tmp]# cd /root/PycharmProjects/day02/

[root@room9pc01 day02]# python3

>>> import day02 #调用名为day02的python文件全名为:day02.py

>>> day02.hi #调用day02文件中的hi变量

'heelo zdd'

>>> day02.pstar() #调用day02中的pstar函数

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

## import 导入模块2

#使用import导入就会执行一遍语句

hi = 'heelo zdd'

def pstar(n=30):

print('$' \* n)

pstar()

pstar(40)

[root@room9pc01 day02]# python3 day02.py #直接执行

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

[root@room9pc01 day02]# python3

Python 3.6.1 (default, Apr 27 2018, 04:53:30)

[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> import day02 #调用这个模块也会执行一遍

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

>>>

## 模块导入的特性 \_\_name\_\_

当模块文件直接运行时,\_name\_的值是:'\_main\_'

当模块文件被导入时,\_name\_的值是:该模块的名字

#利用这个特性可控制模块的执行方式

hi = 'heelo zdd'

def pstar(n=30):

print('$' \* n)

if \_\_name\_\_ == '\_\_main\_\_': #判断\_\_name\_\_状态，如果是直接运行python文件就执行下面调用

pstar()

pstar(40)

[root@room9pc01 day02]# python3 day02.py

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

hi = 'heelo zdd'

def pstar(n=30):

print('$' \* n)

if \_\_name\_\_ == 'day02': #判断\_\_name\_\_状态，如果是被导入python文件就执行下面调用

pstar()

pstar(40)

##[root@room9pc01 day02]# python3

##>>> import day02 #如果是import调用才执行

##$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

##$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

## 随机密码

所有字母大小写+所有数字

import string # 导入string这个模块

print(string.digits) # 输出包含数字0~9的字符串

print(string.ascii\_letters) # 包含所有字母(大写或小写)的字符串

print(string.ascii\_lowercase) # 包含所有小写字母的字符串

print(string.ascii\_uppercase) # 包含所有大写字母的字符串

print(string.punctuation) # 所有特殊字符

### @@随机密码用户自己定义密码长度（自己写）

import string,random

all\_dx = string.ascii\_letters

all\_sx = string.digits

all = all\_dx + all\_sx

def suiji(n=8):

mm = ''

for i in range(n):

sj\_1 = random.choice(all)

mm += sj\_1

return mm

mm\_cd = int(input('请输入密码长度: '))

a = suiji(mm\_cd)

print(a)

### @@随机密码

from random import choice #调用random中的choice

from string import ascii\_letters,digits #/usr/local/lib/python3.6/string.py

all\_chs = ascii\_letters + digits #from方式调用， string可省略（string. ascii\_letters）

def gen\_pass(n=8):

result = ''

for i in range(n):

ch = choice(all\_chs)

result += ch

return (result)

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

print(gen\_pass())

print(gen\_pass(4))

# shutil模块

## shutil:shell工具

## 1,shutil.copyfileobj(复制文件内容@常用)

* shutil.copyfile(src, dst)：复制文件内容（不包含元数据）从src到dst。
* dst必须是完整的目标文件名;拷贝目录参见shutil.copy()。
* 如果src和dst是同一文件，就会引发错误shutil.Error。
* dst必须是可写的，否则将引发异常IOError。
* 如果dst已经存在，它会被替换。
* 特殊文件，例如字符或块设备和管道不能使用此功能，因为copyfile会打开并阅读文件。
* src和dst的是字符串形式的路径名。

import shutil

file\_a = open('/etc/issue','rb')

file\_b = open('/tmp/issue1','wb')

shutil.copyfileobj(file\_a,file\_b)

file\_a.close()

file\_b.close()

####实际结果

[root@room9pc01 tmp]# cat /etc/issue

\S

Kernel \r on an \m

[root@room9pc01 tmp]# cat /tmp/issue1

\S

Kernel \r on an \m

## 2.shutil.copyfile(不用打开文件就直接复制)

shutil.copyfile(文件1，文件2)：不用打开文件，直接用文件名进行覆盖copy。

import shutil

shutil.copyfile("1.txt","3.txt")

## shutil.copymode(权限拷贝)

shutil.copymode(文件1，文件2)：之拷贝权限，内容组，用户，均不变。

def copymode(src,dst):

"""copy mode bits from src to dst"""

if hasattr(os,'chmod'):

st = os.stat(stc)

mode = stat.S\_IMODE(st.st\_mode)

os.chmod(dst,mode)

## 4,shutil.copy(拷贝文件@常用)

* shutil.copy(src, dst)：复制文件src到文件或目录dst。dst目录需先存在
* 如果dst是目录，使用src相同的文件名创建（或覆盖），权限位也会复制。
* src和dst的是字符串形式的路径名。
* 拷贝文件和权限都进行copy。

import shutil

shutil.copy('/etc/issue.net','/tmp') #拷贝文件, （常用，背下来）

[root@room9pc01 tmp]# ls /tmp/issue\*

/tmp/issue1 /tmp/issue2

## 5,shutil.copy2()

shutil.copy2(src, dst)：类似shutil.copy，元数据也复制，实际上先调用shutil.copy，然后使用copystat。这类似于Unix命令cp -p。

#相当于cp -p

## 6,shutil.move(@@常用)

递归移动一个文件。

import shutil

shutil.move('/tmp/issue.net','/hahahahah') # -mv文件（常用，背下来）

## 7,shutil.remtree(@常用)

可以递归删除目录下的目录及文件。

#相当于rm -rf 但是只能用于删除目录,不能删除文件.（常用，背下来）

import shutil

shutil.copytree('/etc/security', '/tmp/anquan') #拷贝目录用作测试

shutil.rmtree('/tmp/anquan') #删除目录,相当于rm -rf

## 8.shutil.copystat

shutil.copystat(文件1，文件)：只拷贝了权限。

将权限，上次访问时间，上次修改时间以及src的标志复制到dst。文件内容，所有者和组不受影响

src： 源文件路径

dst： 将权限复制至dst文件，dst路径必须是真实的路径，并且文件必须存在，否则将会报文件找不到错误

follow\_symlinks：设置为False时，src, dst皆为软连接，可以复制软连接权限、上次访问时间，上次修改时间以及src的标志，如果设置为True，则当成普通文件复制权限。默认为True。Python3新增参数

## 9.shutil.chown ---待查

chown(path, user=None, group=None)： 修改路径指向的文件或文件夹的所有者或分组。Python3新增方法

path：路径

user：所有者，传递user的值必须是真实的，否则将报错no such user

group：分组，传递group的值必须是真实的，否则将报错no such group

import shutil,os

path = os.path.join(os.getcwd() ,"file.txt")

shutil.chown(path,user="root",group="root")

# 语法风格

## 变量复制,多重赋值

x = y = 10

print(x)

print(y) #输出：分两行输出两个10

a , b = 10 ,20

print(a , b) #输出：10 20

a, b = b, a # 互换a和b的值

print( a,b) #输出：20 10

aa , bb = (10,20)

print(aa,bb) #输出：10 20

aaa,bbb = [10,20]

print(aaa,bbb) #输出：10 20

## 合法标识符

合法标识符满足如下条件：

* 不能是关键字
* 第一个字符不能是数字，必须是字母或者下划线，剩下的可以是字母，数字，下划线
* 大小写敏感
* Python标识符字符串规则和其他大部分用C语言编写的高级语言相似

## 关键字

* 关键字应该保持稳定，但是pythong语言是一门不断成长和进化的语言，关键字偶尔会更新
* 关键字被保留，不能作为函数名，变量名等
* 加载关键字列表和iskeyword()函数都放入了keyword模块，以便于查阅

import keyword

print(keyword.kwlist) #输出所有关键字

print(keyword.iskeyword('pass')) #检测是否是关键字 输出:True,说明pass是关键字

print(keyword.iskeyword('adp')) #输出:False,说明adp不是关键字

## 内建

* 除了关键字之外，python还有可以在任何一级代码使用的‘内建’的名字集合，
* 这些名字由解释器设置或使用
* 虽然built-in不是关键字，但是应该把它当做“系统保留字”
* 保留的常量如：True、False 、None等
* 内建不应被当做变量来被覆盖

# 模块布局

## 模块布局模板

#!/usr/bin/env python #起始行

“this is a test module” #模块文档字符串

import sys #导入模块

import os

debug = True #全局变量声明

class FooClass(object): #类定义

'Foo class'

pass

def test(): #函数定义

"test funcDon"

foo = FooClass()

if \_\_name\_\_ == ‘\_\_main\_\_’: #程序主体

test()

## 编程思路：

1、弄清要求

2、发呆：思考程序是怎么运行的（交互的、非交互的？），如果是交互的脑补程序运行有什么

3、思考程序有哪些功能，把这些功能写成函数

4、编写主程序，调用各个功能函数

5、最后编写每个功能函数

## @@创建文件程序

#输入一个文件名，判断文件是否存在，存在则提示已存在，不存在则创建该文件，并可输入内容保存

! /usr/bin/env python

"this is a test module"

import sys,os #调用模块

def get\_fname(): #函数,y

while True: #循环

fname = input('请输入文件名: ') #此处的fname只在本循环中存在

if not os.path.exists(fname): #判断文件名是否已存在,如果不存在则执行break

break

print('该文件已存在,请重试!')

return fname #将fname的结果返回给get\_fname

def get\_content():

content = []

print('请输入内容,另起一行输入"end"结束!')

while True:

line = input('> ')

if line == 'end': #判断输入的如果是end 则执行break,结束所有循环

break

content.append(line) #判断输入的如果不是end 则将line的内容添加到content列表中

return content

def wfile(fname, content):

with open(fname,'w') as fobj: #以写方式打开文件fname，如果文件不存在则创建，with语句结束自动关闭文件

fobj.writelines(content) #将congtent收集的输入所有内容，写入到fname

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

fname = get\_fname() #fname得到get\_fname():中产生的不存在同名的文件名

content = get\_content() #content得到get\_content():中的产生的列表内容

content = [line + '\n' for line in content] #将content列表所有value添加一个\n

wfile(fname,content) #将content的内容写入到名为fname的文件中,没有则创建

# 字符串(str)

## 字符串操作符

* 比较操作符：字符串大小按ASCII码值大小进行比较
* 切片操作符：[] 、 [:] 、 [ : : ]
* 成员关系操作符：in not in



## enumerate新列表

将列表重新组成含标识的列表

alist = ['tom','bob','jerry','alice']

for i\_1 in [0,1,2,3]:

print('%s : %s' % (i\_1,alist[i\_1]))

for i\_2 in range(4):

print('%s : %s' % (i\_2, alist[i\_2]))

for i\_3 in range(len(alist)):

print('%s : %s' % (i\_3, alist[i\_3]))

print(list(enumerate(alist))) #输出[(0, 'tom'), (1, 'bob'), (2, 'jerry'), (3, 'alice')]#enumerate将列表重新组成含标识的列表

for i\_4 in enumerate(alist):

print('%s :　%s' % (i\_4)) #必需两个%s

for i\_5, i\_6 in enumerate(alist):

print('%s:%s' % (i\_5,i\_6)) #与上一样，这种分两个变量更灵活，（常用）

####以上所有输出都一样如下

0:tom

1:bob

2:jerry

3:alice

## 列表翻转

alist = ['tom','bob','jerry','alice']

print(reversed(alist)) #输出一个文件:<list\_reverseiterator object at 0x7f2e1c9274e0>

print(list(reversed(alist))) #翻转，但是列表本身不变,输出['alice', 'jerry', 'bob', 'tom']

print(alist.reverse()) #翻转，改变列表本身

print(sorted(alist)) #排序，但是列表本身不变，输出['alice', 'bob', 'jerry', 'tom']

print(list(sorted(alist) #排序，但是列表本身不变，输出['alice', 'bob', 'jerry', 'tom']

print(alist.sort()) #排序，改变列表本身

## @@判断输入的标识符是否合法程序

import sys

import keyword,string

first\_chs = string.ascii\_letters + '\_' # first\_chs为所有大小写+下划线的集合

other\_chs = first\_chs + string.digits # first\_chs为特殊字符+下划线+数字的集合

def check\_idt(idt):

if keyword.iskeyword(idt): #判断是否是关键字，如果是则执行下面return

return '% 是关键字' % idt #一个函数只能return一次，return后函数结束，

if not idt[0] in first\_chs: #判断第一个字符如果不是大小写或者下划线，则执行return

return '第一个字符不合法'

for ind,ch in enumerate(idt[1:]): #把输入的字符串，从第2个开始重新列表，

if ch not in other\_chs: #判断上面获取的ch如果不是大小写或者下划线，则执行return

return '第%s 个字符不合法' % (ind + 2) #因为是从第2个开始，所以新标识跟实际相差2个

return '%s 是合法的标识符' % idt

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

print(check\_idt(sys.argv[1]))

####执行之后的结果##############

# [root@room9pc01 day02]# python3 day02.py asdf

# asdf 是合法的标识符

# [root@room9pc01 day02]# python3 day02.py 123sdfa

# 第一个字符不合法

# [root@room9pc01 day02]# python3 day02.py asdf@sdf

# 第5 个字符不合法

# 字符串格式化

## 1,基础样式

print('%s' % 'bob')

## 2括号可省略

print('%s' % 'bob') #当只有一个%s 占位符，后面的（）可省略

## 3字符串





print('%s is %s years old' % ('bob',20)) #bob is 20 years old

print('%s is %d years old' % ('bob',20)) #%d 只能做数字

print('%10s%8s' % ('name','age')) #name左对齐且占10个宽度,age左对齐且占8个宽度

print('%10s%8s' % ('zdd',20)) #

print('%-10s%-8s' % ('zdd',20)) #加-号，zdd右对齐且占10个宽度,20右对齐且占8个宽度

## 4了解（进制，浮点）

print('%#o' % 10) #8进制

print('%#x' % 10) #16进制

print('%f' % (5 / 3)) #f为浮点数

print('%5.2f' % (5 / 3))

print('%+d' % 10) #正数前加+号

print('%+d' % -10) #负数不需要加,就是-号

## 5 format函数，位置参数

print('{} is {} years old'.format('bob',20)) #输出bob is 20 years old

print('{0} is {1} years old'.format(20,'bob')) #输出20 is bob years old

print('{1} is {0} years old'.format(20,'bob')) #输出bob is 20 years old #位置1的bob左对齐点10个宽度

print( '{1:<10}:{0:>8}'.format(20, 'bob')) # 位置1的bob左对齐点10个宽度，位置0的20右对齐8个宽度

print**('{} is {} years old'**.format**('bob'**,25**) )**print**('{1} is {0} years old'**.format**(**25,**'bob'))**print**('{:<10}{:<8}'**.format**('name'**,**'age'))**print**('{:>10}{:>8}'**.format**('name'**,**'age'))**#输出  
# bob is 25 years old  
# bob is 25 years old  
# name age  
# name age

## 6原始字符串

# win\_path = 'c:\temp'

# print(win\_path) #输出c: emp，\t被当做特殊字符

#

# win\_path = 'c:\\temp'

# print(win\_path) #输出c:\temp 需要多个\来转义

#

# wpath = r'c:\temp' #原始字符串,字符串中的字符是本身含义 #字符串使用次数特别多（背下来）

# print(wpath) #输出c:\temp 直接原样输出

## 7字符串常用方法：

### 1、去除空白字符

s1 =' hello world\n'

print(s1)

print(s1.strip()) #去除两边的空白字符，包括换行,tab,\t

print(s1.lstrip()) # 去除左边空白字符

print(s1.rstrip()) #去除右边空白字符，包括换行,tab,\t

### 2、切割字符

s2 = 'hello world ni hao'

print(s2) #输出hello world ni hao

print(s2.split()) #输出['hello', 'world', 'ni', 'hao']

s3 = 'hello-world-ni-hao'

print(s3.split()) #输出['hello-world-ni-hao']

print(s3.split('-')) #输出['hello', 'world', 'ni', 'hao'] 指定分隔符‘-’

### 3、拼接字符

slist = ['hello','world','in','hao']

print(slist) #输出['hello', 'world', 'in', 'hao']

print(''.join(slist)) #输出helloworldinhao

print(' '.join(slist)) #输出hello world in hao

print('--'.join(slist)) #输出hello--world--in--hao

### 4、对齐

s2 = 'hello world ni hao'

print(s2.center(50)) #输出 hello world ni hao

print(s2.center(50),'-') #输出 hello world ni hao -

print(s2.center(50,'-')) #输出 ----------------hello world ni hao----------------

print(s2.rjust(50,'-')) #输出 --------------------------------hello world ni hao

print(s2.ljust(50,'-')) #输出 hello world ni hao--------------------------------

### 5、判断开头、结尾

s2 = 'hello world ni hao'

print(s2.startswith('h')) #输出 True

print(s2.startswith('he')) #输出 True

print(s2.startswith('hea')) #输出 False

print(s2.endswith('o')) #输出 True

print(s2.endswith('ao')) #输出 True

print(s2.endswith('sao')) #输出 False

# subprocess模块调用系统命令

python可以使用subprocess模块调用系统命令

>>> import subprocess

>>> rc = subprocess.run('id root', shell=True) #调用系统id命令查询root用户是否存在

>>> rc.returncode # retruncode就是shell中的$?，返回查询的结果

>>>rc = subprocess.call('ping -c4 192.168.1.254 &>/dev/null' , shell=True)

>>>rc #执行成功rc=0 执行不成功rc=1

if rc:

print('192.168.1.254:down') #当re=1，执行

print(rc)

else:

print('192.168.1.254:up') #当re=0，执行

print(rc)

## @@创建用户程序

import sys

import subprocess

from randpass2 import gen\_pass #gen\_pass表示密码输入字符不显示，用于密码

def adduser(user, passwd, fname): #函数获取三个参数

info = '''用户信息

用户名:%s

密码:%s

''' % (user,passwd) #将按照此格式，将用户信息写入/tmp/users.txt文件中

subprocess.run('useradd %s' % user, shell=True) #调用系统命令创建用户

subprocess.run(

'echo %s | passwd --stdin %s' % (passwd,user), #调用系统命令给用户创建密码

shell=True

)

with open(fname,'a') as fobj: #用with方式，以a(追加)形式打开文件，

fobj.write(info) #最后将用户信息写入fname指定的文件中，with自动关闭文件

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

username = sys.argv[1] #username从位置函数获取

pwd = gen\_pass() #获取密码，输入时，隐藏不显示

filename = '/tmp/users.txt' #创建成功，将会把用户名和密码写入/tmp/users.txt文件中

adduser(username,pwd,filename) #执行函数，并传入username,pwd，filename三个参数

###20190415#####################

# 列表[list]:下标

属于容器类型

# from random import randint

# alist1 = [randint(1,100) for i in range(10)]

# print(alist1) #[21, 88, 62, 5, 91, 85, 40, 45, 36, 7]

#

# alist = [21, 88, 62, 5, 91, 85,]

# print(alist.append(30)) #None

# print(alist) #[21, 88, 62, 5, 91, 85, 30]

# print(alist.count(21)) #输出1,统计21在列表中出现的次数

# print(alist.index(21)) #输出0 统计21在列表中的下标

print(alist.index(3,4)) #从alist[4]开始往后搜索，返回3的下标

#

# alist.insert(5,101) #在列表中下标为5的位置插入101

# print(alist) #[21, 88, 62, 5, 91, 101, 85, 30]

#

# alist.reverse()

# print(alist) #[30, 85, 101, 91, 5, 62, 88, 21]

#

# alist.sort() #列表排序

# print(alist) #[5, 21, 30, 62, 85, 88, 91, 101]

#

# alist.remove(30) #删除列表

# print(alist) #[5, 21, 62, 85, 88, 91, 101]

#

# alist.pop() #踢出元素,默认是踢出最后一个,从后往前踢出

# print(alist) #[5, 21, 62, 85, 88, 91]

# alist.pop(2) #踢出下标为2的元素，有两个2的元素，踢出后面一个

# print(alist) #[5, 21, 85, 88, 91]

#

# blist = alist.copy() #将alist拷贝给blist

# print(blist) #[5, 21, 85, 88, 91]

# clist = alist #clist 与 alist 使用相同的地址符

# print(clist) #[5, 21, 85, 88, 91]

#

# alist.extend(clist) #将clist的内容汇入到alist

# print(alist) #[5, 21, 85, 88, 91, 5, 21, 85, 88, 91]

# alist.clear() #清空列表

# print(alist) #[]

ages1 = (11,22,33,44,55) #定义元组

print(list(ages1)) #list将ages1元组转换列表 [11, 22, 33, 44, 55]

# 元组(tuple):下标,不可变

不可变

a = (10)

print(type(a)) #<class 'int'> 说明a是个int整数

print(a) #10

b = (10,)

print(type(b)) #<class 'tuple'> 说明b是个元组

print(b) #(10,)

ages1 = (11,22,33,44,55) #直接定义  
print(list(ages1)) #list将ages1元组转换列表  
ages = tuple(list(ages1)) #将列表转换成元组  
print(ages)  
  
ages2 = tuple('abcd') #将字符串转换成元组  
print(ages2)

## 命名的元组

>>>Point = namedtuple('Point' , ['x','y','z'])

>>>a = Point(10,15,8)

>>>a[0]

>>>10

>>>a.x

>>>10

>>>a[1]

>>>15

>>>a.y

>>>15

>>>a[2]

>>>8

>>>a.z

>>>8

# 字典{dict}:无下标,无序,映射类型

## 1,工厂dict方法创建字典

adict = dict(['ab',['name','tom'],('age',22)])  
print(adict) #{'a': 'b', 'name': 'tom', 'age': 22}  
#dict的参数是个序列对象,序列中有三项,每一项又有两个项目,第一个项目是key,第二个是value

## 1创建具有相同值的默认字典bdict = {}.fromkeys(['tom','jerry','bob'],7) print(bdict) #{'tom': 7, 'jerry': 7, 'bob': 7}

## 2,访问字典

字典是映射类型,意味着没有下标.访问字典中的值需要使用相应的键

adict = dict(['ab',['name','tom'],('age',22)])

print(adict) #输出{'a': 'b', 'name': 'tom', 'age': 22}

for key in adict:

print(adict[key]) #输出所有的value

print('$s : $s' % (key, adict[key]))

#显示如下

a : b

name : tom

age : 22

## 判断

使用 in 和 not in 判断键是否存在于字典中

print('tom' in adict) #False  
print('name' in adict) #True

## 3更新添加字典

字典的kye都是唯一的,不重复的,字典的key必须是不可变类型

adict = dict(['ab',['name','tom'],('age',22)])

adict['name'] = 'jerry' #有name key则更新tom为jerry

print(adict) #{'a': 'b', 'name': 'jerry', 'age': 22}

adict['zdd'] = '123456' #没有zdd key 则添加新的

print(adict) #添加字典{'a': 'b', 'name': 'jerry', 'age': 22, 'zdd': '123456'}

## 4字典-get-检验key\*\*\*

adict = dict(['ab',['name','tom'],('age',22)]) #创建字典

print(adict.get('name', 'not found')) #字典中有name的key,返回tom,没有则返回’not found’

a = adict.get('qq') #字典中没有key为qq项,返回值None给a

print(a) #None

adict.get('qq','not found') #字典中没有key为qq项,返回值None

print(adict.get('qq','not found')) #not found

## 5字典-keys-所有key

adict = dict(['ab',['name','tom'],('age',22)])

b = adict.keys() #返回adict字典中所有的key

print(b) #dict\_keys(['a', 'name', 'age'])

print(adict.keys()) #dict\_keys(['a', 'name', 'age'])

## 6字典-values-返回所有值

adict = dict(['ab',['name','tom'],('age',22)])

print(adict) #{'a': 'b', 'name': 'tom', 'age': 22}

print(adict.values()) #dict\_values(['b', 'tom', 22]) 返回所有的value

## 7字典-items-返回所有

adict = dict(['ab',['name','tom'],('age',22)])

print(adict) #{'a': 'b', 'name': 'tom', 'age': 22}

print(adict.items()) #dict\_items([('a', 'b'), ('name', 'tom'), ('age', 22)])返回所有的(key,value)组成的集合

## 8字典-pop-弹出

adict = dict(['ab',['name','tom'],('age',22)])

print(adict) #{'a': 'b', 'name': 'tom', 'age': 22}

print(adict.pop('a')) #b #弹出key是a的项

print(adict.pop()) #默认弹出最后一个

## 9删除一个

d={'name':'tedu','age':18}

del d['age']

## 10删除所有

d={'name':'tedu','age':18}

d.clear()

## 11 字典函数

len():返回字典中元素的数目

hash()本省不是为字典设计的,但是可以判断某个对象是否可以作为字典的键

## 字典内建方法---待补充

dict={'name':'tedu','age':18}

dict.clear() 删除字典中所有元素

dict.copy() 返回字典（浅复制）的一个副本

dict.get(key)

dict.items() 返回一个包含字典中(键, 值)对元组的列表

dict.values() 返回一个包含字典中所有值的列表

dict.keys() 返回一个包含字典中键的列表

dict.pop(key[, default])

>>> myDict = {'name':'earth', 'port':'80'}

>>> myDict.pop('port')

'80'

>>> myDict

{'name': 'earth'}

>>> myDict.pop('port', 'No such key!')

'No such key!'

## @@@栈程序:方式一

#栈结构,类似于堆叠盘子,先放上去的堆在下方,先拿的都是最后放上去的

import random,sys

list\_a = []

def push\_it(): #进栈

print('压栈') #调试测试

itm = input('>').strip() #定义输入

if itm: #如果itm为非空则执行下面

list\_a.append(itm)

def pop\_it(): #出栈

print('出栈')

if list\_a: #如过list\_a列表为非空,

print('栈中弹出:%s' % list\_a.pop()) #输出的同时,列表中弹出最后一项

else:

print('空栈,无内容弹出')

def view\_it(): #查询

print('\033[31;1m%s\033[0m' % list\_a)

def show\_menu(): #选择菜单

prompt = '''0:压栈

1:出栈

2:查询

3:退出

请选择相应的操作(0/1/2/3):'''

while True:

xz\_1 = (input(prompt).strip()[0])

if xz\_1 not in ['0','1','2','3']:

print('无效,请重新输入')

continue

elif xz\_1=='0':

push\_it()

elif xz\_1=='1':

pop\_it()

elif xz\_1=='2':

view\_it()

else:

print('退出')

break

if \_\_name\_\_ == '\_\_main\_\_': #主程序

show\_menu()

## @@@栈程序,字典方式

栈结构,类似于堆叠盘子,先放上去的堆在下方,先拿的都是最后放上去的

import random,sys

list\_a = []

def push\_it(): #进栈

print('压栈') #调试测试

itm = input('>').strip() #定义输入

if itm: #如果itm为非空则执行下面

list\_a.append(itm)

def pop\_it(): #出栈

print('出栈')

if list\_a: #如果list\_a列表为非空,则执行下面

print('栈中弹出:%s' % list\_a.pop()) #输出的同时,列表中弹出最后一项

else:

print('空栈,无内容弹出')

def view\_it(): #查询

print('\033[31;1m%s\033[0m' % list\_a) #输出查询结果,并标红

def show\_menu(): #选择菜单

cmds = {'0':push\_it,'1':pop\_it,'2':view\_it} #定义字典

#采用字典方式:注意字典中函数名后不要加(),因为是把函数存入字典,

#并不是将函数结果存入字典,

prompt = '''0:压栈

1:出栈

2:查询

3:退出

请选择相应的操作(0/1/2/3):'''

while True:

# xz\_1 = (input(prompt).strip()[0]) #删除用户输入的两端空格,并取出第一个

# if xz\_1 not in ['0','1','2','3']:

# print('无效,请重新输入')

# continue

# elif xz\_1=='0':

# push\_it()

# elif xz\_1=='1':

# pop\_it()

# elif xz\_1=='2':

# view\_it()

# else:

# print('退出')

# break

choice = input(prompt).strip()[0] #删除用户输入的两端空格,并取出第一个

if choice not in ['0','1','2','3']:

print('无效,请重试:')

continue

if choice == '3':

print('退出')

break

cmds[choice]() #在字典中取出出入的数字对应的函数,加上后面的函数,即为调用函数

if \_\_name\_\_ == '\_\_main\_\_': #主程序

show\_menu()

## @@@模拟用户登录程序

#####模拟用用户登录系统

import getpass

userdb = {}

print(userdb)

def new():

print('正在注册')

user = input('请输入新用户名').strip()

if user and user not in userdb:

passwd = input('密码:')

userdb[user] = passwd

def old():

print('正在登录')

user = input('请输入用户名:').strip()

passwd = getpass.getpass('密码:') #定义输入的密码,且是不明文显示,可作为输入窗口

if userdb.get(user) == passwd:

#如果输入的用户不在userdb字典中,返回值将是None不会等于刚刚输入的密码

#或者用户存在,返回的value值不等于,刚刚输入的passwd,都不执行下面

print('登录成功')

else:

print('登录失败')

def menu():

cmds = {'0':new,'1':old}

xz='''0:新用户注册

1:老用户登录

2:退出

请选择(0/1):'''

while True:

choice = input(xz).strip()

if choice not in ['0','1','2']:

print('请重新选择')

continue

if choice == '2':

print('退出')

break

cmds [choice]()

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

menu()

# eval

功能：将字符串str当成有效的表达式来求值并返回计算结果。

语法： eval(source[, globals[, locals]]) -> value

参数：

　　　source：一个Python表达式或函数compile()返回的代码对象

　　　globals：可选。必须是dictionary

　　　ocals：可选。任意map对象

**可以把list(列表),tuple(元组),dict(字典)和string(字符串)相互转化。**

2 #################################################

## 字符串转换成列表

4 >>>a = "[[1,2], [3,4], [5,6], [7,8], [9,0]]"

5 >>>type(a)

6 <type 'str'>

7 >>> b = eval(a)

8 >>> print b

9 [[1, 2], [3, 4], [5, 6], [7, 8], [9, 0]]

10 >>> type(b)

11 <type 'list'>

12 #################################################

## 字符串转换成字典

14 >>> a = "{1: 'a', 2: 'b'}"

15 >>> type(a)

16 <type 'str'>

17 >>> b = eval(a)

18 >>> print b

19 {1: 'a', 2: 'b'}

20 >>> type(b)

21 <type 'dict'>

22 #################################################

## 字符串转换成元组

24 >>> a = "([1,2], [3,4], [5,6], [7,8], (9,0))"

25 >>> type(a)

26 <type 'str'>

27 >>> b = eval(a)

28 >>> print b

29 ([1, 2], [3, 4], [5, 6], [7, 8], (9, 0))

30 >>> type(b)

31 <type 'tuple'>

# linux与windows文件的转换

* windows文本文件的行结束标志是\r\n
* 类unix文本本间的行结束标志是\n

## @@@uninx2dos的转换程序

###uninx2dos的转换程序

import sys

def unix2dos(fname): #接收位置参数

dst\_fname = fname + '.txt' #重命名

with open(fname) as src\_fobj: #打开源文件

with open(dst\_fname,'w') as dst\_fobj: #以w方式打开目标文件,没有则创建

for line in src\_fobj: #遍历源文件

line = line.rstrip() + '\t\n' #重新获取line内容,去掉右边的空格,加上\t\n

dst\_fobj.write(line) #将line内容重新写入目标文件

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

unix2dos(sys.argv[1]) #位置参数

#转换前实际验证

# [root@room9pc01 python]# python3

# >>> f = open('/root/桌面/python/gen\_pass.py.txt')

# >>> f.readline()

# '#! /usr/bin/env python\n' #转换之前是\n结尾

#转换后实际验证

# [root@room9pc01 python]# python3 dayall.py gen\_pass.py

# [root@room9pc01 python]# ls #转换后多了gen\_pass.py.txt 文件

# dayall.py gen\_pass.py gen\_pass.py.txt home-100.py Python-3.7.3.tgz

# [root@room9pc01 python]# python3

# >>> f = open('/root/桌面/python/gen\_pass.py.txt')

# >>> f.readline()

# '#! /usr/bin/env python\t\n' #转换后是\t\n结尾了

# 进度条

## @@@类似进度条程序

**import** time  
n = 0  
print(**'#'** \* 20, end=**''**)  
**while True**:  
 print(**'\r%s@@@%s'** % (**'#'** \* n , **'#'** \* (19 - n)),end=**''**)  
 n +=1  
 **if** n == 20:  
 n = 0  
 time.sleep(0.3)

# 集合(set):不可变,无序

* 由不同元素构成,元素必须是不可变对象
* 常用来去重和关系运算
* 用{}表示,像是一个无值的字典,只有key

from random import randint

alist = [randint(1, 20) for i in range (10)]

print(alist) #十个1到20之间的随机数[13, 19, 12, 4, 20, 5, 13, 8, 16, 16]

aset = set(alist)

print(aset) #按序排列,并去重{4, 5, 8, 12, 13, 16, 19, 20}

aset = set('abc')

print(aset) #{'b', 'a', 'c'}

bset = set('bcd')

print(bset) #{'c', 'd', 'b'}

print(aset|bset) #并集{'b', 'c', 'a', 'd'}

aset.union(bset) # 并集

print(aset&bset) #交集{'b', 'c'}

aset.intersection(bset) # 交集

print(aset-bset) #差集{'a'} aset中有,bset中没有

aset.difference(bset) #差补

print(aset.add('abc')) #None 不能这么显示

aset.add('abc')

print(aset) #先添加在打印{'a', 'c', 'b', 'abc'}.

## 清空集合

a={3,"a",2.1,1}

print(a.clear()) # None

## 集合批量添加,删除

aset = set('abc')

print(aset) #{'b', 'a', 'c'}

aset.update('xyz') #批量添加元素

print(aset) #{'a', 'z', 'b', 'c', 'x', 'y'}

aset.update(['xxx','yyy','zzz'])

print(aset) #{'z', 'x', 'c', 'b', 'a', 'y', 'xxx', 'zzz', 'yyy'}

aset.remove('zzz')

print(aset) #{'xxx', 'c', 'yyy', 'x', 'b', 'z', 'y', 'a'}

## 超集,子集

cset = set('abcde')

dset = set('bcd')

cset.issuperset(dset) # cset是dset的超集么？

cset.issubset(dset) # cset是dset的子集么？

## @@@比较两个a,b两个文件,取出不同的内容

with open('/tmp/passwd1') as f1:

s1 = set(f1) #将第一个文件内容变成s1集合

with open('/tmp/passwd2') as f2:

s2 = set(f2) #将第二个文件内容变成s2集合

s3 = s2 - s1 #s3等于s2相对于s1的差集

with open('/tmp/result.txt','w') as f3: #打开一个新文件到f3

f3.writelines(s3) #将s3列表写入到f3中

###实际操作

**#passwd1文件内容**

root:x:0:0:root:/root:/bin/bash

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

**#passswd2文件内容**

root:x:0:0:root:/root:/bin/bash

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

ni hao a

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

root:x:0:0:root:/root:/bin/bash

zdd

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

roooasdft:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

**#程序运行后产生的新文件内容**

[root@room9pc01 tmp]# cat result.txt

roooasdft:x:0:0:root:/root:/bin/bash

ni hao a

zdd

# 迭代器

迭代器是访问集合元素的一种方式。迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。

## 延迟计算或惰性求值 (Lazy evaluation)

　　迭代器不要求你事先准备好整个迭代过程中所有的元素。仅仅是在迭代至某个元素时才计算该元素，而在这之前或之后，元素可以不存在或者被销毁。这个特点使得它特别适合用于遍历一些巨大的或是无限的集合。

## 可迭代对象

　　迭代器提供了一个统一的访问集合的接口。只要是实现了\_\_iter\_\_()或\_\_getitem\_\_()方法的对象，就可以使用迭代器进行访问。

　　序列：字符串、列表、元组

　　非序列：字典、文件

　　自定义类：用户自定义的类实现了\_\_iter\_\_()或\_\_getitem\_\_()方法的对象

## 创建迭代器对象

　　使用内建的工厂函数iter(iterable)可以获取迭代器对象：

　　语法：

　　　　iter(collection) -> iterator

　　　　iter(callable,sentinel) -> iterator

　　说明：

　　　　Get an iterator from an object.

　　　　In the first form, the argument must supply its own iterator, or be a sequence.

　　　　In the second form, the callable is called until it returns the sentinel.

## 实例展示：

1 使用对象内置的\_\_iter\_\_()方法生成迭代器

2 >>>L1 = [1,2,3,4,5,6]

3 >>>I1 = L1.\_\_iter\_\_()

4 >>>print I1

5 <listiterator object at 0x7fe4fd0ef550>

6 >>> I1.next()

7 1

8 >>> I1.next()

9 2

10 >>> I1.next()

11 3

1 使用内置工厂函数生成迭代器

2 >>> L1 = [1,2,3,4,5,6]

3 >>> I2 = iter(L1)

4 >>> print I2

5 <listiterator object at 0x7fe4fd0ef610>

6 >>> I2.next()

7 1

8 >>> I2.next()

9 2

10 >>> I2.next()

11 3

# time模块时间表示方式

## 1,时间戳,距离1970-1-1 0:00:00 之间的秒数

# import time

# print(time.time()) #1555318981.2148066

## 2,UTC时间:世界协调时

# print(time.ctime()) #Mon Apr 15 17:03:01 2019

## 3,struct\_time九元组

# print(time.localtime()) #localtime将一个时间戳转换为当前时区的struct\_time

# #time.struct\_time(tm\_year=2019, tm\_mon=4, tm\_mday=15, tm\_hour=17, tm\_min=3, tm\_sec=1, tm\_wday=0, tm\_yday=105, tm\_isdst=0)

# t = time.localtime()

# a=(t.tm\_year)

# print(a) #2019

# print(t.tm\_mon) #4

# ##需要掌握的时间表示方式

# print(time.time()) #时间戳

# print(time.strftime('%Y-%m-%d %H:%M:%S')) #显示固定格式2019-04-15 16:56:07

# print(time.strptime('2019-04-15 16:56:07','%Y-%m-%d %H:%M:%S'))

# #将时间字符串转换成struct\_time样式(九元组)

## strptime 和 strftime

# t1 = time.strptime('2019-04-15 16:56:07','%Y-%m-%d %H:%M:%S')

#将时间字符串转换成struct\_time样式(九元组)

# t2 = time.localtime() #时间比较

# print(t1 > t2)

# print(t2 > t1)

#

# print(time.strftime('%a %A')) #显示固定格式2019-04-15 16:56:07

## 时间睡眠

# time.sleep(3) #时间睡眠

## datetime

# import datetime

# t1 = datetime.datetime.now() #取出当前时间,年月日时分秒毫秒

# t2 = datetime.datetime(2019, 4, 10) #创建一个时间

# print(t1) #2019-04-15 17:13:02.111975

# print(t2) #2019-04-10 00:00:00

# print(t1 > t2) #True

#

# days = datetime.timedelta(days=100,hours=10)

# print(t1 - days) #100天零10十小时之前的时间2019-01-05 07:13:02.111975

# print(t1 + days) #100天零10小时之后的时间2019-07-25 03:13:02.111975

# 异常处理

# 错误代码

# try:

# num = int(input('number: '))

# result = 100 / num

# except (ValueError,ZeroDivisionError): #字母无法转换成整数.多个错误情况放一起必须加括号

# print('滚蛋,无效的数字1')

# # except ZeroDivisionError: #0无法作为被除数, 与上面可以一起

# # print('滚犊子,无效的数字2')

# except KeyboardInterrupt: #ctrl+c

# print('\n拜拜了您勒')

# exit() #程序遇到exit就会退出,后续代码不执行

# except EOFError: #ctrl+D

# print('\n会不会用?不会滚蛋')

# exit()

# else:

# print(result) #输出结果

# finally: #不管程序是否发生异常,都会执行

# print('Done')

######触发异常#####

def set\_age(name, age):

if not 0 < age < 120:

raise ValueError('年龄超出范围')

print('%s is %s years old.' % (name, age))

def set\_age2(name, age):

assert 0 < age < 120, '年龄超出范围' # 如果age不在此范围，一定发生AssertionError

print('%s is %s years old.' % (name, age))

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

set\_age2('杨晨', 24)

## 断言异常 \*\*待补充

断言，判断某个表达式的真假，真则程序继续执行，否则抛出 AssertionError 异常。

# OS模块

* import os #调用模块
* 对文件系统的访问大多通过os模块实现
* 该模块是python访问操作系统功能的主要接口
* 有些方法,os并没有提供,如copy等,可以使用shutil模块作为补充



os.fork() #分岔多进程

>>> os.getcwd() #查看当前目录

'/gitall/zdd-repository/zddpython2-day02'

>>> os.mkdir (test) #创建目录

>>> os.mkdir('/tmp/demo') #

>>> os.makedirs('/tmp/aaa/bbb/ccc') #相当于mkedir -P

>>> os.chdir('/tmp/demo') #相当于 cd

>>> os.getcwd()

'/tmp/demo'

>>> os.listdir() #相当于ls

[]

>>> os.makedirs('aaa/bbb/ccc')

>>> os.listdir()

['aaa']

>>> os.mknod('mytest') #相当于touch--mytest

>>> os.listdir() #ls

['aaa', 'mytest']

>>> os.symlink('/etc/hosts','zhuji') #相当于ln -s 创建软连接

>>> os.listdir()

['zhuji', 'aaa', 'mytest']

>>> os.unlink('zhuji') #删除软连接

>>> os.remove('mytest') #相当于rm -f

>>> os.path.isfile('/etc/hosts') #判断是否是文件

True

>>> os.path.isdir('/etc/ho') #判断是否是目录

False

>>> os.path.islink('/etc/hosts') #判断是否是软连接

False

>>> os.path.exists('/etc') #判断文件是否存在

True

>>> os.path.split('.etc/sysconfig/network') #将目录名和文件名分开

('.etc/sysconfig', 'network')

>>> os.path.join('/etc/sysconfig', 'network') #将目录名和文件名合并

'/etc/sysconfig/network'

>>> os.path.getsize('/etc/passwd') #获取文件大小

2460

**import** os  
fname **= 'http://cdn.tmooc.cn/tmooc-web/css/img/tmooc-logo2.png'**a **=** os.path.basename**(**fname**)**print**(**a**)**

#输出: tmooc-logo2.png

扩展:split还能

# pickle模块

* 把数据写入文件时,常规的文件方法只能把字符串对象写入,其他数据需要先转换成字符串再写入
* pickle模块,可以将任意内容存入文件,又可以把它完整无缺地取出来
* 之前的方法无法写入字典,列表,,,pickle就可以

pickle模块dump()存储 和 load()读取方法

>>> import pickle

>>> f = open('/tmp/myfile','wb')

>>> shop\_list = ['apple','egg']

>>> pickle.dump(shop\_list,f) #dump存储写入，将shop\_list列表写入f打开的文件目录中，

>>> f.close()

>>>

>>> with open('/tmp/myfile','rb') as fobj:

... mylist = pickle.load(fobj) #load读取

... #连续回车

>>>

>>> type(mylist)

<class 'list'>

>>>

>>> mylist #mylist列表用load读取的结果

['apple', 'egg']

>>>

[root@room9pc01 tmp]# cat myfile #myfile文件中有内容了,存入的是二进制

�XaqXbq�q.

## @@@记帐本程序

#! /usr/bin/env python

import os,pickle,time

def ruzhang(fname):

print('jz')

date = time.strftime('%Y-%m-%d')

rz\_je = int(input('入账金额：'))

rz\_sm = input('入账说明：')

with open(fname,'rb') as fobj:

data = pickle.load(fobj)

balance = data[-1][-2]+rz\_je

line = [date,rz\_je,0,balance,rz\_sm]

data.append(line)

with open(fname,'wb') as fobj:

pickle.dump(data,fobj)

def chuzhang(fname):

print('cz')

date = time.strftime('%Y-%m-%d')

cz\_je = int(input('出账金额：'))

cz\_sm = input('出账说明：')

with open(fname,'rb') as fobj:

data = pickle.load(fobj)

balance = data[-1][-2]-cz\_je

line = [date,0,cz\_je,balance,cz\_sm]

data.append(line)

with open(fname,'wb') as fobj:

pickle.dump(data,fobj)

def chaxun(fname):

print('cx')

print('%-10s%-8s%-8s%-10s%-20s' % ('登记时间','入账金额','出账金额','余额','说明'))

with open(fname,'rb') as fobj:

data = pickle.load(fobj)

for line in data:

print('%-14s%-13s%-11s%-11s%-20s' % tuple(line))

def view():

xz\_1 = '''0:入账

1:出账

2:查询

3:退出

请选择（0/1/2）：'''

cmds = {'0':ruzhang,'1':chuzhang,'2':chaxun}

fname = 'jzb.txt'

if not os.path.exists(fname):

date\_a = time.strftime('%Y-%m-%d')

data\_a = [

[date\_a,0,0,10000,'初始金额']

]

with open(fname,'wb') as fobj:

pickle.dump(data\_a,fobj)

while True:

xz\_2 = input(xz\_1).strip()

if xz\_2 == '3':

print('选择退出')

exit()

if xz\_2 not in ['0','1','2']:

print('无效输入')

continue

cmds[xz\_2](fname)

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

view()

# 函数

## 函数传参:

**def** myfunc(name, age=23):  
 **pass #pass 不做任何事情，一般用做占位语句。为了保持程序结构完整性***#函数中的name就被称作位置参数，age被称作关键字参数***def** get\_info(name, age):  
 print(**'%s is %s years old'** % (name, age))  
*#get\_info() # error，参数个数不足  
#get\_info('tom', 20, 30) # error, 参数个数太多*get\_info(**'tom'**, 20) *# OK #tom is 20 years old*get\_info(20, **'tom'**) *# OK，但是语义不对 20 is tom years old*get\_info(age=20, name=**'tom'**) *# OK #tom is 20 years old  
#get\_info(age=20, 'tom') # error, 位置参数必须在关键字参数前  
#get\_info(20, name='tom') # error, name得到了多个值*get\_info(**'tom'**, age=20) *# OK #tom is 20 years old*

## 函数参数组

通过把元组(非关键字参数)或者字典(关键字参数)作为参数组传递给函数

def func1(\*args): #args前面的一个\*号,表明args是元组,传参会把参数放入元组

print(args) #调用

def func2(\*\*kwargs): #kwargs前面两个\*\*号,表名kwargs是字典

print(kwargs) #调用

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

func1()

func1('hao')

func1('zdd',123)

func2()

func2(name='tom',age=20)

显示结果:

()

('hao',)

('zdd', 123)

{}

{'name': 'tom', 'age': 20}

## 参数前面加\*号拆开

print('abc')

print(\*'abc')

##print(\*\*'abc') #错误

print(\*[10,20,30])

print(\*(10,20,30))

显示结果:

abc

a b c

10 20 30

10 20 30

## 字典前面加两个\*\*号再传参

表示把字典拆成关键字参数

**def** info(name,age):  
 print(**'%s:%s'** % (name,age))  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 info(\*\*{**'name'**:**'tom'**,**'age'**:20})

显示结果:

tom:20

info(name=**'tom'**,age=20)

显示结果:

tom:20

## @@@简单的加减法数学游戏--有错误搜集

*###简单的加减法数学游戏:增加了错误收集 try***import** random  
**def** add(x,y):  
 **return** x+y  
**def** sub(x,y):  
 **return** x-y  
  
**def** exam():  
 cmds = {**'+'**:add,**'-'**:sub} *#定义函数选择字典* nums = [random.randint(1,100) **for** i **in** range(2)] *#随机生成两个数* nums.sort(reverse=**True**) *#重新排列,默认升序,这是降序* op = random.choice(**'+-'**) *#随机选择是+法还是-法* result = cmds[op](\*nums) *#将列表拆开,传参到add函数或者sub函数,进行运算* prompt = **'%s %s %s = '** % (nums[0],op,nums[1]) *#固定格式,第一个数 +/- 第二个数 =* counter = 0 *#用作循环次数计数* **while** counter < 3:  
 **try**:  
 answer = int(input(prompt)) *#用户输入答案时捕获错误* **except**: *#捕获所有错误操作,不按程序答对,或者答错三次不结束,不推荐* print(**'非法操作,请继续'**)  
 **continue  
  
 if** answer == result: *#如果输入的等于运算返回的结果* print(**'答对了'**)  
 **break** print(**'不对哦'**) *#如果输入的不等于算返回的结果,则执行* counter +=1  
 **else**:  
 print(**'%s%s'** % (prompt,result)) *#输出上面的固定格式和结果***def** main():  
 **while True**:  
 print(**'开始测智商了'**)  
 exam() *#调用函数* **try**:  
 y\_n = input(**'请选择是否继续(y/n)?'**).strip()[0] *#输入是否继续时捕获错误* **except** IndexError:  
 **continue  
 except** (KeyboardInterrupt,EOFError): *#ctrl+c 与 ctrl+d 的错误* y\_n = **'n'  
  
 if** y\_n **in 'nN'**:  
 print(**'\n测试结束'**)  
 **break  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

## 匿名函数(lambda)

* lambda关键字创造匿名函数
* 匿名是因为不需要以标准的def方式来声明,简化语句
* 一个完整的lambda “语句”代表了一个表达式,这个表达式的定义体必须和声明放在同一行
* 用于简单,能够在一行内表示的函数,,计算结果为返回值
* 使用方法: (函数名) = lambda 参数1,参数2 : 表达式　　　 例如: jiafa = lambda x,y:x+y

## @@@简单的加减法--lambda匿名函数方式

import random

def exam():

cmds = {'+':lambda x,y:x+y,'-':lambda x,y:x-y,'\*':lambda x,y:x\*y} #用匿名函数代替运算

nums = [random.randint(1,100) for i in range(2)] #随机生成两个数

nums.sort(reverse=True) #重新排列,默认升序,这是降序

op = random.choice('+-\*') #随机选择是+法还是-法

result = cmds[op](\*nums) #将列表拆开,传参到add函数或者sub函数,进行运算

prompt = '%s %s %s = ' % (nums[0],op,nums[1]) #固定格式,第一个数 +/- 第二个数 =

counter = 0 #用作循环次数计数

while counter < 3:

try:

answer = int(input(prompt)) #用户输入答案时捕获错误

except: #捕获所有错误操作,不按程序答对,或者答错三次不结束,不推荐

print('非法操作,请继续')

continue

if answer == result: #如果输入的等于运算返回的结果

print('答对了')

break

print('不对哦') #如果输入的不等于算返回的结果,则执行

counter +=1

else:

print('%s%s' % (prompt,result)) #输出上面的固定格式和结果

def main():

while True:

print('开始测智商了')

exam() #调用函数

try:

y\_n = input('请选择是否继续(y/n)?').strip()[0] #输入是否继续时捕获错误

except IndexError:

continue

except (KeyboardInterrupt,EOFError): #ctrl+c 与 ctrl+d 的错误

y\_n = 'n'

if y\_n in 'nN':

print('\n测试结束')

break

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

main()

## filter函数过滤(高阶函数)

调用一个布尔函数func来迭代遍历每个序列中的元素,返回一个使func返回值为trure的元素序列

from random import randint

def func1(x): #布尔函数,x%2要么是0 要么是1,

return x % 2 #x的值除以2取余,结果为0的是flase, 结果为1的是ture

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

nums = [randint(1,100) for i in range(10)] #定义10个100以内随机数列表,赋值给nums

print(nums)

result = filter(func1,nums) #将nums列表传参给程序

print(list(result))

result2 = filter(lambda x: x % 2,nums) #以lambda匿名函数方式,执行结果与上面一样

print(list(result2))

result3 = filter(lambda x: not x % 2,nums) #不在奇数列表中.求随机数中的偶数,

print(list(result3))

**显示结果:**

[51, 36, 78, 24, 77, 60, 71, 29, 21, 78]

[51, 77, 71, 29, 21] #

[51, 77, 71, 29, 21]

[36, 78, 24, 60, 78]

## map函数

from random import randint

def func1(x):

return x + 2

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

nums = [randint(1,100) for i in range(10)]

print(nums)

result = map(func1,nums) #将nums中的每个数字交给func1函数进行加工,返回结果

print(list(result))

result2 = map(lambda x: x + 2, nums) #将nums中每个数字交给lambda函数进行加工,返回结果

print(list(result2))

# #输出结果:

# [80, 1, 61, 44, 28, 37, 47, 39, 61, 83] #nums值

# [82, 3, 63, 46, 30, 39, 49, 41, 63, 85] #map给func1函数加工后

# [82, 3, 63, 46, 30, 39, 49, 41, 63, 85] #直接用lambda加工

# 高级函数应用

## 全局变量,局部变量

x = 10

def foo():

print(x)

foo()

def foo2():

x = 'hello' #局部变量也有变量x,它将会把全局变量x在此函数中遮盖住

print(x) #输出hello

foo2()

print(x) #输出10

def foo3():

global x #声明x为全局变量,将会覆盖之前的全局变量

x = 10000

print(x) #输出:10000

foo3()

print(x) #输出:10000

## 名字空间

i = 1

def f():

i = 2

def g():

print(i)

return g

func = f()

func()

print(i)

**分类**

* local，局部命名空间，每个函数所拥有的命名空间，记录了函数中定义的所有变量，包括函数的入参、内部定义的局部变量。
* global，全局命名空间，每个模块加载执行时创建的，记录了模块中定义的变量，包括模块中定义的函数、类、其他导入的模块、模块级的变量与常量。
* Built-in，python自带的内建命名空间，任何模块均可以访问，放着内置的函数和异常。

**搜索顺序依次是**:局部>全局>内建

**生命周期**

* local（局部命名空间）在函数被调用时才被创建，但函数返回结果或抛出异常时被删除。（每一个递归函数都拥有自己的命名空间）。
* global（全局命名空间）在模块被加载时创建，通常一直保留直到python解释器退出。
* Built-in（内建命名空间）在python解释器启动时创建，一直保留直到解释器退出。

## 偏函数(partial):

改造现有函数,生成新函数,

是将函数式编程的概念和默认参数以及可变参数结合在一起

一个带有多个参数的函数,如果其中某些参数基本上固定的,那么就可以通过偏函数为这些参数赋默认值

from functools import partial

def add(a,b,c,d,e):

return a + b + c + d + e

a = add(10,20,30,40,5) #执行add函数,结果赋值给a

print(a)

myadd = partial(add,10,20,30,40) #将前面四个参数给固定为默认值,将函数add生成新函数myadd

b = myadd(5) #执行 新函数myadd并新传参5,结果赋值给b,

print(b)

## @@@简单GUI程序,,简单测试

import tkinter

from functools import partial

window = tkinter.Tk()

lb = tkinter.Label(window, text="Hello World!", font="Arial 20")

# b1 = tkinter.Button(window, fg='white', bg='blue', text='Button 1')

MyButton = partial(tkinter.Button, window, fg='white', bg='blue')

b1 = MyButton(text='Button 1')

b2 = MyButton(text='Button 1')

b3 = MyButton(text='Button 1')

qb = MyButton(text='QUIT', command=window.quit)

lb.pack()

b1.pack()

b2.pack()

qb.pack()

# 递归函数

如果函数包含了对其自身的调用,该函数就是递归的.

## 递归算法:

5!=5x4x3x2x1

5!=5x4!

5!=5x4x3!

5!=5x4x3x2!

5!=5x4x3x2x1!

## @@@递归函数案例

def func1(x): #定义

if x == 1:

return 1

return x \* func1(x - 1)

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

print(func1(5))

## @@@快速排序函数

from random import randint

def paixu(item):

if len(item) < 2:

return item

px\_zj = item[0]

px\_zx = [] #比列表第一个数小的放到这个集合

px\_zd = [] #比列表第一个数打的放到这个集合

for a in item[1:]: #从列表第二个数开始遍历

if a <= px\_zj:

px\_zx.append(a) #把比第一个数小的添加到px\_zx列表

else:

px\_zd.append(a) #把比第一个数大的添加到px\_zd列表

return paixu(px\_zx) + [px\_zj] + paixu(px\_zd)

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

nums = [randint(1,100) for i in range(10)]

print(nums)

print(paixu(nums)) #调用函数,并将nums列表传参给item

# 生成器-yield

* 生成器本身是一个带yield语句函数
* 与迭代器相似,生成器以另外的方式来运作
* 一个函数或者子程序,只返回一次,当达到一个真正的返回或者函数结束没有更多的值返回,会报StopIteration异常

def my\_gen():

yield 'hello world'

num = 10 + 5

yield num

yield 100

yield 1

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

my\_gen()

a = my\_gen()

print(a.\_\_next\_\_())

print(a.\_\_next\_\_())

print(a.\_\_next\_\_())

print(a.\_\_next\_\_())

# print(a.\_\_next\_\_()) #报错,只定义了4个生成器

print() #空格

mg = my\_gen()

for i in mg: #实际使用中使用for循环

print(i)

#输出结果

hello world

15

100

1

hello world

15

100

1

# 内部函数

## 闭包

一个函数套着另外的函数

def deco(func):

def red():

return '\033[31;1m%s\033[0m' % func()

return red

def hello():

return 'hello world!'

def greet():

return '你好'

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

print(hello()) #普通调用

a = deco(hello) #将hello函数作为参数传递给deco ,deco的返回值是red函数

print(a()) #调用 a 函数,实际上是调用red函数

## 装饰器

def deco(func):

def red():

return '\033[31;1m%s\033[0m' % func()

return red

def hello():

return 'hello world!'

@deco #装饰器

def greet():

return '你好'

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

# print(hello()) #普通调用

a = deco(hello) #将hello函数作为参数传递给deco ,deco的返回值是red函数

print(a()) #调用 a 函数,实际上是调用red函数

print(greet())

# 模块和文件

import sys

print(sys.path) #模块导入的时候,将会在sys.path定义的位置下搜索模块

## 模块导入

import time

import os,sys #不推荐使用

from random import random,choice,randint

import pickle as p #导入的同时,给模块起别名

## 包

def pstar():

print('\*' \* 30)

import mypace.hi

## 相对导入模块

## 绝对导入模块

# 模块

模块支持从逻辑上组织python代码

当代码量变得相当大的时候,最好把代码分成一些有组织的代码段

代码段相互间有一定练习,可能是一个包含数据成员和方法的类,也可鞥是一组相关批次独立的操作,

## 搜索路径

1,模块导入的时候,将会在sys.path定义的位置下搜索模块

import sys

print(sys.path)

路径具体如下:

['/gitall/zdd-repository', '/root/PycharmProjects/day02', '/usr/local/lib/python36.zip', '/usr/local/lib/python3.6', '/usr/local/lib/python3.6/lib-dynload', '/usr/local/lib/python3.6/site-packages', '/root/bin/pycharm2017/pycharm-2017.3/helpers/pycharm\_matplotlib\_backend']

2如果希望自己编写的模块也能在任意位置被导入的两种方法:

方法1,把模块文件拷贝到site-packages/目录下

方法2,使用PYTHONPATH环境变量定义

[root@room9pc01 ~]# export PYTHOPATH=/var/ftp/nsd.... (目录路径为模块所在路径)

# 模块导入方法

方法1,import time #常用

方法2,import os,sys #不推荐使用

方法3,from random import random,choice,randint #常用

方法4,import pickle as p #导入的同时,给模块起别名

## 导入和加载

当导入模块时,模块的顶层代码会被执行

一个模块不管被导入(import)多少次,只会被加载(load)一次



## 从zip文件中导入



# 包

## 目录结构

包是一个有层次的文件目录结构,为平坦的名称空间加入有层次的组织结构

允许程序员把有联系的模块组合到一起

[root@room8pc16 day03]# mkdir mypack

[root@room8pc16 day03]# vim mypack/hi.py

hello = 'hello world'

def pstar():

print('\*' \* 30)

>>> import mypack.hi

>>> mypack.hi.hello

'hello world'

>>> mypack.hi.pstar()

如果使用的是python2，需要在目录下创建\_\_init\_\_.py，否则不能成为包

## 绝对导入模块



## 相对导入模块



# 内置模块

## hashlib模块(MD5哈希算法)

hashlib用来替换md5和sha模块,并使他们的API一致,专门提供hash算法

hash(哈希):是一种加密算法,可以理解为将原始数据,生成一串固定长度的乱码,只要原始数据相同,生成的乱码一定是相同,,,即使原始数据有微小的变化,生成的乱码一定不同.,且无法通过乱码推算出原始数据

用途：

(1) 存储加密的密码 # grep root /etc/shadow

(2) 校验文件的完整性

m = hashlib.md5() #创建文件,

m.update(data) #更新

m.hexdigest() #输出

[root@room8pc16 day03]# cp /etc/passwd /tmp/mima

[root@room8pc16 day03]# md5sum /etc/passwd /tmp/mima

574e11b1a648bc9669b07015ead96f47 /etc/passwd

574e11b1a648bc9669b07015ead96f47 /tmp/mima

包括的算法有：md5/sha

>>> import hashlib

>>> m = hashlib.md5(b'123456') #创建文件

>>> m.hexdigest() #输出

'e10adc3949ba59abbe56e057f20f883e'

>>> m1 = hashlib.md5()

>>> m1.update(b'12')

>>> m1.update(b'34')

>>> m1.update(b'56')

>>> m1.hexdigest()

'e10adc3949ba59abbe56e057f20f883e'

## @@@创建一个将文件生成哈希值的程序

import hashlib

import sys,pickle

def ahash(fname):

m = hashlib.md5() #创建md5文件

with open(fname,'rb') as fobj:

while True:

data = fobj.read(4096)

if not data:

break

m.update(data) #更新数据,将文件内容分4字节一次加入到m文件

return m.hexdigest() #输出完整文件的哈希值

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

try:

f = sys.argv[1]

print(ahash(f))

except IndexError:

print('错误,请输入需要检测的文件名')

except (FileNotFoundError,NameError):

print('请输入正确的文件名')

## tarfile模块(压缩包)

tarfile模块允许创建/访问tar文件, 同时支持gzip,bzip2格式

tar包、压缩、解压缩

### tarfile.open打包

打包解压过程

import os

import tarfile

os.chdir('/etc') #进入目录

os.mkdir('/pythonbackup') #创建目录

tar = tarfile.open('/pythonbackup/mytest.tar.gz','w:gz') #创建压缩文件

#开始使用gzip压缩.以w方式,会创建一个mytest.tar.ge文件,pythontest目录需提前创建

tar.add('hosts') #添加入要压缩的文件绝对路径

tar.add('/etc/hosts') #绝对路径,会目录结果一起打包

tar.add('security') #继续添加(这是目录)

tar.close()

### tarfile.open解压

#以上是压缩文件的全部步骤,以下是解压缩的全部过程

tar = tarfile.open('/pythonbackup/mytest.tar.gz','r:gz') #打开之前创建tar包,以r的方式

tar.extractall('/pythonbackup/mydemo') #解压缩到指定目录,目录没有则会创建

tar.close()

#先解压缩  
deploy\_dir **= '/var/www/deploy/'**tar **=** tarfile.open**(**app\_fname,**'r:gz')**tar.extractall**(**path**=**deploy\_dir**) #####**tar.close**()**

## zipfile模块(压缩包)

import zipfile

z = zipfile.ZipFile("day5.zip","w")

z.write("a")

### z.extractall解压：

z.extractall(“a”)

# os.walk 原理分析

>>> import os

>>> result = list(os.walk('/tmp/mydemo/security'))

>>> len(result) #统计列表有五项组成

5

>>> r1 = result[0] #第一项

>>> len(r1) #第一项中有3项组成

3

>>> r2 = result[1] #第二项

>>> len(r2) #第二项也是三项组成

3

>>> r1 #查看第一项如下(是个元组,那么os.walk获得的是五项元组组成的内容)

('/tmp/mydemo/security', ['console.apps', 'console.perms.d', 'limits.d', 'namespace.d'], ['access.conf', 'chroot.conf', 'console.handlers', 'console.perms', 'group.conf', 'limits.conf', 'namespace.conf', 'namespace.init', 'opasswd', 'pam\_env.conf', 'sepermit.conf', 'time.conf', 'pwquality.conf'])

#共有5个元组,第一个元组r1(就是result[0] )的三项分别是:

'/tmp/mydemo/security', #第一项文件路径字符串

['console.apps', 'console.perms.d', 'limits.d', 'namespace.d'], #第二项为第一项的路径中的目录列表

['access.conf', 'chroot.conf', 'console.handlers', 'console.perms', 'group.conf', 'limits.conf', 'namespace.conf', 'namespace.init', 'opasswd', 'pam\_env.conf', 'sepermit.conf', 'time.conf', 'pwquality.conf'] #第三项为第一项路径中的文件列表

>>> r2 #查看第二项如下

('/tmp/mydemo/security/console.apps', [], ['config-util', 'xserver', 'liveinst', 'setup'])

共有5个元组,第二个元组r2(就是result[1])的三项分别是:

'/tmp/mydemo/security/console.apps', #第一项文件路径字符串,该路经在r1的目录列表第一个

[], #第二项为第一项的路径中的目录列表

['config-util', 'xserver', 'liveinst', 'setup'] #第三项为第一项路径中的文件列表

**进行查看**

>>> r1[0]

'/tmp/mydemo/security'

>>> r2[0]

'/tmp/mydemo/security/console.apps'

>>> r1[1]

['console.apps', 'console.perms.d', 'limits.d', 'namespace.d']

>>> r2[1]

[]

>>> r1[2]

['access.conf', 'chroot.conf', 'console.handlers', 'console.perms', 'group.conf', 'limits.conf', 'namespace.conf', 'namespace.init', 'opasswd', 'pam\_env.conf', 'sepermit.conf', 'time.conf', 'pwquality.conf']

>>> r2[2]

['config-util', 'xserver', 'liveinst', 'setup']

os.walk()得到的每一项都是元组，每个元组由三项构成，这三项分别是：路径字符串、

该路径下的目录列表、该目录下文件的列表

>>> for path, folders, files in os.walk('/tmp/mydemo/security'): #遍历walk形成的所有元组

... for file in files: #再遍历所有文件名

... os.path.join(path, file) #合并路径与文件名

path:得到所有的路径名

folders:得到所有的路径中的所有目录名

files:得到所有的文件名

# @@@备份程序

###具有完全备份和增量备份的程序  
**from** time **import** strftime  
**import** os  
**import** tarfile  
**from** check\_md5 **import** check\_md5  
**import** pickle  
  
#定义全量备份函数  
**def full\_backup(**src,dst,md5file**):** #给完全备份的tar包命名  
 # fname = os.path.basename(src) #只获取目标的文件名,可结合到下面一条语句了  
 fname **= '%s\_full\_%s.tar.gz' % (**os.path.basename**(**src**)**, strftime**('%Y%m%d'))** #取名  
 fname **=** os.path.join**(**dst,fname**)** #将备份的tar包名拼接绝对路径  
  
 #打包文件  
 tar **=** tarfile.open**(**fname,**'w:gz')** #打包开始  
 tar.add**(**src**)** #添加要打包的文件  
 tar.close**()** #计算每个文件的md5值  
 md5dict **= {}** #准备用来存放哈希值的列表  
 **for** path, folders, files **in** os.walk**(**src**):** #os.walk方式来获取文件路径和文件名,遍历元组,详情请看walk的详解  
 **for** file **in** files**:** #再遍历得到的所有文件名  
 key **=** os.path.join**(**path, file**)** #拼接所有文件的路径/文件名  
 md5dict**[**key**] =** check\_md5**(**key**)** #经过check\_md5程序产生的哈希值赋值给字典md5dict  
 #自编写的check\_md5模块必须与本程序文件放同一目录,该模块作用是生成哈希值  
 #将字典写入文件  
 **with** open**(**md5file,**'wb') as** fobj**:** #以wb方式打开md5file指定的文件,没有则创建  
 pickle.dump**(**md5dict,fobj**)** #pickle是能将任何内容存入文件中,将字典mdict的内容写入md5file中  
  
**def incr\_backup(**src,dst,md5file**):** #给备份的tar包命名  
 # fname = os.path.basename(src)  
 fname **= '%s\_incr\_%s.tar.gz' % (**os.path.basename**(**src**)**,strftime**('%Y%m%d'))** fname **=** os.path.join**(**dst,fname**)** #计算每个文件的md5值  
 md5dict **= {}** #将用来装哈希值  
 **for** path , folders, files **in** os.walk**(**src**):  
 for** file **in** files**:** key **=** os.path.join**(**path,file**)** md5dict**[**key**] =** check\_md5**(**key**)  
  
 with** open**(**md5file,**'rb') as** fobj**:** old\_md5 **=** pickle.load**(**fobj**)  
  
 with** open**(**md5file,**'wb') as** fobj**:** pickle.dump**(**md5dict,fobj**)** tar **=** tarfile.open**(**fname,**'w:gz')  
 for** key **in** md5dict**:** #遍历md5dict字典,返回key值  
 **if** md5dict**[**key**] !=** old\_md5.get**(**key**):** tar.add**(**key**)** tar.close**()  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** src **= '/pythonbackup/mydemo/security'** #定义要备份的源目录  
 dst **= '/pythonbackup/demo'** #定义放备份文件的目录  
  
 md5file **= '/pythonbackup/demo/md5.data'** #定义校验目录  
 **if** strftime**('%a') == 'Mon':** #判断是不是星期三  
 full\_backup**(**src,dst,md5file**)** #调用全量备份函数  
 **else:** incr\_backup**(**src,dst,md5file**)** #调用增量备份函数

# OOP简介

oop:面向对象的编程

## 基本概念:

类(class):描述具有相同的属性和方法的对象集合,定义该集合中每个对象所共有的属性和方法,对象是类的实例

实例化:创建一个类的实例,类的具体队形.

方法:类中定义的函数.

对象:通过类定义的数据结构实例.对象包括两个数据成员(类变量和实例变量)和方法.

创建类,创建实例

## 绑定方法

### 构造器方法创建类

* 当实例化类的对象时,构造器方法默认自动调用
* 实例本身作为第一个参数,传递给self

class Warriot:

def \_\_init\_\_(self,name,weapon): #\_init\_是构造器方法,当实例化时自动调用,不是必需的

self.name = name

self.weapon = weapon

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

lb = Warriot('吕布','方天画戟')

#lb实例,自动调用\_init\_ .lb 与 self 对应,吕布与name对应,方天画戟与weapon对应

zdd = Warriot('zdd','杀猪刀')

print('%-8s %s' % (lb.name , lb.weapon))

print('%-8s %s' % (lb.name, zdd.weapon))

print('%-8s %s' % (zdd.name , lb.weapon))

#实例化时,实例自动作为第一个参数传递给\_init\_方法

#self只是一个变量名,名字也可以是其他名字,python中习惯起名self

#拥有self(绑定到实例的属性),

#输出结果:

吕布 方天画戟

吕布 杀猪刀

zdd 方天画戟

**绑定:绑定一些属性到实例上**

### 其他绑定方法

**class Warriot:  
 def \_\_init\_\_(**self,name,weapon**):** #\_init\_初始化实例属性,构造器方法,当实例化时自动调用  
 self.name **=** name  
 self.weapon **=** weapon  
  
 **def speak(**self,content**):** #创建类中函数,叫做方法,设置content参数  
 print**('%s: 我手持%s, %s' % (**self.name, self.weapon, content**))  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** lb **=** Warriot**('吕布'**,**'方天画戟')**

#lb实例,自动调用\_init\_ .lb 与 self 对应,吕布与name对应,方天画戟与weapon对应  
 zdd **=** Warriot**('zdd'**,**'杀猪刀')** print**(**lb.name , lb.weapon**)** #吕布 方天画戟  
 print**(**zdd.name, zdd.weapon**)** #zdd 杀猪刀  
 print**()** lb.speak**('人在塔在')** #调用speak方法,并传参给content  
 zdd.speak**('%s你就是渣渣,老子拔你网线' %** lb.name**)** lb.speak**('......')**# ###实际输出  
# 吕布 方天画戟  
# zdd 杀猪刀  
#  
# 吕布: 我手持方天画戟, 人在塔在  
# zdd: 我手持杀猪刀, 吕布你就是渣渣,老子拔你网线  
# 吕布: 我手持方天画戟, ......

## 组合和派生

#两个不同的类,一个类是另一个类的组件

* 类被定义后,目标就是要把它当成一个模块来使用,并把这些对象嵌入到你的代码中去
* 组合就是让不同的类混合并加入到其他类中来增加功能和代码重用性
* 可以在一个大点的类中创建其他类的实例,实现一些其他属性和方法来增强对原来的类对象

##老师代码game2.py,###这种方法不常用,常用的请看下一种方式  
**class Weapon:  
 def \_\_init\_\_(**self, name, strength**):** self.name **=** name  
 self.strength **=** strength  
  
**class Warrior:  
 def \_\_init\_\_(**self,name,weapon**):** #\_init\_初始化实例属性,构造器方法,当实例化时自动调用  
 self.name **=** name  
 self.weapon **=** weapon

#weapon参数接收的是Weapon('方天画戟',100),所以又调用上面的Weapon方法  
  
 **def speak(**self,content**):** #创建类中函数,方法,设置content参数  
 print**('%s: 我手持%s, %s' % (**self.name, self.weapon, content**))  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** w **=** Weapon**('方天画戟'**,100**)** print**(**w.name,w.strength**)** #输出:方天画戟 100  
  
 lb **=** Warrior**('吕布'**, w**)** #lb实例,传参为weapon是传的'吕布和Weapon('方天画戟',100)'  
 # print(lb.weapon)   
 print**(**lb.weapon.name, lb.weapon.strength**)** #输出:方天画戟 100

### 组合常用方式

##########上面的代码进化成下面(老师代码game3.py)  
  
**class Weapon:  
 def \_\_init\_\_(**self, name, strength**):** self.name **=** name  
 self.strength **=** strength  
  
**class Warrior:  
 def \_\_init\_\_(**self,name,wname,wstrength**):** #修改地方  
 self.name **=** name  
 self.weapon **=** Weapon**(**wname,wstrength**)**

#修改地方,wname,wstrength传给上面Weapon方法  
 **def speak(**self,content**):** #创建类中函数,方法,设置content参数  
 print**('%s: 我手持%s, %s' % (**self.name, self.weapon, content**))  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** lb **=** Warrior**('吕布'**,**'方天画戟'**, 100**)** print**(**lb.weapon**)** #输出:<\_\_main\_\_.Weapon object at 0x7fc0c3ae87f0>是对象  
 print**(**lb.name, lb.weapon.name, lb.weapon.strength**)** #输出:吕布 方天画戟 100

## 父类与子类

* oop(面向对象的编程)的更强大方面之一是能够使用一个已经定义好的类,扩展它或者对其进行修改,而不会影响系统中使用现存类的其他代码片段,
* ood(面向对象设计)允许类特征在子孙类或子类中进行继承
* 继承:子类继承了父类的所有属性,不管是数据属性还是方法
* 继承覆盖:如果子类中有和父类同名的方法,父类方法将被覆盖
* 如果要访问父类的方法,则要调用一个未绑定的父类方法,明确给出子类的实例.

创建子类只需要在类名后面写明父类即可

####老师game4.py#子类与父类###########################  
**class Warrior:  
 def \_\_init\_\_(**self,name,weapon**):** #\_init\_初始化实例属性,构造器方法,当实例化时自动调用  
 self.name **=** name  
 self.weapon **=** weapon  
  
  
 **def speak(**self,content**):** #创建类中函数,方法,设置content参数  
 print**('%s: 我手持%s, %s' % (**self.name, self.weapon, content**))  
  
  
class MaleWarriot(**Warrior**):** #括号中指定父类(基类),MaleWarriot子类继承了父类所有属性  
 **def attcck(**self**):** print**('attack(攻击)')  
  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** lb **=** MaleWarriot**('吕布'**,**'方天画戟')** print**(**lb.name , lb.weapon**)** #输出吕布 方天画戟  
  
 lb.speak**('aaa')**

#输出: 吕布: 我手持方天画戟, aaa,先子类MaleWarriot中,找speak方法,找不到就到父类去找  
  
 lb.attcck**()** #输出:attack(攻击) 子类拥有父类与子类的所有功能

## 多重继承:

python允许多重继承,即一个类可以是多个父类的子类,子类可以拥有所有父类的属性

**class A:  
 def func1(**self**):** print**('a func')  
  
 def func(**self**):** print**('aaaaaaaa')  
  
class B:  
 def func2(**self**):** print**('b func')  
  
 def func(**self**):** print**('BBBBBBBBBB')  
  
class C(**B, A**):** # 父类可以有多个  
 **def func3(**self**):** print**('c func')  
  
 def func(**self**):** print**('CCCCCCCCCCC')  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** c1 **=** C**()** # c1具有ABC三个类的方法  
 c1.func1**()** c1.func2**()** c1.func3**()** c1.func**()** # 同名方法查找的顺序是自下向上，自左向右  
#显示结果:  
# a func  
# b func  
# c func  
# CCCCCCCCCCC

## 三个魔法方法

### \_\_init\_\_ 方法

实例化类实例时默认会调用的方法,具体请参阅”出版商程序”,创建实例时自动调用\_\_init\_\_方法

### \_\_str\_\_ 方法

打印/显示实例时调用方法, print(实例名) 自动调用\_\_str\_\_方法

返回字符串

具体请参阅”出版商程序”

### \_\_call\_\_ 方法

用于创建可调用的实例,具体请参阅”出版商程序” 实例名()自动调用\_\_call\_\_方法

## @@@出版商程序

**class Book:  
 def \_\_init\_\_(**self,title,author**):** self.title **=** title  
 self.author **=** author  
  
 **def \_\_str\_\_(**self**):  
 return '<%s>' %** corepy.title  
  
 **def \_\_call\_\_(**self**):** print**('<%s> is written by %s ' % (**self.title , self.author**))  
  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** corepy **=** Book**('Python核心编程'**, **'Wesley')** #调用\_\_init\_\_方法  
 print**(**corepy**)** #自动调用\_\_str\_\_方法,打印实例, 输出:<Python核心编程>  
 corepy**()**

#自动调用\_\_call\_\_方法,使实例像函数一样,可以调用使用

输出:<Python核心编程> is written by Wesley  
  
##显示结果  
# <Python核心编程>  
# <Python核心编程> is written by Wesley

# 正则



- 是范围,如果要在[]里匹配-符号 , -两边不能有字符 [0-9]范围, [09-]匹配09和-

^是开头意思,如果在[]里面的开头表示取反, [^0-9]取反 [0-9^]表示0-9和^符号

\w 匹配任意数字字母字符,与[0-9a-zA-Z]同义 \W则取反

\s 匹配空白字符,与[\r\v\f\t\n]同义 \S取反



<https://jex.im/regulex/> 正则解释网站

/t\*m 匹配m前面的t 出现零到多次

/t\+m 匹配m前面的t 出现1到多次

/t\?m 匹配m前面的t 出现0到1次 如 ttm 只匹配tm



/\<tom\> 匹配独立单词tom, 如果是tommorrow就不能匹配

## 正则经典案例

### 替换:将mac地址改为通过冒号分开的样式

192.168.1.1 000C29123456

192.168.1.3 525400A9B321

192.168.1.10 525400243D4A

思路：

1、先匹配到mac地址: 出现在结尾的12个任意字符

2、将mac地址分成6组: 每两个字符用括号()分组

3、在6组之间加上冒号: 替换时，\1表示第1个分组，\n表示第n个分组

:%s/\(..\)\(..\)\(..\)\(..\)\(..\)\(..\)$/\1:\2:\3:\4:\5:\6/

### 匹配IP地址

^(\d+\.){3}\d+

### 匹配网站上的图片连接

http:.\*\.(png|jpg

# re模块(核心函数和方法)

import re

## match函数

#match函数:从字符串开头开始匹配,匹配成功,则返回一个匹配对象,否则返回None  
a **=** re.match**('f..'**,**'food')** #从头开始匹配,  
print**(**a**)** #输出:<\_sre.SRE\_Match object; span=(0, 3), match='foo'>  
  
b **=** re.match**('f..'**,**'seafood')** #未匹配到,返回None  
print**(**b**)** #输出:None

## search函数

###search函数:在字符串中查找正则表达式模式的第一次出现,如果匹配成功,则返回一个匹配对象,否则返回None  
c **=** re.search**('f..'**,**'seafood')** #搜索  
print**(**c.group**())** #输出:foo 匹配对象的group方法获取匹配内容

## group方法:

##group方法:使用match或search匹配成功后,返回的匹配对象可以通过group方法获得匹配内容

## findall函数:

####findall函数:在字符串中查找正则表达式模式的所有(非重复)出现,返回一个匹配对象的列表  
d **=** re.findall**('f..'**,**'seafood')** #返回所有匹配内容的列表  
print**(**d**)** #输出:['foo'] --列表  
  
e **=** re.findall**('f..'**,**'seafood is food')**print**(**e**)** #输出:['foo', 'foo'] --列表

## finditer函数:

####finditer函数:和findall函数有相同的功能,但返回的不是列表而是迭代器,对于每个匹配,该迭代器返回一个匹配对象  
f **=** re.finditer**('f..'**,**'seafood is food')** #返回的是由匹配对象构成的生成器  
print**(**f**)** #输出<callable\_iterator object at 0x7f13b3b67630>  
**for** i **in** f**:** print**(**i.group**())**输出结果  
foo  
foo

## split方法:

###split方法:根据正则表达式中的分隔符把字符分隔为一个列表,并返回成功匹配的列表  
###字符串也有类似的方法,但是正则表达式更加灵活  
g **=** re.split**('-|\.'** , **'hello-world.tar.ge')**print**(**g**)** #输出:['hello', 'world', 'tar', 'ge']

##将横杠- 或者点.当做分隔符,将字符分隔,以列表方式输出

## sub方法:

####sub方法:把字符串中所有匹配正则表达式的地方替换成新的字符串  
h **=** re.sub**('X'**,**'zdd'**,**'Hi X . How art you X')** #将目标里的X全部换成zdd  
print**(**h**)** #输出:Hi zdd . How art you zdd

##将所有大写’X’,替换成zdd

## compile函数

###compile函数:对正则表达式模式进行编译,返回一个正则表达式对象  
##不是必须要用这种方式,但是在大量匹配的情况下,可以提升效率  
i **=** re.search**('f..'** , **'seafood')**print**(**i.group**())** #输出:foo,如下也可以用compile函数  
patt **=** re.compile**('f..')** #模式先编译,可提升效率,建议做法  
i **=** patt.search**('seafood')**

#编译后的对象也有search(返回第一个匹配)/findall(返回列表)等方法  
print**(**i.group**())** #输出:foo

##compile() 函数将一个字符串编译为字节代码。

## 贪婪匹配

##贪婪匹配: 默认情况下,\* +总是尽量多的匹配  
# # \* +和? 都是贪婪匹配操作符,在其后加上?可以取消其贪婪匹配行为  
# # 正则表达式匹配对象通过groups函数获取子组  
  
j **=** re.search**('.+(\d+)'**,**'his phone number is :15011223345')**print**(**j.group**())** #j.group总是匹配全部的模式.+(\d+) #输出:his phone number is :15011223345  
  
print**(**j.group**(**1**))** #匹配第1个()中的内容  
# print(j.group(2)) #报错  
#说明:.+是贪婪匹配,它尽可能匹配更多,的内容，\d+至少需要一个数字，所以.+给\d+留了  
  
###如果希望\d+能匹配更多的内容，使用?来取消贪婪匹配，让\d+匹配更多内容 如下  
k **=** re.search**('.+?(\d+)'** , **'his phone number is :15011223345')**print**(**k.group**())** #输出:his phone number is :15011223345  
  
print**(**k.group**(**1**))** #输出:15011223345

## @@@分析apache访问日志--函数方式

###分析apache访问日志  
**import** re  
  
**def count\_patt(**fname,patt**):** patt\_dict **= {}** cpatt **=** re.compile**(**patt**)** #编译,patt由参数传过来,ip或者br的正则  
 **with** open**(**fname**) as** fobj**:** #打开文件  
 **for** line **in** fobj**:** #遍历这个文件,获得每一行的内容  
 m **=** cpatt.search**(**line**)**

#经过上面编译后,在这search每一行,返回符合ip正则或者br正则  
 **if** m**:** # 如果匹配到内容，放到下面key变量，None表示False  
 key **=** m.group**()** patt\_dict**[**key**] =** patt\_dict.get**(**key,0**) +** 1  
 **return** patt\_dict  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** fname **= 'access\_log'** #access\_log为apache日志文件  
 ip **= '^(\d+\.){3}\d+'** # 1.11.123.45, 12345.11111.23423.2535234  
 br **= 'Chrome|Firefox|MSIE'** print**(**count\_patt**(**fname,ip**))** print**(**count\_patt**(**fname,br**))**

## @@@分析apache访问日志--oop方式

**本方式采用:通过对象直接调用被封装的内容**

分析apache访问日志--oop方式---通过对象直接调用被封装的内容  
# #1统计每个客户访问apache服务器的次数  
# #2将统计信息通过字典方式显示出来  
# #3分别统计客户端是firefox和msie的访问次数  
# #4分别使用函数式编程和面向对象编程的方式实现  
**import** re  
**class wenjianfenxi:** #创建wenjianfenxi类  
  
 **def count(**self,fname,patt**):** #间接调用被封装的内容  
 #类中定义的函数叫做“方法”：count方法  
 # print(self.name)  
 # print(self.patt)  
 patt\_dict **= {}** cpatt **=** re.compile**(**patt**)** #编译,self.patt = patt由参数传过来,ip或者br的正则  
 **with** open**(**fname**) as** fobj**:** #打开文件，self.name = fname  
 **for** line **in** fobj**:** #遍历这个文件,获得每一行的内容  
 m **=** cpatt.search**(**line**)** #经过上面编译后,在这search每一行,返回符合ip正则或者br正则  
 **if** m**:** # 如果匹配到内容，放到下面key变量，None表示False  
 key **=** m.group**()** patt\_dict**[**key**] =** patt\_dict.get**(**key,0**) +** 1  
 **return** patt\_dict  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** fname **= 'access\_log'** ip **= '^(\d+\.){3}\d+'** # 1.11.123.45, 12345.11111.23423.2535234  
 br **= 'Chrome|Firefox|MSIE'** lb **=** wenjianfenxi**()** print**(**lb.count**(**fname,ip**))** print**(**lb.count**(**fname,br**))**

## @@@分析apache访问日志--oop方式(自己写)

**本方式采用--通过self间接调用被封装的内容**

###分析apache访问日志--oop方式  
#1统计每个客户访问apache服务器的次数  
#2将统计信息通过字典方式显示出来  
#3分别统计客户端是firefox和msie的访问次数  
#4分别使用函数式编程和面向对象编程的方式实现  
**import** re  
**class wenjianfenxi:** #创建wenjianfenxi类  
 **def \_\_init\_\_(**self,fname,patt**):** #类中定义的函数叫做“方法”：  
 #\_\_init\_\_ 为初始化实例属性  
 self.name **=** fname  
 self.patt **=** patt  
  
 **def count(**self**):** #间接调用被封装的内容  
 #类中定义的函数叫做“方法”：count方法  
 # print(self.name)  
 # print(self.patt)  
 patt\_dict **= {}** cpatt **=** re.compile**(**self.patt**)** #编译,self.patt = patt由参数传过来,ip或者br的正则  
 **with** open**(**self.name**) as** fobj**:** #打开文件，self.name = fname  
 **for** line **in** fobj**:** #遍历这个文件,获得每一行的内容  
 m **=** cpatt.search**(**line**)** #经过上面编译后,在这search每一行,返回符合ip正则或者br正则  
 **if** m**:** # 如果匹配到内容，放到下面key变量，None表示False  
 key **=** m.group**()** patt\_dict**[**key**] =** patt\_dict.get**(**key,0**) +** 1  
 **return** patt\_dict  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** fname **= 'access\_log'** ip **= '^(\d+\.){3}\d+'** # 1.11.123.45, 12345.11111.23423.2535234  
 br **= 'Chrome|Firefox|MSIE'** lb\_ip **=** wenjianfenxi**(**fname,ip**)** print**(**lb\_ip.count**())** # 根据wenjianfenxi类创建对象lb\_ip对象，将fname,ip封装到lb\_ip的fname 和 patt中  
  
 lb\_br **=** wenjianfenxi**(**fname,br**)** print**(**lb\_br.count**())** # 根据wenjianfenxi类创建对象lb\_br对象，将fname,br封装到lb\_ip的fname 和 patt中

## @@@分析apache访问日志--oop方式(张志刚)

# ###分析apache访问日志--oop方式---通过self间接调用被封装的内容222  
**import** re #正则模块  
**from** collections **import** Counter #调用Counter统计排序模块  
  
**class wenjianfenxi:** #创建wenjianfenxi类  
 **def \_\_init\_\_(**self,fname**):** #较上一种方法,这里少定义了patt参数  
 #\_\_init\_\_ 为初始化实例属性  
 self.name **=** fname  
  
  
 **def count(**self,patt**):** #较上一种方法,这里多定义了patt参数  
 #类中定义的函数叫做“方法”：count方法  
 patt\_dict **= {}** cpatt **=** re.compile**(**patt**)** #较上一种方法,这里只定义了patt,不是调用self.patt参数了  
 **with** open**(**self.name**) as** fobj**:** #打开文件，self.name = fname  
 **for** line **in** fobj**:** #遍历这个文件,获得每一行的内容  
 m **=** cpatt.search**(**line**)** #经过上面编译后,在这search每一行,返回符合ip正则或者br正则  
 **if** m**:** # 如果匹配到内容，放到下面key变量，None表示False  
 key **=** m.group**()** patt\_dict**[**key**] =** patt\_dict.get**(**key,0**) +** 1  
 **return** patt\_dict  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** fname **= 'access\_log'** ip **= '^(\d+\.){3}\d+'** # 1.11.123.45, 12345.11111.23423.2535234  
 br **= 'Chrome|Firefox|MSIE'** lb\_ip **=** wenjianfenxi**(**fname**)** # 根据wenjianfenxi类创建对象lb\_ip对象，将fname,ip封装到lb\_ip的fname 和 patt中  
 print**(**lb\_ip.count**(**ip**))** #较上一种方法,这里使用对象传参时,添加了新的变量ip.这个变量给了count方法的patt  
 print**(**lb\_ip.count**(**br**))** #较上一种方法,这里使用对象传参时,添加了新的变量br.这个变量给了count方法的patt  
 #较上一种方法,这里少创建一个对象

# Counter统计排序模块

**from** collections **import** Counter ##排序函数  
c **=** Counter**()**c.update**(['192.168.1.1'])**c.update**(['192.168.1.1'])**c.update**(['192.168.1.20'])**c.update**(['zdd'])**c.update**(['zdd'])**c.update**(['zdd'])**c.update**(['zdd123'])**print**(**c**)**#输出:Counter({'zdd': 3, '192.168.1.1': 2, '192.168.1.20': 1, 'zdd123': 1})  
#自动按出现次数从多到少排序  
print**(**c.most\_common**())**#输出:[('zdd', 3), ('192.168.1.1', 2), ('192.168.1.20', 1), ('zdd123', 1)]  
print**(**c.most\_common**(**2**))** #将排在前的2个输出  
#输出[('zdd', 3), ('192.168.1.1', 2)]  
print**(**c.most\_common**(**3**))** #将排在前的3个输出  
#输出:[('zdd', 3), ('192.168.1.1', 2), ('192.168.1.20', 1)]

## @@@分析apache访问日志--oop方式(张志刚)

**添加了Counter统计排序模块**

####改进  
  
**import** re #正则模块  
**from** collections **import** Counter #调用Counter统计排序模块  
  
**class wenjianfenxi:** #创建wenjianfenxi类  
 **def \_\_init\_\_(**self,fname**):** #较上一种方法,这里少定义了patt参数  
 #\_\_init\_\_ 为初始化实例属性  
 self.name **=** fname  
  
  
 **def count(**self,patt**):** #较上一种方法,这里多定义了patt参数  
 #类中定义的函数叫做“方法”：count方法  
 patt\_dict **=** Counter**()** #这里修改,将{} 换成Counter()  
 cpatt **=** re.compile**(**patt**)** #较上一种方法,这里只定义了patt,不是调用self.patt参数了  
 **with** open**(**self.name**) as** fobj**:** #打开文件，self.name = fname  
 **for** line **in** fobj**:** #遍历这个文件,获得每一行的内容  
 m **=** cpatt.search**(**line**)** #经过上面编译后,在这search每一行,返回符合ip正则或者br正则  
 **if** m**:** # 如果匹配到内容，放到下面key变量，None表示False  
 key **=** m.group**()** patt\_dict.update**([**key**])** #这里修改,将 patt\_dict[key] = patt\_dict.get(key,0) + 1换成 patt\_dict.update([key])  
 **return** patt\_dict  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** fname **= 'access\_log'** ip **= '^(\d+\.){3}\d+'** # 1.11.123.45, 12345.11111.23423.2535234  
 br **= 'Chrome|Firefox|MSIE'** lb\_ip **=** wenjianfenxi**(**fname**)** # 根据wenjianfenxi类创建对象lb\_ip对象，将fname,ip封装到lb\_ip的fname 和 patt中  
 print**(**lb\_ip.count**(**ip**))** #较上一种方法,这里使用对象传参时,添加了新的变量ip.这个变量给了count方法的patt  
 print**(**lb\_ip.count**(**br**))** #较上一种方法,这里使用对象传参时,添加了新的变量br.这个变量给了count方法的patt  
 #较上一种方法,这里少创建一个对象  
  
  
 #输出的结果按value次数大小排序输出u

# socket模块

## c/s架构

client/server架构

## 套接字

通过ip地址能找到主机,通过端口能找到程序

* TCP用主机的IP地址加上主机上的端口号作为TCP连接的端点，这种端点就叫做套接字（socket）或插口。
* 套接字用（IP地址：端口号）表示。
* 它是网络通信过程中端点的抽象表示，包含进行网络通信必需的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。

## 面向连接与无连接

* 无论使用哪一种地址家族,套接字的类型只有面向连接和无连接的的套接字
* 面向连接的主要协议就是传输空值协议TCP,套接字类型为 type=SOCK\_STREAM
* 无连接的主要协议是用户数据报协议UDP,套接字类型为type=SOCK\_DGRAM
* python中使用socket模块中的socket函数实现套接字创建

## str和bytes字符类型

# 字符有str类型和bytes类型，它们之间的转换：

# >>> s1 = '您好'

# >>> type(s1)

# <class 'str'>

# >>> s1.encode()

# b'\xe6\x82\xa8\xe5\xa5\xbd'

# >>> b1 = s1.encode()

# >>> b1

# b'\xe6\x82\xa8\xe5\xa5\xbd'

# >>> b1.decode()

# '您好'

## 端口号

小于254的:well-known,众所周知的端口号.例如:80 , 22 等

小于1024的:保留个unix应用的

大于1024的:自定义

# socket函数与方法

## 创建TCP服务器/客户端(用来了解底层工作模式)





### @@@创建TCP服务器与客户端(一次对话)

##套接字--创建TCP服务器  
**import** socket  
  
host **= ''** #表示0.0.0.0  
port **=** 12345  
addr **= (**host,port**)**s **=** socket.socket**()** #默认创建TCP套接字  
#程序执行结束后,操作系统默认会保留套接字1分钟,在1分钟内无法再次使用相同的套接字  
s.setsockopt**(**socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1**)** #取消1分钟限制  
  
s.bind**(**addr**)** #绑定地址到套接字  
s.listen**(**1**)** #启动监听过程 netstat -tlpn |grep :12345  
  
  
cli\_sock , cli\_addr **=** s.accept**()** #等待客户端连接,收到连接返回(客户机套接字)  
  
print**('这个客户机连上来了: '**,cli\_addr**)**data **=** cli\_sock.recv**(**1024**)** #最多一次接收1024字节数据  
print**(**data**)**cli\_sock.send**(b'How are you?\r\n')** # 自动回复的数据必须是bytes类型  
cli\_sock.close**()**s.close**()**

### @@@创建TCP服务器与客户端(多次对话)

##套接字--创建TCP服务器--两边对话--服务端server  
**import** socket  
  
host **= ''** #表示0.0.0.0  
port **=** 12345  
addr **= (**host,port**)**s **=** socket.socket**()** #默认创建TCP套接字 s 为变量,可自定义  
 #程序执行结束后,操作系统默认会保留套接字1分钟,在1分钟内无法再次使用相同的套接字 --如下取消  
s.setsockopt**(**socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1**)** #取消1分钟限制  
  
s.bind**(**addr**)** #绑定地址到套接字  
s.listen**(**1**)** #启动监听过程 netstat -tlpn |grep :12345  
  
**while True:  
  
 try:** cli\_sock , cli\_addr **=** s.accept**()**

#接受客户端连接,收到连接返回(客户机套接字)并重新赋值给cli\_sock , cli\_addr  
 #s.accept() 接受客户端连接,  
 **except** KeyboardInterrupt**:** print**(' 退出')  
 break** print**('这个客户机连上来了: '**,cli\_addr**)** #有client连上之后,在server端输出内容  
  
 **while True:** data **=** cli\_sock.recv**(**1024**)** #等待接受客户端的数据.最多一次接收1024字节数据  
 **if** data.strip**() == b'quit':**

#判断对方发送来的是否是quit,如果是则跳出循环,将关闭套接字  
 **break** print**(**data.decode**()**,end**='')** #server端输出客户端发送来的数据  
  
 a **=** input**('>:') + '\r\n'** #获取server端回复的信息  
 # b = a.encode() #将输入的字符转换为:bytes类型  
 # cli\_sock.send(b) #向对方回复的数据必须是bytes类型,回复变量 b 的内容给对方  
 cli\_sock.send**(**a.encode**())** #server端发送数据  
  
 cli\_sock.close**()**s.close**()**####

客户端  
##套接字--创建TCP服务器--两边对话--客户端client  
**import** socket  
  
server **= '192.168.4.254'** #服务端地址  
port **=** 12345 #服务端TCP服务器端口  
addr **= (**server,port**)**c **=** socket.socket**()** #创建套接字  
c.connect**(**addr**)** #连接服务端  
  
**while True:** data **=** input**('quit to end >') + '\r\n'** #上面服务端程序做的是先收再发,这边客户端就先发再收  
 c.send**(**data.encode**())** #将输入的数据转换成bytes类型,发送给服务端  
 **if** data.split**() == 'quit':** #将输入的字符串去掉两边空白(split的作用),再判断  
 **break** rdata **=** c.recv**(**1024**)** #接受对方(服务端)的数据  
 print**(**rdata.decode**()**,end**='')**c.close**()**

## 创建UDP服务器(用来了解底层工作模式)





### @@@创建UDP服务端和客户端(一次对话)

#创建udp服务器server---一次对话  
**import** socket  
  
host **= ''**port **=** 12345  
addr **= (**host,port**)**s **=** socket.socket**(**type**=**socket.SOCK\_DGRAM**)**s.setsockopt**(**socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1**)**s.bind**(**addr**)**data , cli\_addr **=** s.recvfrom**(**1024**)** #一次最多接收1024字节数据，返回值是(数据，客户机地址)  
  
print**(**data.decode**()**,end**='')**s.sendto**(b'from server:hello world\r\n'**, cli\_addr**)** #项客户端地址发送数据  
s.close**()**##创建udp客户端client--一次对话  
**import** socket  
  
host **= '192.168.4.254'**port **=** 12345  
addr **= (**host,port**)**c **=** socket.socket**(**type**=**socket.SOCK\_DGRAM**)**c.sendto**(b'from client:nihao\r\n'**, addr**)**info **=** c.recvfrom**(**1024**)**print**(**info**())**

### @@@创建UDP服务端和客户端(多次对话)

创建udp服务器server**---**多次对话**--**返回信息加上时间戳  
**import** socket  
**from** time **import** strftime  
  
host **= ''**port **=** 12345  
addr **= (**host,port**)**s **=** socket.socket**(**type**=**socket.SOCK\_DGRAM**)**s.setsockopt**(**socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1**)**s.bind**(**addr**)  
  
while True:  
 try:** data , cli\_addr **=** s.recvfrom**(**1024**)** #一次最多接收1024字节数据，返回值是(数据，客户机地址)  
 **except** KeyboardInterrupt**:** print**(' 结束')  
 break** #data = data.decode() #bytes转str  
 print**(**data.decode,end**='')** rdata **= '[%s] %s' % (**strftime**('%H:%M:%S')**,data**)** s.sendto**(**rdata.encode**()** , cli\_addr**)** #str转bytes 向客户端发送数据  
  
s.close**()**##创建udp客户端client--多此对话--返回信息加上时间戳  
**import** socket  
  
host **= '192.168.4.254'**port **=** 12345  
addr **= (**host,port**)**c **=** socket.socket**(**type**=**socket.SOCK\_DGRAM**)  
  
while True:** data **=** input**('> ') + '\r\n'  
  
 if** info.decode **== 'quit':  
 break** c.sendto**(**data.encode**()**, addr**)** # info = c.recvfrom(1024) # (数据，服务器地址)  
 data **=** c.recvfrom**(**1024**)[**0**]** print**(**data.decode**()**, end**='')**

c.close**()**

# 多进程

## forking

## fork(分岔)

* linux系统运行程序,命令时,如果是fork执行,父进程先产生子进程,指令在子进程中执行
* 父进程将自身资源拷贝一份,命令在子进程中运行时,就具有和父进程完全一样的运行环境
* 子进程执行完毕后将会销毁,回到父进程

## 进程的生命周期

* 父进程fork出子进程并挂起
* 子进程运行完毕后,释放大部分资源,并通知父进程,这个时候的子进程被称作为僵尸进程
* 父进程获知子进程结束,子进程所有资源释放



## 脚本执行方式

执行脚本：

[root@room8pc16 day01]# cat chdir.sh

#!/bin/bash

cd /home

1. bash chdir.sh #使用解释器执行,不会切换目录,应为是fork执行
2. ./chdir.sh #里面定义了解释器,用里面的解释器去执行,不会切切换目录,因为是fork执行
3. source chdir.sh

## 僵尸进程

* 僵尸进程没有任何可执行代码,也不能被调度
* 如果系统中存在过多的僵尸进程,将会导致没有可用的进程号,导致系统不能产生新的进程
* 可以视图杀死其父进程或重启系统来消除僵尸进程

## forking编程

需要调用os.fork()函数实现forking功能

## fork编程思路：

1、考虑清楚，父子进程分别做什么工作

2、一般来说，父进程只产生子进程

3、子进程负责做具体的工作，完成后要退出

**import** os  
  
print**('start')**os.fork**()** #产生子进程,以下代码将会在父进程和子进程中同时执行  
print**('hello world')**#输出结果:  
start  
hello world  
hello world

**import** os  
print**('start')**retval **=** os.fork**()** # 父进程的返回值是非0值(子进程的PID)，子进程的返回值是0  
**if** retval**:** #判断retval是否非空  
 print**('父进程')  
else:** print**('子进程')**print**('both')**#输出结果:  
start  
父进程  
both  
子进程  
both

## fork变成必须添加子进程退出

否则循环会造成大批量子进程的创建

################################  
**import** os  
**for** i **in** range**(**3**):** retval **=** os.fork**()  
 if not** retval**:** print**('hello World')** exit**()** # 子进程执行完毕后退出，否则它将继续生成子进程  
  
# #输出结果:  
# Hello World!  
# Hello World!  
# Hello World!

## zombie僵尸进程的出现

观察僵尸进程：

1、子进程优先于父进程结束

2、父进程没有处理僵尸进程的代码

## os.waitpid处理子进程

waitpid() 处理子进程,

* 它接受第一个参数写为-1,表示与wait()函数的作用相同,
* 第二个参数,如果是0表示挂起父进程,直到子进程退出,如果是1表示不挂起.
* waitpid()的返回值:有两个值

如果子进程尚未退出是0(状态),

如果子进程已经结束,他的返回值是(子进程PID,状态)

import os

import time

print('start')

retval = os.fork()

if retval:

print('父进程')

result = os.waitpid(-1,0) #挂起父进程,子进程变成僵尸进程,处理后才继续向下执行print(result)

print(result) #输出(26364, 0)

else:

print('子进程')

time.sleep(10)

exit()

time.sleep(5)

print('Done')

# 多线程

## 什么是多线程

多线程（英语：multithreading），是指从软件或者硬件上实现多个线程并发执行的技术。

在一个程序中，这些独立运行的程序片段叫作“线程”（Thread），利用它编程的概念就叫作“多线程处理（Multithreading）”。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程（台湾译作“执行绪”），进而提升整体处理性能。

## thread和threading模块

python的thread和threading区别

python提供了多种模块用来支持多线程编程，

thread（在python3中改名为\_thread）,threading,和 queue模块。

通过加入queue模块，用户可以创建多个线程共享数据的队列数据结构。

thread和threading模块都可以用来创建和管理线程，

* thread模块提供了基本的线程和锁支持。
* threading提供的是更高级的完全的线程管理。

低级别的thread模块是推荐给高手用，一般应用程序推荐使用更高级的threading模块：

* 它更先进有完善的线程管理支持，此外，在thread模块的一些属性会和threading模块的这些属性冲突。
* thread模块有很少的（实际上是一个）同步原语，而threading却有很多。
* thread模块没有很好的控制，特别当你的进程退出时，

比如：当主线程执行完退出时，其他的线程都会无警告，无保存的死亡，而threading会允许默认，重要的子线程完成后再退出，它可以特别指定daemon类型的线程。

## 多线程编程思路

多线程也是用于提升效率的,与fork不同,fork会产生僵尸进程,线程不会

编程思路:

1,主线程生成工作线程

2,工作线程做具体的工作,工作做完,自动退出,不会产生僵尸进程

3,不会产生子进程未关闭的系统崩溃的情况

## GIL:全局解释器锁

python是一个解释器, 它负责把程序代码发送到cpu,由于GIL的限制,python解释器某一个时刻只允许运行一个线程

由于物理上得限制，各CPU厂商在核心频率上的比赛已经被多核所取代。为了更有效的利用多核处理器的性能，就出现了多线程的编程方式，而随之带来的就是线程间数据一致性和状态同步的困难。即使在CPU内部的Cache也不例外，为了有效解决多份缓存之间的数据同步时各厂商花费了不少心思，也不可避免的带来了一定的性能损失。

Python当然也逃不开，为了利用多核，Python开始支持多线程。而解决多线程之间数据完整性和状态同步的最简单方法自然就是加锁。 于是有了GIL这把超级大锁，而当越来越多的代码库开发者接受了这种设定后，他们开始大量依赖这种特性（即默认python内部对象是thread-safe的，无需在实现时考虑额外的内存锁和同步操作）。

## 时间片???

线程1开始处理,时间片用完,线程2开始处理,时间片用完,线程1又开始处理,时间片用完,线程2又开始处理直到处理完成.

时间片即CPU分配给各个程序的时间，每个线程被分配一个时间段，称作它的时间片，即该进程允许运行的时间，使各个程序从表面上看是同时进行的。如果在时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。而不会造成CPU资源浪费。在宏观上：我们可以同时打开多个应用程序，每个程序并行不悖，同时运行。但在微观上：由于只有一个CPU，一次只能处理程序要求的一部分，如何处理公平，一种方法就是引入时间片，每个程序轮流执行。

## IO密集型应用:

瓶颈来自于IO(输入/输出)

多进程多线程都适用

比喻:群发出请求,谁回应就处理谁,不回应的暂时不管,处理回应的

## cpu(计算)密集型应用:

测试加法时间的例子属于这类,瓶颈来自于CPU,

多进程适用

比喻:算术,谁算的快,谁就运行快

## @@@#扫描存活主机--多线程+oop方式

####扫描存活主机--多线程+oop方式  
**import** subprocess  
**import** os  
**import** threading  
  
**class Ping:  
 def \_\_init\_\_(**self,host**):** self.host **=** host  
  
 **def \_\_call\_\_(**self**):** rc **=** subprocess.run**('ping -c2 %s &>/dev/null' %** self.host , shell**=True)  
 if** rc.returncode **==** 0**:** print**('%s : up' %** self.host**)  
 else:** print**('%s : down' %** self.host**)  
if** \_\_name\_\_ **== '\_\_main\_\_':** iplist **= ['176.52.8.%s' %** i **for** i **in** range**(**1,255**)]  
 for** ip **in** iplist**:** t **=** threading.Thread**(**target**=**Ping**(**ip**))** #创建一句Ping的类创建target实例  
 t.start**()** #实际是执行target()

###多线程---join  
# import threading  
# import time  
# def say\_hi(n=3):  
# print('hello world')  
# time.sleep(n)  
# print('done')  
# if \_\_name\_\_ == '\_\_main\_\_':  
# print('start')  
# t1 = threading.Thread(target=say\_hi)  
# t1.start()  
# t1.join() #等待t1子线程结束  
# t2 = threading.Thread(target=say\_hi,args=(7,))  
# t2.start()  
###输出如下  
# start  
# hello world  
# done #睡眠三秒后出现  
# hello world #上面执行完之后出现  
# done #上一步出现后,再睡眠7秒后出现

## 单进程,多进程,多线程比较

# import time  
# import threading  
# import os  
# def myadd():  
# result = 0  
# for i in range(1,20000001):  
# result += i  
# print(result)  
# if \_\_name\_\_ == '\_\_main\_\_':  
# ####单进程:代码交给一个CPU运行  
# start1 = time.time()  
# for i in range(2):  
# myadd()  
# end1 = time.time()  
# print(end1 - start1)  
# print()  
# #####多进程:代码分配给多个cpu运算  
# start2 = time.time()  
# for i in range(2):  
# retval = os.fork()  
# if not retval:  
# myadd()  
# exit()  
# for i in range(2):  
# os.waitpid(-1,0)  
# end2 = time.time()  
# print(end2 - start2)  
# print()  
# #####多线程,  
# start3 = time.time()  
# t1 = threading.Thread(target=myadd)  
# t1.start()  
# t2 = threading.Thread(target=myadd)  
# t2.start()  
# t1.join() #等待t1子线程结束  
# t2.join() #等待t2子线程结束  
# end3 = time.time()  
# print(end3 - start3)  
# print()  
###多线程运行速度与单进程速度差不多的原因是:GIL全局解释器锁限制了  
###python是一个解释器, 它负责把程序代码发送到cpu,由于GIL的限制,python解释器某一个时刻只允许运行一个线程

# PyMysql 模块(安装)

pymsql是Python中操作MySQL的模块，

默认没有,需要到官方下载

python额外包的下载:https://pypi.org

## pymysql在线安装

[root@room9pc01 .pip]# pip3 --version #查看pip工具版本,一般随python安装就安装好

[root@room9pc01 nsd\_2018]# pip3 --version

pip 9.0.1 from /usr/local/lib/python3.6/site-packages (python 3.6)

**1,直接安装**

[root@room8pc16 day02]#pip3 install pymysql

**2,如果现在安装比较慢:可实用国内镜像站点**

[root@room8pc16 day02]# cat ~/.pip/pip.conf

[global]

index-url = http://mirrors.163.com/pypi/simple/

[install]

trusted-host=mirrors.163.com

**3,此方法也适用,python别的软件或程**序

## pymysql本地安装

# cd pymysql\_pkgs/ #进入准备好的安装包目录

# pip3 install \*

**使用之前请安装好MySQL**

安装mariadb-server 或mysql-server 并启动,配置root 密码

[root@room9pc01 ~]# tar -xf mysql-5.7.17.tar

[root@room9pc01 ~]# yum -y install mysql-community-\*.rpm

[root@room9pc01 ~]# systemctl start mysqld

[root@room9pc01 ~]# systemctl enable mysqld

[root@room9pc01 ~]# systemctl status mysqld

[root@room9pc01 ~]# grep 'temporary password' /var/log/mysqld.log

2019-04-22T08:25:27.053853Z 1 [Note] A temporary password is generated for root@localhost: jwac+?0V+4Aq

[root@room9pc01 ~]# mysql -uroot -p'jwac+?0V+4Aq #初次登录

mysql> alter user root@"localhost" identified by "123qqq...AAA"; #修改密码

mysql> create database nsd1811 default charset utf8;

# PyMsql的使用

概念--游标: 游标就是游动的标识

通俗的说:一条sql取出对应n条结果资源的接口/句柄,就是游标,沿着游标可以一次取出一行

## 创建表

**import** pymysql  
  
conn **=** pymysql.connect**(** host**='127.0.0.1'**,  
 port**=**3306,  
 user**='root'**,  
 passwd**='123qqq...AAA'**,  
 db**='nsd1811'**,  
 charset**='utf8')** #创建连接  
cursor **=** conn.cursor**()** #创建游标,查询数据默认为元组类型  
##sql语句  
# create\_dep = '''create table departments(  
# dep\_id INT ,dep\_name VARCHAR(20),PRIMARY KEY (dep\_id)  
# )'''  
# cursor.execute(create\_dep) #执行sql语句  
  
# create\_emps = '''create table employees(  
# emp\_id INT ,emp\_name VARCHAR(20), email VARCHAR(50),dep\_id INT,  
# PRIMARY KEY (emp\_id),FOREIGN KEY (dep\_id) REFERENCES departments(dep\_id) )'''  
# cursor.execute(create\_emps)  
  
create\_sal**='''create table salary(  
auto\_id INT ,emp\_id INT , date DATE ,basic INT , awards INT ,  
PRIMARY key(auto\_id), FOREIGN KEY (emp\_id) REFERENCES employees(emp\_id) )'''**cursor.execute**(**create\_sal**)**# create\_dep = '''create table employees(  
# emp\_id INT,  
# emp\_name VARCHAR(20),  
# email VARCHAR(50),  
# dep\_id INT ,  
# PRIMARY KEY (emp\_id),FOREIGN KEY(dep\_id) REFERENCES departments(dep\_id))'''  
# cursor.execute(create\_dep) #执行sql语句

conn.commit**()** #提交改动  
cursor.close**()** #关闭游标  
conn.close**()** #关闭连接

# SQLAlchemy

* 不仅仅只用于连接mysql数据库,也能连接其他数据库
* sqlAlchemy是python中最著名的ORM(Object Relationship Mapping)框架了。
* SQLAlchemy是Python编程语言下的一款开源软件。提供了SQL工具包及对象关系映射（ORM）工具，使用MIT许可证发行。
* SQLAlchemy“采用简单的Python语言，为高效和高性能的数据库访问设计，实现了完整的企业级持久模型”。SQLAlchemy的理念是，SQL数据库的量级和性能重要于对象集合；而对象集合的抽象又重要于表和行。

表与类对应,

表字段与类变量对应

## SQLAlchemy安装

1在线安装:

[root@room9pc01 ~]# pip3 install sqlalchemy

Collecting sqlalchemy

Downloading http://mirrors.163.com/pypi/packages/2b/b2/e6f5c5efc68942edefaa924e8fbea0b32375baa434a511cbf6bb17769cf6/SQLAlchemy-1.3.3.tar.gz (5.9MB)

100% |████████████████████████████████| 5.9MB 155kB/s

Installing collected packages: sqlalchemy

Running setup.py install for sqlalchemy ... done

Successfully installed sqlalchemy-1.3.3

You are using pip version 9.0.1, however version 19.0.3 is available.

You should consider upgrading via the 'pip install --upgrade pip' command.

2本地安装,

# cd sqlalchemy\_pkgs/

# pip3 install \*

## ORM技术

ORM技术：Object-Relational Mapping，把关系数据库的表结构映射到对象上。

Object：对象，对应python的class

Relationship：关系，对应关系型数据库

Map：映射

将数据库的表映射成一个class，表中的每个字段就是class中的类属性(变量)

创建class的实例，与表中的一行记录映射

一句话解释的话就是，一种可以把model中的模型和数据库中的一条数据相互转换的工具。

数据库表是一个二维表，包含多行多列。把一个表的内容用Python的数据结构表示出来的话，可以用一个list表示多行，list的每一个元素是tuple，表示一行记录，比如，包含id和name的user表：

[

('1', 'Michael'),

('2', 'Bob'),

('3', 'Adam')

]

Python的DB-API返回的数据结构就是像上面这样表示的。

但是用tuple表示一行很难看出表的结构。如果把一个tuple用class实例来表示，就可以更容易地看出表的结构来：

class User(object):

def \_\_init\_\_(self, id, name):

self.id = id

self.name = name

[

User('1', 'Michael'),

User('2', 'Bob'),

User('3', 'Adam')

]

这就是传说中的ORM技术：Object-Relational Mapping，把关系数据库的表结构映射到对象上

## 第一步，导入SQLAlchemy，并初始化DBSession：

使用前,先创建数据库,指定字符集为utf8,否则插入中文数据会报错,

mysql> create database tedu1811 DEFAULT CHARSET=utf8;

Query OK, 1 row affected (0.00 sec)

创建引擎,mysql连接engine **=** create\_engine**(...)**

声明映射,创建基类:Base **=** declarative\_base**()**

创建映射类:**class Departments(**Base**):**

ORM访问数据库的句柄称作为Session

**from** sqlalchemy **import** create\_engine  
**from** sqlalchemy.ext.declarative **import** declarative\_base  
**from** sqlalchemy **import** Column, Integer ,String,ForeignKey,Date  
**from** sqlalchemy.orm **import** sessionmaker

engine **=** create\_engine**(  
 'mysql+pymysql://root:123qqq...AAA@127.0.0.1/tedu1811?charset=utf8'**,  
 encoding **= 'utf8'**,  
 # echo = True  
**)**# 创建引擎，根据数据库类型，选择适当的连接方式  
# mysql+pymysql://用户名:密码@服务器/数据库?参数, --初始化连接

# '数据库类型+数据库驱动名称://用户名:口令@机器地址:端口号/数据库名'  
# echo=True，调试模式，屏幕上打印操作详情，生产环境需要关闭  
  
Base **=** declarative\_base**()** #创建ORM映射类的基类,声明映射  
Session **=** sessionmaker**(**bind**=**engine**)** #创建会话类,绑定引擎 新建的crud.py用的上  
  
**class Departments(**Base**):** #定义表对象  
 \_\_tablename\_\_**='departments'** #指定该类与哪个表对应,departments为要创建的表名  
 dep\_id **=** Column**(**Integer,primary\_key**=True)** #表结构,设置dep\_id字段,并设置为主键  
 dep\_name **=** Column**(**String**(**20**)**,unique**=True)** #表结构,设置dep\_id字段,属性为唯一,不能重复  
  
**class Employees(**Base**):** \_\_tablename\_\_ **= 'employees'** emp\_id **=** Column**(**Integer,primary\_key**=True)** emp\_name **=** Column**(**String**(**20**))** email **=** Column**(**String**(**50**))** dep\_id **=** Column**(**Integer, ForeignKey**('departments.dep\_id'))** #dep\_id设置为外键约束,参考于departments表中的dep\_id  
**class Salary(**Base**):** \_\_tablename\_\_ **= 'sqlary'** auto\_id **=** Column**(**Integer,primary\_key**=True)** date **=** Column**(**Date**)** emp\_id **=** Column**(**Integer,ForeignKey**('employees.emp\_id'))** baisc **=** Column**(**Integer**)** awards **=** Column**(**Integer**)  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** Base.metadata.create\_all**(**engine**)** #如果上面的表已存在,不会再创建  
# ###成功屏幕将会显示创建过程:  
# ......  
# 2019-04-23 11:08:20,824 INFO sqlalchemy.engine.base.Engine ###echo=True，调试模式 的原因  
# CREATE TABLE departments (  
# dep\_id INTEGER NOT NULL AUTO\_INCREMENT,  
# dep\_name VARCHAR(20),  
# PRIMARY KEY (dep\_id),  
# UNIQUE (dep\_name)  
# 2019-04-23 11:08:21,087 INFO sqlalchemy.engine.base.Engine COMMIT

## 第二步,增删改查,

**固定格式为:(以插入一条数据为例)**

**from** SqlAlchemy **import** Departments, Employees, Salary, Session  
  
session **=** Session**()** #创建session

#插入数据语句  
hr **=** Departments**(**dep\_id**=**1, dep\_name**='人事部')**session.add**(**hr**)** #将hr添加到session

#查询语句

query1 **=** session.query**(**Departments**)** # 将class作为参数，返回实例  
# print(query1) # query1只是个sql语句  
deps **=** query1.all**()** # 取出查询的全部结果  
# print(deps) # deps是由 数据库表每行记录生成的实例 构成的列表  
**for** dep **in** query1**:** print**(**dep.dep\_id, dep.dep\_name**)**

#删除语句

query16 **=** session.query**(**Departments**)**.filter**(**Departments.dep\_id**==**6**)**ui **=** query16.one**()**session.delete**(**ui**)**

#修改语句

query15 **=** session.query**(**Departments**)**.filter**(**Departments.dep\_name**=='人事部')**  
# hr = query15.one()  
# print(hr) #将实例赋值给'hr'  
# print(hr.dep\_name) #输出'人事部'  
# hr.dep\_name ='人力资源部' #将人事部修改成了'人力资源部'

session.commit  
session.close

## 增删改查实际案例

**from** SqlAlchemy **import** Departments, Employees, Salary, Session  
  
session **=** Session**()** #创建session  
#######插入一条数据  
# hr = Departments(dep\_id=1, dep\_name='人事部')  
# session.add(hr) #将hr添加到session  
#  
# session.commit  
# session.close  
###解释:  
# hr为变量, Departments,为创建表时候的表对象,可查看SqlAlchemy.py文件,再插入对应的值  
  
########插入多条数据  
# ops = Departments(dep\_id=2, dep\_name='运维部')  
# dev = Departments(dep\_id=3, dep\_name='开发部')  
# qa = Departments(dep\_id=4, dep\_name='测试部')  
# finance = Departments(dep\_id=5, dep\_name='财务部')  
# ui = Departments(dep\_id=6, dep\_name='设计部')  
# session.add\_all([ops, dev, qa, finance, ui])  
#######插入多条数据  
# yc = Employees(  
# emp\_id = 1,  
# emp\_name = '杨晨',  
# email = 'yc@163.com',  
# dep\_id = 2  
# )  
# zyp = Employees(  
# emp\_id = 2,  
# emp\_name = '郑云鹏',  
# email = 'zyp@163.com',  
# dep\_id = 2  
# )  
# lzj = Employees(  
# emp\_id = 3,  
# emp\_name = '李注江',  
# email = 'lzj@163.com',  
# dep\_id = 2  
# )  
# cyj = Employees(  
# emp\_id = 4,  
# emp\_name = '陈益建',  
# email = 'cyj@qq.com',  
# dep\_id = 1  
# )  
# hqp = Employees(  
# emp\_id = 5,  
# emp\_name = '黄勤品',  
# email = 'hqp@tarena.com',  
# dep\_id = 3  
# )  
# zds = Employees(  
# emp\_id = 6,  
# emp\_name = '赵东升',  
# email = 'zds@.tedu.cn',  
# dep\_id = 3  
# )  
# cd = Employees(  
# emp\_id = 7,  
# emp\_name = '陈栋',  
# email = 'cd@tedu.cn',  
# dep\_id = 4  
# )  
# session.add\_all([yc, zyp, lzj, cyj, hqp, zds, cd])  
  
#############################  
# query1 = session.query(Departments) # 将class作为参数，返回实例  
# # print(query1) # query1只是个sql语句  
# deps = query1.all() # 取出查询的全部结果  
# # print(deps) # deps是由 数据库表每行记录生成的实例 构成的列表  
# for dep in query1:  
# print(dep.dep\_id, dep.dep\_name)  
#############################  
# query2 = session.query(Employees)  
# print(query2) #未切片.返回的是sql语句  
# print(query2.all()) #返回的是实例构成的列表  
# for emp in query2:  
# print(emp.emp\_id, emp.emp\_name, emp.email)  
#############################  
# ##将类变量作为参数，返回值是元组\*\*\*  
# query3 = session.query(Employees.emp\_name, Employees.email)  
# print(query3)  
# print(query3.all()) #返回的是元组组成的列表  
# for name, email in query3:  
# print(name, email)  
############-------排序查询  
# query4 = session.query(Employees).order\_by(Employees.emp\_id)  
# print(query4) #取的是sql语句  
# for emp in query4:  
# print(emp.emp\_id, emp.emp\_name, emp.email)  
############------切片查询  
###取切片时,涉及到了值,所以下面query5不再是sql语句,而是实例构成的列表  
# query5 = session.query(Departments).order\_by(Departments.dep\_id)[2:5]  
# print(query5) #取的不是sql语句,而是实例构成的列表  
# for dep in query5:  
# print(dep.dep\_id,dep.dep\_name)  
###########-----过滤查询  
# query6 = session.query(Departments).filter(Departments.dep\_id<=3)  
# print(query6) #返回查询语句  
# for dep in query6:  
# print(dep.dep\_id, dep.dep\_name)  
  
##########------模糊查询  
# query7 = session.query(Employees).filter(Employees.email.like('%.cn'))  
# print(query7)  
# for emp in query7:  
# print(emp.emp\_name,emp.email)  
  
#########-----in  
# query8 = session.query(Employees).filter(Employees.dep\_id.in\_([1,3]))  
# print(query8)  
# for emp in query8:  
# print(emp.emp\_name,emp.dep\_id)  
  
#########----in--取反加 ~  
# query9 = session.query(Employees).filter(~Employees.dep\_id.in\_([1,3]))  
# print(query9)  
# for emp in query9:  
# print(emp.emp\_name,emp.dep\_id)  
########----and\_ 和 or\_--需要单独导入  
# from sqlalchemy import and\_,or\_  
# query10 = session.query(Departments) \  
# .filter(and\_(Departments.dep\_id>=2,Departments.dep\_id<6))  
# print(query10)  
# for dep in query10:  
# print(dep.dep\_id , dep.dep\_name)  
#  
# query11 = session.query(Departments) \  
# .filter(or\_(Departments.dep\_id<=2))  
  
######---统计查询  
# query12 = session.query(Departments).count()  
# print(query12)  
  
#####---多表查询  
# query13 = session.query(Employees.emp\_name,Departments.dep\_name)\  
# .join(Departments)  
# for emp\_name,dep\_name in query13:  
# print(emp\_name,dep\_name)  
##--注意join里面的内容,谁在后就join谁\*\*  
# query14 = session.query(Departments.dep\_name, Employees.emp\_name)\  
# .join(Employees)  
# for emp\_name ,dep\_name in query14:  
# print(emp\_name,dep\_name)  
  
#####---字段更新  
# query15 = session.query(Departments).filter(Departments.dep\_name=='人事部')  
# print(query15.all()) #返回的是长度为1的列表  
  
# print(query15.all()[0]) #同下一样  
# print(query15.one()) #返回的是实例,同上一样  
# hr = query15.one()  
# print(hr) #将实例赋值给'hr'  
# print(hr.dep\_name) #输出'人事部'  
# hr.dep\_name ='人力资源部' #将人事部修改成了'人力资源部'  
#####---删除一条记录  
# query16 = session.query(Departments).filter(Departments.dep\_id==6)  
# ui = query16.one()  
# session.delete(ui)  
  
session.commit**()**session.close**()**

# 万维网与HTTP

## HTTP概述:

* 超文本传输协议http,是互联网上最为广泛的一种网络协议,所有www文件都必须遵守这个标准,设计HTTP最初的目的是为了提供一种发布和接受HTML页面的方法.
* HTTP是一个客户端和服务端请求和应答的标准,通过使用web浏览器,网络爬虫或者其他的工具,客户端发起一个到服务器上指定端口(默认为80端口)的HTTP请求

## HTTP工作过程:

1. 客户端发起一个请求,建立一个到服务器指定端口的连接
2. 服务端则在那个端口监听客户端发送过来的请求
3. 一旦接受到请求,服务器向客户端发回一个状态行,比如:”HTTP/1.1 200 OK”,和响应的”消息”
4. 消息的消息体可能是请求的文件,错误消息,或者其他一些信息
5. HTTP使用TCP协议

## 请求和响应:

1. 客户端向服务器发送获取文档的请求(request)
2. 一旦发送完请求,客户端就会进行等待,知道从服务器接收到完整的响应(response)为止
3. 当前最流行的HTTP1.4版本的协议中,不允许客户端在尚未收到上一个请求前就在同一个套接字上开始发送第二个请求

## HTTP请求信息3部分组成：

1、请求方法（GET/POST）、URI、协议/版本

2、请求头(Request Header)

3、请求正文

POST http://xg.mediportal.com.cn/health/sms/verify/telephone HTTP/1.1

User-Agent: DGroupPatient/1.052701.230/Dalvik/2.1.0 ([Linux](http://lib.csdn.net/base/linux" \t "/root/文档\x/_blank" \o "Linux知识库); U; [Android](http://lib.csdn.net/base/android" \t "/root/文档\x/_blank" \o "Android知识库) 5.1.1; KIW-AL10 Build/HONORKIW-AL10)  
Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
Host: xg.mediportal.com.cn  
Connection: Keep-Alive  
Accept-Encoding: gzip  
Content-Length: 33   
  
telephone=15527177736&userType=1&

**------------------------------请求信息详解-----------------------------------------------------------------**

**请求方法:第一行绿色部分--------------**

方法:POST

URL:http://xg.mediportal.com.cn/health/sms/verify/telephone

协议:HTTP

版本:1.1

**请求头:中间红色部分---------------**

User-Agent: #用户发送请求的客户端环境

Content-Type: #表单默认的提交数据格式,浏览器会根据Content-Type来决定如何显示返回的消息体内容

Host: #请求资源的Intenet主机和端口号

Connection: Keep-Alive #持久连接

Accept-Encoding: gzip #浏览器能够进行解码的数据编码方式

Content-Length: 33 #请求正文的长度

备注:Content-Type是返回消息中非常重要的内容，表示后面的文档属于什么MIME类型。Content-Type: [type]/[subtype]; parameter。例如最常见的就是text/html，它的意思是说返回的内容是文本类型，这个文本又是HTML格式的。原则上浏览器会根据Content-Type来决定如何显示返回的消息体内容

**请求正文:最后一行蓝色部分------------**

telephone= ....

备注:请求头和请求正文之间是一个空行，这个行非常重要，它表示请求头已经结束，接下来的是请求正文。请求正文中可以包含客户提交的查询字符串信息：

## HTTP响应格式3部分

HTTP应答与HTTP请求相似，HTTP响应也由3个部分构成，分别是：

1、状态行

2、响应头(Response Header)

3、响应正文

POST http://xg.mediportal.com.cn/health/sms/verify/telephone HTTP/1.1

User-Agent: DGroupPatient/1.052701.230/Dalvik/2.1.0 (Linux; U; Android 5.1.1; KIW-AL10 Build/HONORKIW-AL10)

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

Host: xg.mediportal.com.cn

Connection: Keep-Alive

Accept-Encoding: gzip

Content-Length: 33

telephone=15527177736&userType=1&

## HTTP重要请求方法

http请求方法:

常用方法:GET,POST,HEAD

其他方法:OPTIONS,PUT,DELETE.TRACE和CONNECT

HTTP/1.1协议中共定义了八种方法（有时也叫“动作”）来表明Request-URL指定的资源不同的操作方式

## GET方法:请求获取由Request-URI所标识的资源

向特定的资源发出请求。它本质就是发送一个请求来取得服务器上的某一资源。资源通过一组HTTP头和呈现数据（如HTML文本，或者图片或者视频等）返回给客户端。GET请求中，永远不会包含呈现数据。

## POST方法:在Request-URI所标识的资源后附加新的数据

向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 Loadrunner中对应POST请求函数：web\_submit\_data,web\_submit\_form

**get 和 post区别**

两者的区别:

　　　　get请求无消息体，只能携带少量数据,一般是浏览器输入网址访问网站,点击页面中的超链接,搜索表单

　　　　post请求有消息体，可以携带大量数据,一般来说是，表单提交数据，如登陆、注册

　　携带数据的方式：

　　　　get请求将数据放在url地址中

　　　　post请求将数据放在消息体中

GET请求请提交的数据放置在HTTP请求协议头中，而POST提交的数据则放在实体数据中；

GET方式提交的数据最多只能有1024字节，而POST则没有此限制。

其他请求方法:

1、OPTIONS

返回服务器针对特定资源所支持的HTTP请求方法也可利用向web服务器发送‘\*’的请求来测试服务器的功能性

2、HEAD

向服务器索与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以再不必传输整个响应内容的情况下，就可以获取包含在响应小消息头中的元信息。

3、PUT

向指定资源位置上传其最新内容

4、DELETE

请求服务器删除Request-URL所标识的资源

5、TRACE

回显服务器收到的请求，主要用于测试或诊断

6、CONNECT

HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

注意：

1）方法名称是区分大小写的，当某个请求所针对的资源不支持对应的请求方法的时候，服务器应当返回状态码405（Mothod Not Allowed）；当服务器不认识或者不支持对应的请求方法时，应返回状态码501（Not Implemented）。

2）HTTP服务器至少应该实现GET和HEAD/POST方法，其他方法都是可选的，此外除上述方法，特定的HTTP服务器支持扩展自定义的方法。

# urllib模块:

## urllib简介

urllib提供了一系列用于操作URL的功能。

urllib包括四个模块:

* urllib.request 可以用来发送request(请求)和获取request的结果
* urllib.error 包含了urllib.request产生的异常
* urllib.parse 用来解析和处理URL
* urllib.robotparse用来解析页面的robots.txt文件

## urllib.request模块

* urllib,request.urlopen(网页链接) 打开并爬取一个
* 读取内容常见的3中方式

1. read()读取文件的全部内容,与readlinse()不同的是,read()会把读取到的内容赋值给一个字符串变量
2. readlines读取文件的全部内容,readlinse()会把读取到的内容赋值给一个列表变量(不常用)
3. readline()读取文件的一行内容.(不常用)

## request.urlopen获取网络源代码

**from** urllib **import** request  
  
respone **=** request.urlopen**('http://www.baidu.com')** # 向指定的url发送请求，并返回服务器响应的类文件对象  
print**(**respone.read**())** # 类文件对象支持 文件对象的操作方法，如read()方法读取文件全部内容，返回字符串

实际上，如果我们在浏览器上打开百度主页， 右键选择“查看源代码”，你会发现，跟我们刚才打印出来的是一模一样。也就是说，上面的3行代码就已经帮我们把百度的首页的全部代码爬了下来。

## @@@爬取网页

**import** sys  
**from** urllib **import** request  
  
**def download(**url,fname**):** r **=** request.urlopen**(**url**)  
 with** open**(**fname,**'wb') as** fobj**:  
 while True:** data**=**r.read**(**1024**)  
 if not** data**:  
 break** fobj.write**(**data**)  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** download**(**sys.argv**[**1**]** , sys.argv**[**2**])**

## urllib.reques**t**.Request的使用

如果需要执行更复杂的操作，比如增加HTTP报头，必须创建一个 Request 实例来作为urlopen()的参数；而需要访问的url地址则作为 Request 实例的参数。

####---修改请求头  
**import** urllib.request  
  
request **=** urllib.request.Request**("http://www.baidu.com")** # url 作为Request()方法的参数，构造并返回一个Request对象  
response **=** urllib.request.urlopen**(**request**)** # Request对象作为urlopen()方法的参数，发送给服务器并接收响应  
  
html **=** response.read**()**print**(**html**)**###查看请求信息将会变成:  
# GET http://www.baidu.com/ HTTP/1.1  
# Accept-Encoding: identity  
# Host: www.baidu.com #请求资源的主机变www.baidu.com  
# User-Agent: Python-urllib/3.7  
# Connection: close

## @@@修改请求头

,模拟正常的浏览器访问

>>> from urllib import request

>>> headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86\_64; rv:52.0) Gecko/20100101 Firefox/52.0'}

>>> obj = request.Request('http://127.0.0.1/', headers=headers)

>>> r = request.urlopen(obj)

# tail -f /var/log/http/access\_log

## eog linux查看图片命令

~]# eog 图片目录 #可查看图片

##---爬取一个网站的所有图片,按原名命名--老师写day04-download.py---待补充注释

## 数据编码：encode() 和 decode

data='中国'  
print(data.encode()) #转换成utf8的二进制形式 #输出：b'\xe4\xb8\xad\xe5\x9b\xbd'  
print(type(data)) #输出：<class 'str'> 数据类型为字符型  
# help(data.encode) #可看到默认的编码方式是utf8  
data\_1 = data.encode('gbk') #转换成gbk的二进制形式,  
print(data\_1) #输出：b'\xd6\xd0\xb9\xfa'  
print(type( data\_1)) #输出<class 'bytes'> 数据类型是bytes(二进制)  
# print(data\_1.decode()) #bdata是gbk编码，默认decode用的是utf8，所以报错  
print(data\_1.decode('gbk')) #指定采用gbk的方式解码 输出：中国

## urllib.error模块

urllib.error 包含了urllib.request产生的异常

报错后会导致程序终端,需要捕获报错信息,且可以输出

## @@@处理下载错误程序

from urllib import request, error  
url1 = 'http://127.0.0.1/abcd'  
url2 = 'http://127.0.0.1/ban'  
try:  
 r1 = request.urlopen(url1)  
except error.HTTPError as e: # 把异常捕获后起名为e，e只是个变量  
 print('错误:', e) #输出了捕获的报错信息，程序不会终止  
try:  
 r2 = request.urlopen(url2)  
except error.HTTPError as e:  
 print('错误:', e)

## urllib.request.quote模块(URL编码)

URL标准中只允许一部分字符，比如数字，字母，部分符号等

汉字是不符合URL标准的，需要编码，浏览器看到汉字，实际上是经过编码的，

编程时，需要把这些字符先编码成允许的字符

from urllib import request  
url\_1 = 'http://www.sogou.com/web?query=复联4'  
# request.urlopen(url\_1) #报错，汉字不是URL允许的字符  
print(request.quote('复联4')) #编码：输出：%E5%A4%8D%E8%81%944  
url\_2 = 'http://www.sogou.com/web?query=' + request.quote('复联4') #真正的网址  
print(url\_2) #输出：http://www.sogou.com/web?query=%E5%A4%8D%E8%81%944

# python bs4(爬虫)

自己课后了解

# paramiko(远程管理)

paramiko是一个用于做远程控制的模块，使用该模块可以对远程服务器进行命令或文件操作,paramiko是用python语言写的一个模块，遵循SSH2协议，支持以加密和认证的方式，进行远程服务器的连接。

由于使用的是python这样的能够跨平台运行的语言，所以所有python支持的平台，如Linux, Solaris, BSD, MacOS X, Windows等，paramiko都可以支持，因此，如果需要使用SSH从一个平台连接到另外一个平台，进行一系列的操作时，paramiko是最佳工具之一。

paramiko

1、安装

(1)在线

# pip3 install paramiko

(2)离线

# cd paramiko\_pkgs/

# pip3 install \*

2、准备用于远程管理的虚拟机

node4.tedu.cn 192.168.1.4

node5.tedu.cn 192.168.1.5

node6.tedu.cn 192.168.1.6

3、应用

>>> import paramiko

>>> ssh = paramiko.SSHClient() # 创建用于连接ssh服务器的客户端实例

>>> ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy()) # 相当于回答yes,设置自动添加主机秘钥

>>> ssh.connect('192.168.4.4', username='root', password='123456') #连接ssh服务端

>>> ssh.exec\_command('mkdir /tmp/demo') #在ssh服务端执行的命令

>>> ssh.close()

说明：exec\_command()执行命令的返回值是个三元组，分别代表了输入、输出和错误

每一个部分都是类文件对象，它们都有read()方法。

@@@远程

## @@@ssh远程主机,执行命令

**import** paramiko  
  
**def rcmd(**host, user**='root'**, passwd**=None**, port**=**22, cmd**=None):** ssh **=** paramiko.SSHClient**()** #创建连接  
 ssh.set\_missing\_host\_key\_policy**(**paramiko.AutoAddPolicy**()** #相当于在连接是输入yes  
 ssh.connect**(**host, username**=**user, password**=**passwd, port**=**port**)** #连接  
 stdin, stdout, stderr **=** ssh.exec\_command**(**cmd**)** #命令结果为一个元组,有三个元素,分别赋值  
 print**(**stdout**)** #  
 out **=** stdout.read**()** #得到的是bills类型数据,需要out.decode()转成字符  
 error **=** stderr.read**()  
 if** out**:** print**('[%s] \033[32;1mOUT\033[0m:\n%s' % (**host, out.decode**()))  
 if** error**:** print**('[%s] \033[31;1mERROR\033[0m:\n%s' % (**host, error.decode**()))** ssh.close**()  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** rcmd**('192.168.1.4'**, passwd**='a'**, cmd**='id root; id wangwu')**

## @@@ssh远程主机,并发多线程，执行命令(较完善)

执行程序，指定一个主机文件（文件每行写一个IP地址），运行程序，将会ssh远程到每一台主机，并且执行一个命令

**import** paramiko  
**import** sys  
**import** getpass  
**import** os  
**import** threading  
  
**def rcmd(**host, user**='root'**, passwd**=None**, port**=**22, cmd**=None):** ssh **=** paramiko.SSHClient**()** #创建连接  
 ssh.set\_missing\_host\_key\_policy**(**paramiko.AutoAddPolicy**())** #相当于在连接是输入yes  
 ssh.connect**(**host, username**=**user, password**=**passwd, port**=**port**)** #连接  
 stdin, stdout, stderr **=** ssh.exec\_command**(**cmd**)**

#命令结果为一个元组,有三个元素,分别赋值

#exec\_command()执行命令的返回值是个三元组，分别代表了输入、输出和错误  
 out **=** stdout.read**()** #得到的是bills类型数据,需要out.decode()转成字符  
 error **=** stderr.read**()  
 if** out**:** #如果输出不为空,有输出  
 print**('[%s] \033[32;1mOUT\033[0m:\n%s' % (**host, out.decode**()))  
 if** error**:** #如果错误不为空,有错误  
 print**('[%s] \033[31;1mERROR\033[0m:\n%s' % (**host, error.decode**()))** ssh.close**()** #关闭连接  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':  
 if** len**(**sys.argv**) !=** 3**:** #判断是不是三个参数都有  
 print**("Usage: %s ipfile 'command'" %** sys.argv**[**0**])** exit**(**1**) #退出码为1** ipfile **=** sys.argv**[**1**]** #第2个位置参数,第1个默认为本身python文件本身  
 **if not** os.path.isfile**(**ipfile**):** #如果主机地址文件没有  
 print**('No such file:'**, ipfile**)** exit**(**2**) #退出码为2** passwd **=** getpass.getpass**()** #输入密码,密文方式  
 command **=** sys.argv**[**2**]** #第3个位置参数  
 **with** open**(**ipfile**) as** fobj**:  
 for** line **in** fobj**:** ip **=** line.strip**()** # rcmd(ip, passwd=passwd, cmd=command) 改用多线程  
 t **=** threading.Thread**(**target**=**rcmd, args**=(**ip,**)**, kwargs**={'passwd':** passwd, **'cmd':** command**})** t.start**()** # 拆分多线程命令:target(\*args, \*\*kwargs)  
 # 拆分多线程传参:target(ip, passwd=passwd, cmd=command)  
## ]# python3 paramikotest.py paramikoip.txt 'sleep 3 ;id root'  
# 执行命令时,多个参数用单引号,用;号隔开

# 邮件

## 邮件编程：

**1、准备邮件**

>>> from email.mime.text import MIMEText

# 书写正文，邮件是纯文本

>>> message = MIMEText('My email test.\r\n', 'plain', 'utf8')

# 编写头部信息

>>> from email.header import Header

>>> message['From'] = Header('root', 'utf8') #发送方

>>> message['To'] = Header('zhangsan', 'utf8') #接收方

>>> message['Subject'] = Header('py mail test', 'utf8') #主题

**2、发送邮件**

>>> import smtplib

>>> smtp = smtplib.SMTP()

>>> smtp.connect('127.0.0.1') #邮件服务器地址

>>> smtp.sendmail('root', ['root', 'zhangsan'], message.as\_string())

>>> smtp.close()

# mail

# su - zhangsan

$ mail

## @@@邮件服务器收发邮件

from email.mime.text import MIMEText  
from email.header import Header  
import smtplib  
import getpass  
  
def send\_email(msg, subject, sender, receivers, host, passwd):  
 message = MIMEText(msg, 'plain', 'utf8') #邮件正文内容  
 message['From'] = Header(sender, 'utf8') #发件人  
 message['To'] = Header(receivers[0], 'utf8') #收件人  
 message['Subject'] = Header(subject, 'utf8') #邮件主题  
 smtp = smtplib.SMTP()  
 smtp.connect(host) #邮件服务器地址  
 smtp.login(sender, passwd) #发件人邮箱密码  
 smtp.sendmail(sender, receivers, message.as\_bytes())  
 #sendmail方法发送邮件,必须有收件人,发件人,消息主体,message是一个字符串,表示邮件  
if \_\_name\_\_ == '\_\_main\_\_':  
 msg = 'python smtplib 邮件测试\r\n' #邮件正文内容  
 subject = 'py mail test' #邮件主题  
 sender = 'zhangzhigang79@126.com' #发件人  
 receivers = ['zhangzhigang79@126.com'] #收件人  
 host = 'smtp.126.com' #邮件服务器地址,这个地方要查询  
 passwd = getpass.getpass() #发件人邮箱密码  
 send\_email(msg, subject, sender, receivers, host, passwd)

## @@@用网易邮箱给自己发送邮件

**from** email.mime.text **import** MIMEText  
**from** email.header **import** Header  
**import** smtplib  
**import** getpass  
  
**def send\_email(**msg, subject, sender, receivers, host, passwd**):** message **=** MIMEText**(**msg, **'plain'**, **'utf8')** #邮件正文内容  
 message**['From'] =** Header**(**sender, **'utf8')** #发件人  
 message**['To'] =** Header**(**receivers**[**0**]**, **'utf8')** #收件人  
 message**['Subject'] =** Header**(**subject, **'utf8')** #邮件主题  
 smtp **=** smtplib.SMTP**()** smtp.connect**(**host**)** #邮件服务器地址  
 smtp.login**(**sender, passwd**)** #发件人邮箱密码  
 smtp.sendmail**(**sender, receivers, message.as\_bytes**())** #sendmail方法发送邮件,必须有收件人,发件人,消息主体,message是一个字符串,表示邮件  
**if** \_\_name\_\_ **== '\_\_main\_\_':** msg **= 'python smtplib 邮件测试\r\n'** #邮件正文内容  
 subject **= 'py mail test'** #邮件主题  
 sender **= 'zhangzhigang79@126.com'** #发件人  
 receivers **= ['zhangzhigang79@126.com']** #收件人  
 host **= 'smtp.126.com'** #邮件服务器地址,这个地方要查询  
 passwd **=** getpass.getpass**()** #发件人邮箱密码  
 send\_email**(**msg, subject, sender, receivers, host, passwd**)**

# json

* JSON(JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。JSON的数据格式其实就是python里面的字典格式，里面可以包含方括号括起来的数组，也就是python里面的列表。
* 可在网络上传输各种各样的数据类型
* Json最广泛的应用是作为AJAX中web服务器和客户端的通讯的数据格式。现在也常用于http请求中，所以对json的各种学习，是自然而然的事情
* 在python中，有专门处理json格式的模块—— json 和 picle模块（上面有此模块专题）
* Json 模块提供了四个方法： dumps、dump、loads、load
* pickle 模块也提供了四个功能：dumps、dump、loads、load

json 主要有两种结构:

1. “键 / 值 ”对的集合:python中主要对应成字典
2. 值的有序列表:在大部分语言中,它被理解为数组

## dumps方法:json.dumps

使用简单的json.dumps方法对简单数据类型进行编码,既可在网络中传输

**import** json  
adict **= {'name': 'tom'**, **'age':** 20**}**jdata1 **=** json.dumps**(**adict**)** # 将字典转换成json字符串  
print**(**jdata1**)** #输出:{"name": "tom", "age": 20}

**下面列举一下dumps（）的可填参数：**

indent：如果填0或者不填，则按照一行进行打印，否则按照indent的数值显示前面的空格（正整数形式）；

separators：分隔符，默认为“(',',':')”，它表示key之间用“,”隔开，key和value之间用“:”隔开；

encoding：编码格式，默认值是UTF-8；

sort\_keys：对key、value进行排序，默认值是False，即不排序；

ensure\_ascii：默认为True，如果dict对象里含有none-ASCII的字符，则显示\uXX的格式，如果为False，则能正常显示出来；

import json  
data1 = {'str3':'xyz','str2':'efgh','str1':'abcd'}  
d1 = json.dumps(data1,sort\_keys=True)  
print(d1) #输出：{"str1": "abcd", "str2": "efgh", "str3": "xyz"}

## loads方法:json.loads

对编码后的json对象进行decode解码. 转换为Python字典，得到原始数据,需要使用json.loads()函数

import json  
adict = {'name': 'tom', 'age': 20} #原始数据  
jdata1 = json.dumps(adict) # 将字典转换成json字符串  
print(jdata1) #输出:{"name": "tom", "age": 20}  
jdata2 = json.loads(jdata1) # 将json字符串转成字典  
print(jdata2) #输出:{'name': 'tom', 'age': 20}

## json.dump（）和 json.load（）

相对于上面所讲的dumps和loads来说，dump和load函数的功能类似，只不过前者是用来处理字符串类型的，而后者是用于处理文件类型的，如下所示：

## 天气预报

import urllib.request  
import json  
r = urllib.request.urlopen('http://www.weather.com.cn/data/sk/101010100.html')  
data\_1 = r.read()  
data\_2 = json.loads(data\_1) # 将json字符串转成字典  
print(data\_2) #输出：{'weatherinfo': {'city': '北京', 'cityid': '101010100', 'temp': '27.9'。。。。。。  
print(data\_2['weatherinfo']['city']) #输出：北京  
print(data\_2['weatherinfo']['temp']) #输出：27.9

# wget模块

安装：pip3 install wget

>>> import wget

>>> wget.download('https://upload-images.jianshu.io/upload\_images/14715425-24448b60df04d2eb.png')

#下载在当前目录

#第一次下载,文件将会存为:14715425-24448b60df04d2eb.png

#再次下载,文件将会存为:14715425-24448b60df04d2eb(1).png

# requests模块:

注意,与urllibe.request不同,是两个不同的模块

1、安装

(1) 在线

# pip3 install requests

(2) 离线

# cd requests\_pkgs/

# pip3 install \*

方法、描述

requests.request() 构造一个请求，支持以下各种方法

requests.get() 向html网页提交get请求的方法

requests.post() 向html网页提交post请求的方法

requests.head() 获取html头部信息的主要方法

requests.put() 向html网页提交put请求的方法

requests.options() 向html网页提交options请求的方法

requests.patch() 向html网页提交局部修改的请求

requests.delete() 向html网页提交删除的请求

## 1-requests.request()方法

格式：requests.request(method, url, \*\*kwargs)

method : 请求方式，如： GET/HEAD/POST等等

url : 拟获取页面的url链接

\*\*kwargs: 控制访问的参数，共13个

访问控制参数一共有13个，其是可选的，使用时，需要用命名方法来调用，

例如：params=kv（kv 为事先定义好的一个字典）

1. params 字典或字节序列，作为参数增加到 url 中
2. data 字典、字节序列或文件对象，作为 Request 的内容
3. json JSON 格式的数据，作为 Request 的内容
4. headers 字典，可以用其来控制访问链接的 HTTP 头
5. cookies 字典或 CookieJar，Request 中的 cookie
6. auth 元组，支持 HTTP 认证功能
7. files 字典类型，向服务器传输文件时使用的字段
8. timeout 设定超时时间，以秒为单位，如果设定时间内未返回，将产生一个 TimeoutError 异常
9. proxies 字典类型，设定访问代理服务器，还可增加登录认证，能有效的隐藏用户爬取网页的源 IP，从而防止对爬虫的逆追踪
10. allow\_redirects 重定向开关，True/False，默认为True
11. stream 获取内容立即下载开关，默认为True
12. verify 认证 SSL 证书开关，默认为True
13. cert 保存本地 SSL 证书路径的字段

## 2- requests.get() 方法

1）作用：通过给定 url 来构造一个向 HTTP 请求资源的 Request 对象

2）返回值：返回一个包含 HTTP 资源的 Response 对象，其包含从 HTTP 返回的所有相关资源

3）格式：r = requests.get(url, \*\*kwargs)

## 3- requests.head() 方法

1）作用：通过给定 url 来构造一个向 HTTP 请求资源的 Request 对象

2）返回值：返回一个包含 HTTP 资源的 Response 对象，其包含从 HTTP 返回的所有头部资源

3）格式：requests.head(url, \*\*kwargs)

## 4- requests.post() 方法

1）作用：向 url 提交新增数据，格式为字典，自动编码为 form(表单)

2）格式：requests.post(url, \*\*kwargs)

## 5- requests.put() 方法

1）作用：向 url 提交新增数据，格式为字符串，自动编码为 data

2）格式：requests.put(url, \*\*kwargs)

## 6- requests.patch() 方法

1）作用：向 url 提交局部修改请求

2）格式：requests.patch(url,, \*\*kwargs)

## 7- requests.delete() 方法

1）作用：向 url 提交删除请求

2）格式：requests.delete(url, \*\*kwargs)

## request使用

**(1) 访问文本页面**

>>> r = requests.get('http://www.sogou.com')

>>> r.text

**(2) 访问非文内容(图片、视频)**

>>> r = requests.get('https://upload-images.jianshu.io/upload\_images/14715425-24448b60df04d2eb.png')

>>> with open('/tmp/jianshu.png', 'wb') as fobj:

... fobj.write(r.content)

# eog /tmp/jianshu.png

**(3) 访问json格式数据**

>>> r = requests.get('http://www.weather.com.cn/data/zs/101010100.html')

>>> r.json()

**(4) 字符编码**

>>> r.encoding

'ISO-8859-1' # 默认的字符编码无法准确地显示汉字

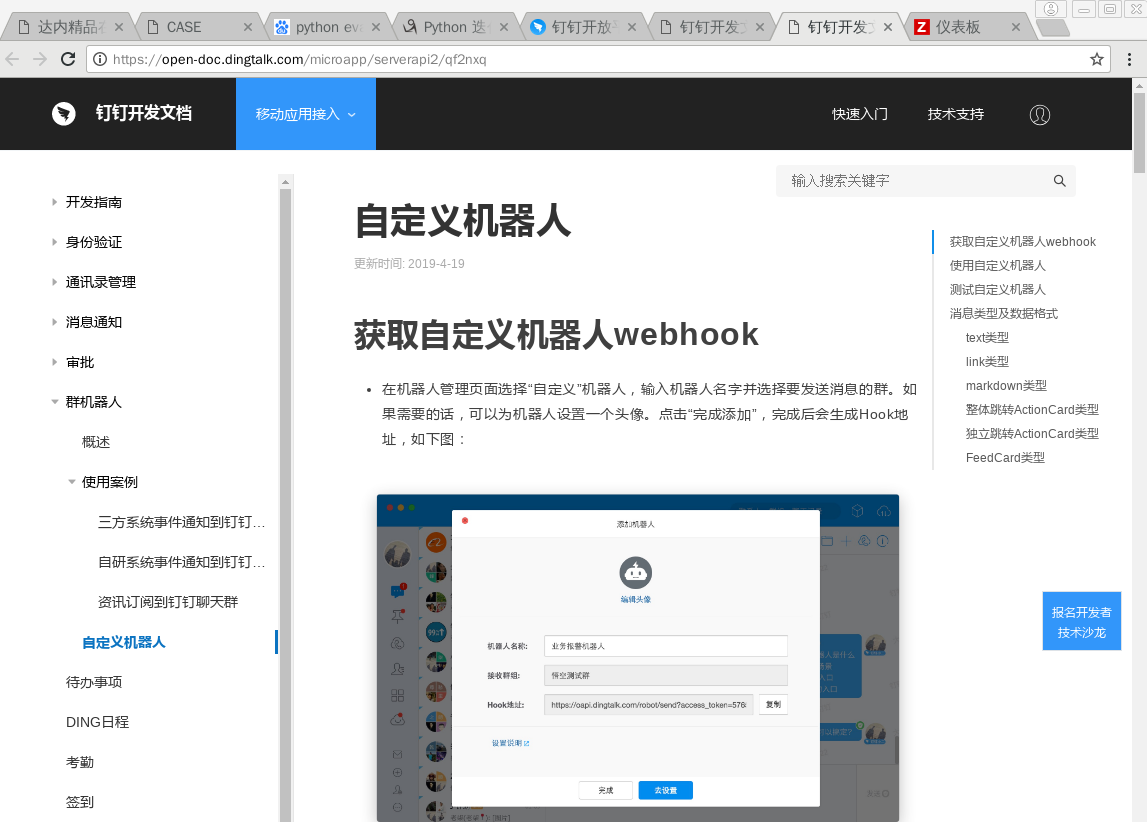
>>> r.encoding = 'utf8'

>>> r.json()

* 网络上的资源均有其特定的编码，如果没有编码，将无法通过有效的解析方式使得人类可读，因此我们需要通过编码来进行解析
* r.encoding 是通过对 header进行分析,如果其中不存在 charset 字段，则认为编码为 ISO‐8859‐1
* r.apparent\_encoding 根据网页内容而分析出的编码方式，其解析方式要比 r.encoding 更为准确，可看作是 r.encoding 的备选
* r.text 则是根据 r.encoding 来显示网页内容

## @@@自定义钉钉机器人:

访问:https://open-doc.dingtalk.com,搜索自定义机器人可获得开发调用说明



也可跟着张志刚讲师的简书100例里面的

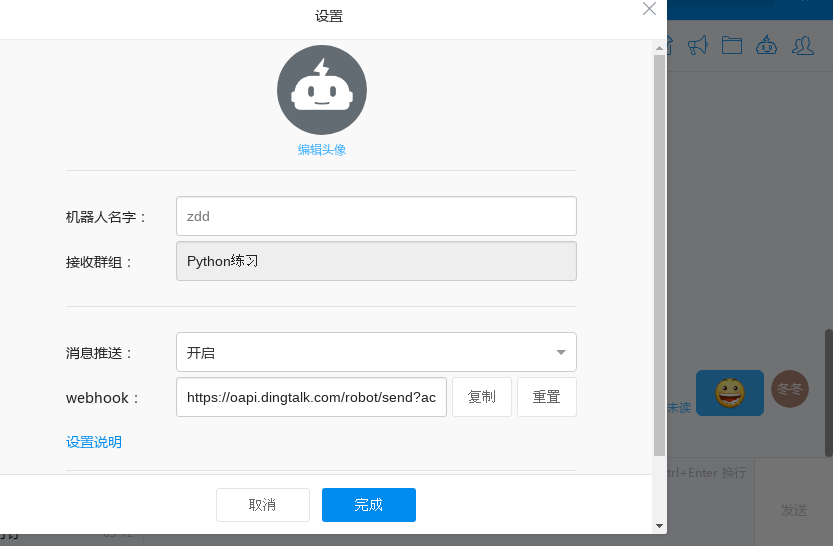


**代码案例**

#!/usr/bin/env python3  
**import** json  
**import** requests  
**import** sys  
  
**def send\_msg(**url, reminders, msg**):** headers **= {'Content-Type': 'application/json;charset=utf-8'}** data **= {  
 "msgtype": "text"**, # 发送消息类型为文本  
 **"at": {** # "atMobiles": reminders, #@某一个人  
 **"isAtAll": False**, #不@所有人  
 **}**,  
 **"text": {  
 "content":** msg, #消息正文  
 **}  
 }** r **=** requests.post**(**url, data**=**json.dumps**(**data**)**, headers**=**headers**)  
 return** r.text  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** msg **=** sys.argv**[**1**]** reminders **= ['13973169942']** # 特殊提醒要查看的人,就是@某人一下  
 url **= 'https://oapi.dingtalk.com/robot/send?access\_token=0f8f9b33b9d3b5e09082306d6ae56deee996718e5d1acd339906af2b3dd0b1aa'** #url地址来源与钉钉机器人管理复制webhook地址而来

print**(**send\_msg**(**url, reminders, msg**))**

url地址获取:webhook



**实际使用:**

[root@room9pc01 zdd-repository]# python3 dingding.py test

{"errmsg":"ok","errcode":0}

[root@room9pc01 zdd-repository]# python3 dingding.py hallo 我是机器人

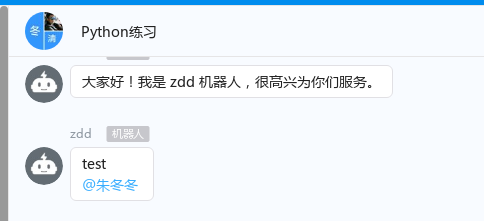
{"errmsg":"ok","errcode":0}

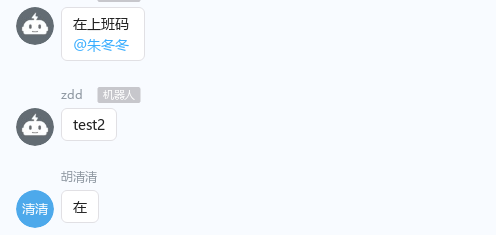
[root@room9pc01 zdd-repository]# python3 dingding.py 在上班码

{"errmsg":"ok","errcode":0}

[root@room9pc01 zdd-repository]# python3 dingding.py test2

{"errmsg":"ok","errcode":0}





头部详解:

headers = {'Content-Type': 'application/json;charset=utf-8'}

Content-Type

application/json

charset=utf-8

## @@@自定义机器人2:

**import** json  
**import** requests  
**import** getpass  
**def dingtalk(**url**):** headers **= {'Content-Type': 'application/json;charset=utf-8'}** data **= {  
 "msgtype": "link"**,  
 **"link": {  
 "text": "这个即将发布的新版本，创始人陈航（花名“无招”）称它为“红树林”。而在此之前，每当面临重大升级，产品经理们都会取一个应景的代号，这一次，为什么是“红树林”？"**,  
 **"title": "时代的火车向前开"**,  
 **"picUrl": "https://upload.jianshu.io/users/upload\_avatars/12347101/6b21f2d2-de50-4c8f-ba23-bf7f80ba77aa.jpg"**,  
 **"messageUrl": "http://www.baidu.cn"  
 }  
}** #以上内容均由钉钉官方公布的格式,只需复制粘贴,按照格式填写内容即可  
 r **=** requests.post**(**url, data**=**json.dumps**(**data**)**, headers**=**headers**)  
 return** r.json**()  
if** \_\_name\_\_ **== '\_\_main\_\_':** # url = '此处填写webhook的内容'  
 url **= 'https://oapi.dingtalk.com/robot/send?access\_token=0f8f9b33b9d3b5e09082306d6ae56deee996718e5d1acd339906af2b3dd0b1aa'** print**(**dingtalk**(**url**))**

**实际使用:**

[root@room9pc01 zdd-repository]# python3 dingding2.py

{'errmsg': 'ok', 'errcode': 0}

[root@room9pc01 zdd-repository]# python3 dingding2.py

{'errmsg': 'ok', 'errcode': 0}



# Cookie

# zabbix编程

**import** psutil  
  
temps **=** psutil.sensors\_temperatures**()**print**(**temps**)** #返回一个字典  
  
 #输出:{'acpitz': [shwtemp(label='', current=27.8, high=119.0, critical=119.0), shwtemp(label='', current=29.8, high=119.0, critical=119.0)], 'coretemp': [shwtemp(label='Physical id 0', current=36.0, high=84.0, critical=100.0), shwtemp(label='Core 0', current=35.0, high=84.0, critical=100.0), shwtemp(label='Core 1', current=33.0, high=84.0, critical=100.0), shwtemp(label='Core 2', current=32.0, high=84.0, critical=100.0), shwtemp(label='Core 3', current=36.0, high=84.0, critical=100.0)]}  
  
cpu0\_temp **=** temps**['coretemp'][**0**]**print**(**cpu0\_temp**)** #输出:shwtemp(label='Physical id 0', current=36.0, high=84.0, critical=100.0)  
  
print**('%s: %s' % (**cpu0\_temp.label, cpu0\_temp.current**))** #输出:Physical id 0: 36.0

## 部署zabbix

地址是:192.168.1.11/zabbix/index.php

## zabbixAPI

https://www.zabbix.com/documentation/3.4/zh/manual 访问官方手册API部分



## JSON-RPC

JSON-RPC，是一个无状态且轻量级的远程过程调用（RPC）传送协议，其传递内容透过 JSON 为主。相较于一般的 REST 透过网址（如 GET /user）调用远程服务器，JSON-RPC 直接在内容中定义了欲调用的函数名称（如 {"method": "getUser"}），这也令开发者不会陷于该使用 PUT 或者 PATCH 的问题之中。 本规范主要定义了一些数据结构及其相关的处理规则。它允许运行在基于 Socket、HTTP 等诸多不同消息传输环境的同一进程中。其使用 JSON（RFC 4627）作为数据格式。

由于 JSON-RPC 使用JSON，它具有与其相同的类型系统。JSON可以表示四个基本类型（String、Numbers、Boolean 和 Null）和两个结构化类型（Objects 和 Array）。任何时候文档涉及JSON数据类型，第一个字母都必须大写：Object、Array、String、Number、Boolean、Null。包括 True 和 False 也要大写。

**请求对象**

发送一个请求对象至服务端代表一个RPC调用，一个请求对象包含下列成员：

* jsonrpc
* method
* params
* id

服务端必须回答相同的值如果包含在响应对象。这个成员用来两个对象之间的关联上下文。在请求对象中不建议使用 NULL 作为 id 值，因为该规范将使用空值认定为未知id的请求。另外，由于JSON-RPC 1.0 的通知使用了空值，这可能引起处理上的混淆。使用小数是不确定性的，因为许多十进制小数不能精准的表达为二进制小数。

**响应对象**

当发起一个RPC调用时，除通知之外，服务端都必须回复响应。响应表示为一个 JSON 对象，使用以下字段：

* jsonrpc
* result
* error
* id

响应对象必须包含 result 或 error 字段，但两个字段不能同时存在。

**错误对象**

当一个RPC调用遇到错误时，返回的响应对象必须包含错误成员参数，并且为带有下列成员参数的对象：

* code
* message
* data

/api\_jsonrpc.php 在nginx网站目录内

[root@zabbix zabbix]# ls /usr/local/nginx/html/zabbix/api\_jsonrpc.php

/usr/local/nginx/html/zabbix/api\_jsonrpc.php #在nginx网站目录内

url = 192.168.1.11/zabbix/api\_jsonrpc.php #192.168.1.11为zabbix服务器地址

## @@@连接zabbix,获取zabbix版本

**import** requests  
**import** json  
  
url **= 'http://192.168.1.11/zabbix/api\_jsonrpc.php'**headers **= {'Content-Type': 'application/json-rpc'}**data **= {  
 "jsonrpc": "2.0"**, #jsonrpc版本,固定的  
 **"method": "apiinfo.version"**, #手册上提供的  
 **"params": []**,  
 **"id":** 1 #随便给数字  
**}** #zabbix的官网api手册上复制下来  
  
r **=** requests.post**(**url,headers**=**headers , data **=** json.dumps**(**data**))**print**(**r.json**())** #返回值,主要是result部分  
  
 #输出{'jsonrpc': '2.0', 'result': '3.4.4', 'id': 1}  
 #result:zabbix的版本

**data 的来源:**



请求里面的内容,是上面案例的data,

响应是上面案例执行后返回的结果:经过json转换即输出结果,

## @@@利用zabbix的API接口操作zabbix

###--zabbix  
**import** requests  
**import** json  
  
url **= 'http://192.168.1.11/zabbix/api\_jsonrpc.php'** #注意api\_jsonrpc.php文件所在位置,源安装包内有,  
headers **= {'Content-Type': 'application/json-rpc'}** #头部信息  
###获取版本###  
data **= {  
 "jsonrpc": "2.0"**, #jsonrpc版本,固定的  
 **"method": "apiinfo.version"**, #手册上提供的  
 **"params": []**,  
 **"id":** 1 #随便给数字  
**}** #zabbix的官网api手册上复制下来  
###输出:{'jsonrpc': '2.0', 'result': '3.4.4', 'id': 101}  
###zabbix的版本信息为:3.4.4  
  
#####获取授权码  
# data = {  
# "jsonrpc": "2.0",  
# "method": "user.login",  
# "params": {  
# "user": "Admin", #zabbix的管理员用户  
# "password": "zabbix" #管理员用户的密码  
# },  
# "id": 1  
# }  
####输出:{'jsonrpc': '2.0', 'result': '561c47b7b2242a0c0d8260db7de9e47d', 'id': 1}  
####授权码为:561c47b7b2242a0c0d8260db7de9e47d  
  
###获取组信息  
# data = {  
# "jsonrpc": "2.0",  
# "method": "hostgroup.get",  
# "params": {  
# "output": "extend",  
# "filter": {  
# "name": [  
# "Zabbix servers",  
# "Linux servers"  
# ]  
# }  
# },  
# "auth": "561c47b7b2242a0c0d8260db7de9e47d", #使用上一步查询出来的授权码  
# "id": 1  
# }  
###输出:{'jsonrpc': '2.0', 'result': [{'groupid': '2', 'name': 'Linux servers', 'internal': '0', 'flags': '0'}, {'groupid': '4', 'name': 'Zabbix servers', 'internal': '0', 'flags': '0'}], 'id': 1}  
###的到组信息为:'groupid': '2',  
  
###获取Template os linux模板ID  
# 获取Template os linux模板ID  
# data = {  
# "jsonrpc": "2.0",  
# "method": "template.get",  
# "params": {  
# "output": "extend",  
# "filter": {  
# "host": [  
# "Template OS Linux",  
# # "Template OS Windows"  
# ]  
# }  
# },  
# "auth": "561c47b7b2242a0c0d8260db7de9e47d",  
# "id": 1  
# }  
  
#输出为:{'jsonrpc': '2.0', 'result': [{'proxy\_hostid': '0', 'host': 'Template OS Linux', 'status': '3', 'disable\_until': '0', 'error': '', 'available': '0', 'errors\_from': '0', 'lastaccess': '0', 'ipmi\_authtype': '-1', 'ipmi\_privilege': '2', 'ipmi\_username': '', 'ipmi\_password': '', 'ipmi\_disable\_until': '0', 'ipmi\_available': '0', 'snmp\_disable\_until': '0', 'snmp\_available': '0', 'maintenanceid': '0', 'maintenance\_status': '0', 'maintenance\_type': '0', 'maintenance\_from': '0', 'ipmi\_errors\_from': '0', 'snmp\_errors\_from': '0', 'ipmi\_error': '', 'snmp\_error': '', 'jmx\_disable\_until': '0', 'jmx\_available': '0', 'jmx\_errors\_from': '0', 'jmx\_error': '', 'name': 'Template OS Linux', 'flags': '0', 'templateid': '10001', 'description': '', 'tls\_connect': '1', 'tls\_accept': '1', 'tls\_issuer': '', 'tls\_subject': '', 'tls\_psk\_identity': '', 'tls\_psk': ''}], 'id': 1}  
###获取的模板id为:'templateid': '10001',  
  
  
###创建主机  
# 创建主机web2，它在2号组中，使用10001模板  
# data = {  
# "jsonrpc": "2.0",  
# "method": "host.create",  
# "params": {  
# "host": "web2",  
# "interfaces": [ # 定义通过什么方式进行监控  
# {  
# "type": 1, # 1号类型是zabbix agent  
# "main": 1,  
# "useip": 1,  
# "ip": "192.168.1.254",  
# "dns": "",  
# "port": "10050"  
# }  
# ],  
# "groups": [  
# {  
# "groupid": "2"  
# }  
# ],  
# "templates": [  
# {  
# "templateid": "10001"  
# }  
# ],  
# "inventory\_mode": 0, # 主机资产记录  
# "inventory": {  
# "macaddress\_a": "01234",  
# "macaddress\_b": "56768"  
# }  
# },  
# "auth": "561c47b7b2242a0c0d8260db7de9e47d",  
# "id": 1  
# }  
  
##输出:{'jsonrpc': '2.0', 'result': {'hostids': ['10254']}, 'id': 1}  
##在zabbix服务里面,添加了一台被监控主机且,hostids:10254  
  
####获取主机信息  
# data = {  
# "jsonrpc": "2.0",  
# "method": "host.get",  
# "params": {  
# "output": "extend",  
# "filter": {  
# "host": [  
# # "Zabbix server",  
# # "Linux server"  
# "web2"  
# ]  
# }  
# },  
# "auth": "561c47b7b2242a0c0d8260db7de9e47d",  
# "id": 1  
# }  
  
###输出为:{'jsonrpc': '2.0', 'result': [{'hostid': '10254', 'proxy\_hostid': '0', 'host': 'web2', 'status': '0', 'disable\_until': '1556262109', 'error': 'Get value from agent failed: cannot connect to [[192.168.1.254]:10050]: [111] Connection refused', 'available': '2', 'errors\_from': '1556261344', 'lastaccess': '0', 'ipmi\_authtype': '-1', 'ipmi\_privilege': '2', 'ipmi\_username': '', 'ipmi\_password': '', 'ipmi\_disable\_until': '0', 'ipmi\_available': '0', 'snmp\_disable\_until': '0', 'snmp\_available': '0', 'maintenanceid': '0', 'maintenance\_status': '0', 'maintenance\_type': '0', 'maintenance\_from': '0', 'ipmi\_errors\_from': '0', 'snmp\_errors\_from': '0', 'ipmi\_error': '', 'snmp\_error': '', 'jmx\_disable\_until': '0', 'jmx\_available': '0', 'jmx\_errors\_from': '0', 'jmx\_error': '', 'name': 'web2', 'flags': '0', 'templateid': '0', 'description': '', 'tls\_connect': '1', 'tls\_accept': '1', 'tls\_issuer': '', 'tls\_subject': '', 'tls\_psk\_identity': '', 'tls\_psk': ''}], 'id': 1}  
###查询出的主机'hostid': '10254'  
  
r **=** requests.post**(**url,headers**=**headers , data **=** json.dumps**(**data**))**print**(**r.json**())** #返回值,result部分为主要信息

## @@@121-配置zabbix通过钉钉机器人报警作业

# Jenkins基础:

Jenkins概述:

Jenkins是一个开源软件项目，是基于Java开发的一种持续集成工具，用于监控持续重复的工作，旨在提供一个开放易用的软件平台，使软件的持续集成变成可能。

Jenkins功能包括：

1、持续的软件版本发布/测试项目。

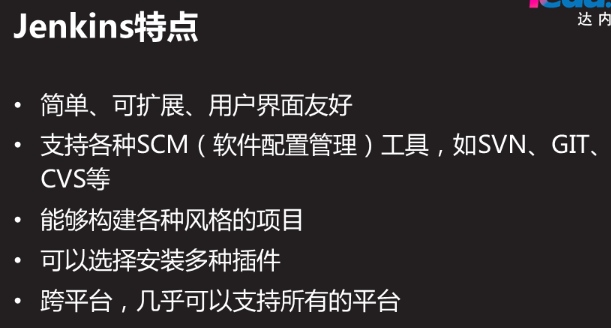
2、监控外部调用执行的工作。

jenkins: 实现CI,实行下载、编译程序等

编译执行：C / go

解释执行：python / php / shell

jenkins特点



持续集成:CI

持续交互:CD

## 安装Jenkins

配置node4成为jenkins服务器

1、node4.tedu.cn eth0: 192.168.4.4 eth1:自动获得，连接default网络

(1) node4的第二块网卡采连接到default网络

(2) 删除virbr0网桥

[root@node4 ~]# ifconfig virbr0 down

[root@node4 ~]# brctl delbr virbr0

(3) 配置IP

[root@node4 ~]# LANG=C nmtui

Edit a connection -> Add -> Ethernet -> Profile name: eth1; Device: eth1

-> OK

[root@node4 ~]# ifup eth1

2、安装

拷贝jenkins rpm包到node4

[root@node4 ~]# rpm -ihv jenkins-2.138.2-1.1.noarch.rpm

3、配置jenkins

jenkins是java程序，需要安装java

# yum install -y java

[root@node4 ~]# systemctl start jenkins

[root@node4 ~]# systemctl enable jenkins

访问 http://192.168.4.4:8080

按向导初始化。其中安装插件时，什么也不要选择。“选择插件来安装” -> “无”

“创建第一个管理员”界面，选择右下角“使用admin账号登陆” -> 保存并完成

进入管理页面后，点击右上角的admin -> 左侧的设置 -> 修改密码

4、安装插件

(1) 使用国内镜像

首页 -> 系统管理 -> 插件管理 -> 高级 -> 升级站点:

https://mirrors.tuna.tsinghua.edu.cn/jenkins/updates/update-center.json

-> 提交

(2) 安装插件

可选插件 -> git parameter -> 直接安装 -> 安装完成后重启Jenkins(空闲时)

(3) 在jenkins服务器上安装git

[root@node4 ~]# yum install -y git

(4)如果在线安装不上，可以先下载，下载网址是：

http://updates.jenkins-ci.org/download/plugins/git-parameter/

下载后，在首页 -> 系统管理 -> 插件管理 -> 高级 -> 上传插件

## 上线流程1--git使用

gitlab服务地址:192.168.4.101

群组:devops

项目:myweb

### git上传至gitlab服务器

[root@room9pc01 ~]# mkdir /tmp/myweb

[root@room9pc01 ~]# git init /tmp/myweb

初始化空的 Git 版本库于 /tmp/myweb/.git/

[root@room9pc01 ~]# cd /tmp/myweb

### 第一个版本编辑并上传

[root@room9pc01 myweb]# echo '<h1>Hello World</h1>' > index.html

[root@room9pc01 myweb]# git add .

[root@room9pc01 myweb]# git commit -m "myweb version 1.0"

[master（根提交） 1bea7c0] myweb version 1.0

1 file changed, 1 insertion(+)

create mode 100644 index.html

[root@room9pc01 myweb]# git tag 1.0 #打标签

[root@room9pc01 myweb]# git tag #查看标签

1.0

### 第二个版本编辑上传

[root@room9pc01 myweb]# echo '<h2>my web 2.0</h2>' >> index.html

[root@room9pc01 myweb]# git add .

[root@room9pc01 myweb]# git commit -m "my web 2.0"

[master c5de119] my web 2.0

1 file changed, 1 insertion(+)

[root@room9pc01 myweb]# git tag 2.0

[root@room9pc01 myweb]# pwd

/tmp/myweb

[root@room9pc01 myweb]# git status

# 位于分支 master

无文件要提交，干净的工作区

### git重命名远程服务

[root@room9pc01 myweb]# git remote rename origin old-origin

error: 不能重命名配置小节 'remote.origin' 到 'remote.old-origin'

### git添加远程仓库

[root@room9pc01 myweb]# git remote add origin <http://192.168.4.101/devops/myweb.git>

#添加的远程仓库名为origin 地址为:<http://192.168.4.101/devops/myweb.git>

### git推送所有提交

[root@room9pc01 myweb]# git push -u origin --all #推送给名为origin对应的仓库

Username for 'http://192.168.4.101': zdd

Password for 'http://zdd@192.168.4.101':

Counting objects: 6, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (6/6), 471 bytes | 0 bytes/s, done.

Total 6 (delta 0), reused 0 (delta 0)

To http://192.168.4.101/devops/myweb.git

\* [new branch] master -> master

分支 master 设置为跟踪来自 origin 的远程分支 master。

### git推送所有的标签(tag)

[root@room9pc01 myweb]# git push -u origin --tags

Username for 'http://192.168.4.101': zdd

Password for 'http://zdd@192.168.4.101':

Total 0 (delta 0), reused 0 (delta 0)

To http://192.168.4.101/devops/myweb.git

\* [new tag] 1.0 -> 1.0

\* [new tag] 2.0 -> 2.0

[root@room9pc01 myweb]#

### git相关查询命令

[root@room9pc01 day05]# git status #查询提交状态

# 位于分支 master

# 未跟踪的文件:

# （使用 "git add <file>..." 以包含要提交的内容）

#

# ../day02/forkping2.py

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）

[root@room9pc01 day05]# git branch #查询分支

\* master

[root@room9pc01 day05]# git config --list #查询远程仓库列表

user.name=zdd

user.email=635518693@qq.com

core.editor=vim

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

remote.origin.url=https://github.com/Mrzhangzhg/nsd\_2018.git

remote.origin.fetch=+refs/heads/\*:refs/remotes/origin/\*

branch.master.remote=origin

branch.master.merge=refs/heads/master

## 上线流程2 -Jenkins拉取内容

Jenkins服务为:192.168.4.4:8080

### jenkins拉取gitlab上的代码

相当于用在jenkins服务器上执行,git clone

在jenkins进行如下操作

创建自由风格的软件项目 -> 参数化构建过程 -> 添加参数选git parameter

-> name: webver, parameter type: tag, Default Value: origin/master

-> 源码管理 -> Repository URL: http://192.168.4.1/devops/myweb.git

-> Branches to build: $webver ->保存

执行工程，拉取代码：Build with Parameters -> 选择1.0后构建

-> 左下角的Build History，成功显示蓝球，失败是红球

#以上webver是个变量,

### 查看拉取下来的代码：

[root@node4 ~]# ls /var/lib/jenkins/workspace/

[root@node4 ~]# ls /var/lib/jenkins/workspace/myweb/

index.html

[root@node4 ~]# cat /var/lib/jenkins/workspace/myweb/index.html

<h1>Hello World</h1>

<h2>my web 2.0</h2>

[root@node4 ~]#

## 上线流程3-Jenkins创建web下载

Jenkins服务器上安装apache,

[root@node4 ~]# yum -y install httpd

[root@node4 ~]# systemctl start httpd

[root@node4 ~]# systemctl enable httpd

### 创建apache目录

这里只创建/var/www/html/deploy/packages目录,

/var/www/html/deploy: #将用来存储版本文件

/var/www/html/deploy/live\_ver： #将用来记录最新版本信息文件

/var/www/html/deploy/last\_ver： #将用来记录前一个版本文件

/var/www/html/deploy/packages/: #将用来存储压缩后的软件包和它的md5值

创建目录:

[root@node4 ~]# mkdir -p /var/www/html/deploy/packages/

[root@node4 ~]# chown -R jenkins:jenkins /var/www/html/deploy/

#packages目录将用来装所有版本的包

### 将check out的版本放到子目录中

在源码管理下面，Additional Behaviours -> check out to a sub-directory

-> Local subdirectory for repo: myweb-$webver #在Jenkins上面设置myweb项目

#作用就是将新拉取下来的版本文件全部放入到这个myweb项的 myweb-$webver目录里面,

#myweb-1.0, 的组成就是myweb-$webver $webver就是构建的git服务上的tag标签

验证:

[root@node4 ~]# ls /var/lib/jenkins/workspace/myweb

index.html myweb-1.0 #多了myweb-1.0目录

[root@node4 ~]# ls /var/lib/jenkins/workspace/myweb/myweb-1.0/

index.html

[root@node4 ~]# cat /var/lib/jenkins/workspace/myweb/myweb-1.0/index.html

<h1>Hello World</h1> #myweb-1.0目录里面的内容就是git上1.0版本的内容

**(2)将软件打包放到apache相关目录下**

构建 -> 增加构建步骤 -> 执行shell-->填入以下shell代码 #在Jenkins上面设置myweb项目

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

deploy\_dir=/var/www/html/deploy

cp -r myweb-$webver $deploy\_dir/packages/

cd $deploy\_dir/packages

rm -fr myweb-$webver/.git

tar czf myweb-$webver.tar.gz myweb-$webver

rm -rf myweb-$webver

md5sum myweb-$webver.tar.gz | awk '{print $1}' > myweb-$webver.tar.gz.md5

cd $deploy\_dir

[ -f live\_ver ] && cat live\_ver > last\_ver

echo $webver > live\_ver

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

[root@node4 ~]# ls /var/lib/jenkins/workspace/myweb

index.html myweb-1.0

[root@node4 ~]# ls /var/www/html/deploy

live\_ver packages

[root@node4 ~]# cat /var/www/html/deploy/live\_ver

1.0

[root@node4 ~]# ls /var/www/html/deploy/packages/

myweb-1.0.tar.gz myweb-1.0.tar.gz.md5

[root@node4 ~]# ls /var/www/html/deploy/packages/

myweb-1.0.tar.gz myweb-1.0.tar.gz.md5 myweb-2.0.tar.gz myweb-2.0.tar.gz.md5

[root@node4 ~]#

## @@@python程序,使得主机群更新版本

**import** requests  
**import** wget  
**import** os  
**import** hashlib  
**import** tarfile  
  
**def has\_new\_ver(**ver\_fname,ver\_url**):  
 if not** os.path.isfile**(**ver\_fname**):** #判断本地有没有版本文件  
 **return True  
  
 with** open**(**ver\_fname**) as** fobj**:** #如果存在则打开  
 local\_ver **=** fobj.read**()** #读取版本文件内容赋值给local\_ver  
  
 r **=** requests.get**(**ver\_url**)** #get远程jenkins的版本文件  
 remote\_ver **=** r.text #读取内容.文本格式,读取赋值给remote\_ver  
  
 **if** local\_ver **!=** remote\_ver**:** #本地和远程版本信息对比  
 **return True  
 return False  
  
  
def has\_error(**fname,md5\_url**):** m **=** hashlib.md5**()** #创建md5文件  
 **with** open**(**fname,**'rb') as** fobj**:** #打开  
 **while True:** data **=** fobj.read**(**4096**)  
 if not** data**:  
 break** m.update**(**data**)** r **=** requests.get**(**md5\_url**)** #  
 **if** m.hexdigest**() ==** r.text.strip**():  
 return False  
 return True  
  
  
def deploy(**app\_fname**):** ##此时app\_fname为/var/www/download/myweb-1.0.tar.gz  
 #先解压缩  
 deploy\_dir **= '/var/www/deploy/'** tar **=** tarfile.open**(**app\_fname,**'r:gz')** tar.extractall**(**path**=**deploy\_dir**)** tar.close**()** #拼出解压目录的绝对路径  
 app\_path **=** os.path.basename**(**app\_fname**)** #只取myweb-1.0.tar.gz  
 app\_path **=** app\_path.replace**('.tar.gz'**,**'')** #只取myweb-1.0  
 app\_path **=** os.path.join**(**deploy\_dir,app\_path**)** #合并网页实际路径/var/www/deploy/myweb-1.0  
 # print(app\_path) #测试用  
 #创建连接,如果连接已存在,先删除它  
 link **= '/var/www/html/nsd1811'** #创建连接指向网页/var/www/deploy/myweb-1.0  
 **if** os.path.exists**(**link**):** #判断删除原链接  
 os.remove**(**link**)** os.symlink**(**app\_path,link**)  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** app\_dir **= '/var/www/download'** ver\_fname **= '/var/www/deploy/live\_ver'** ver\_url **= 'http://192.168.4.4/deploy/live\_ver'  
 if not** has\_new\_ver**(**ver\_fname,ver\_url**):** #使用has\_new\_ver函数判断有没有新版本  
 print**('没有新版本')** exit**(**1**)** #如果有新版本,则下载  
 r **=** requests.get**(**ver\_url**)** ver **=** r.text.strip**()** #获取服务器上的版本号  
 app\_url **= 'http://192.168.4.4/deploy/packages/myweb-%s.tar.gz' %** ver #最新版本包的地址  
 wget.download**(**app\_url, app\_dir**)** #下载最新版本的包  
  
 #校验下载的压缩包是否损坏  
 app\_fname **=** app\_url.split**('/')[-**1**]** app\_fname **=** os.path.join**(**app\_dir , app\_fname**)** #拼接压缩包的绝对路径  
  
 md5\_url **=** app\_url **+ '.md5'** #拼出md5值的网址  
  
 **if** has\_error**(**app\_fname, md5\_url**):** print**('文件已损坏')** os.remove**(**app\_fname**)** #如果文件损坏,则删除它  
 exit**(**2**)** print**('文件未损坏')** #如果下载的文件是完好的,则部署  
 deploy**(**app\_fname**)** #更新本地版本文件  
 **with** open**(**ver\_fname,**'w') as** fobj**:** fobj.write**(**r.text**)** #r在下载版本那requests.get(ver\_url)

# Ansible基础

ansible是基于python编写,,

安装ansible

(1) 在线

# pip3 install ansible

(2) 离线

# cd ansible\_pkg/

# pip3 install \*

## 配置vim自动缩进

* Yaml的缩进不能使用Tab键
* 建议缩进为两个空格
* 实现yml文件按tab键缩进两个空格,可以按以下方式对vim进行定制

(1) 配置vim

set ai # 自动缩进

set ts=4 # tab键是4空格

set et # 将tab转换成空格

set encoding=utf8 # 字符编码

autocmd FileType yaml setlocal sw=2 ts=2 et ai # 只有yaml文件生效的配置

[root@room9pc01 mylibraby]#vim ~/.vimrc

[root@room9pc01 mylibraby]# cat ~/.vimrc

set ai

set et

set ts=4

set encoding=utf8

autocmd FileType yaml setlocal sw=2 ts=2 et ai

#实现效果:以后的yaml文件中一次tab键就是相当于两个空格

## ansible应用

2、准备三台虚拟机，用于远程管理测试

node4.tedu.cn 192.168.4.4

node5.tedu.cn 192.168.4.5

node6.tedu.cn 192.168.4.6

(1) 在物理机上配置名称解析

[root@room8pc16 ~]# for i in {1..254}

> do

> echo -e "192.168.4.$i\tnode$i.tedu.cn\tnode$i" >> /etc/hosts

> done

(2) 收集所有服务器的密钥

[root@room8pc16 ~]# ssh-keyscan 192.168.4.{4..6} node{4..6} node{4..6}.tedu.cn >> ~/.ssh/known\_hosts

3、配置ansible工作环境

[root@room8pc16 day06]# mkdir myansible

[root@room8pc16 day06]# cd myansible

[root@room8pc16 myansible]# vim ansible.cfg

[defaults]

inventory = hosts

remote\_user = root

[root@room8pc16 myansible]# vim hosts

[dbservers]

node4.tedu.cn

[webservers]

node5.tedu.cn

node6.tedu.cn

测试

[root@room8pc16 myansible]# ansible all -m ping -k

### 通过playbook实现免密登陆、配置yum

(2) 编写playbook

[root@room8pc16 myansible]# vim authkey.yml

---

- name: configure auth key

hosts: all

tasks:

- name: upload public key

authorized\_key:

user: root

state: present

key: "{{ lookup('file', '/root/.ssh/id\_rsa.pub') }}"

(3) 检查语法

[root@room8pc16 myansible]# ansible-playbook --syntax-check authkey.yml

(4) 执行

[root@room8pc16 myansible]# ansible-playbook authkey.yml -k

### 拷贝yum文件到主机群,playbook方法

[root@room8pc16 myansible]# mkdir files

[root@room8pc16 myansible]# vim files/server.repo

[root@room8pc16 myansible]# vim yum.yml

---

- name: configure yum

hosts: all

tasks:

- name: upload yum repo file

copy:

src: files/server.repo

dest: /etc/yum.repos.d/server.repo

[root@room8pc16 myansible]# ansible-playbook --syntax-check yum.yml

[root@room8pc16 myansible]# ansible-playbook yum.yml

### 用playbook配置lamp

[root@room8pc16 myansible]# vim lamp.yml

---

- name: configure dbservers

hosts: dbservers

tasks:

- name: install mariadb

yum:

name: mariadb-server

state: latest

- name: start mariadb service

service:

name: mariadb

state: started

enabled: yes

- name: configure webservers

hosts: webservers

tasks:

- name: install apache

yum:

name: [httpd, php, php-mysql]

state: present

- name: start httpd service

service:

name: httpd

state: started

enabled: yes

[root@room8pc16 myansible]# ansible-playbook --syntax-check lamp.yml

[root@room8pc16 myansible]# ansible-playbook lamp.yml

### 查询模块ansible-doc

## Playbook

### 使用playbook

* playbook是Ansiblede配置,部署,;编排语言 cat
* 他们可以被描述为一个需要希望远程主机执行命令的方案,或者一组程序运行的命令集合
* playbook由一个到多个play组成
* 每个play可以指定哪些主机执行哪些任务
* 执行任务一般通过调用模块来实现

### Yaml简介

* playbooks的格式 cat是YAML
* 语法做到最小化,意在避免playbook 成为一种编程语言或是脚本
* 使用YAML是因为它像XML或Json是一种利于人们读写的数据格式

### Yaml语法

### 创建playbook

请看上面的ansible应用

## ansible编程

### 命名元组

>>>Point = namedtuple('Point' , ['x','y','z'])

>>>a = Point(10,15,8)

>>>a[0] #使用下标输出值

>>>10

>>>a.x #使用命名输出值

>>>10

>>>a[1]

>>>15

>>>a.y

>>>15

>>>a[2]

>>>8

>>>a.z

>>>8

## ansible常用属性

### Dataloader()函数解释

python中将yml文件转换成python数据类型的函数

# initialize needed objects  
#DataLoader() 用于查找并解析yml json ini文件,把他们转换成python的数据类型,----可查看笔记的yuml详细解析  
loader **=** DataLoader**()** # Takes care of finding and reading yaml, json and ini files  
passwords **=** dict**(**vault\_pass**='secret')**

yaml文件格式能够解释成python的数据类型，如把lamp.yml改为python数据类型如下：

[

'name': 'configure dbservers',

'hosts': 'dbservers',

'tasks:

[

'name': 'install mariadb',

'yum': {

'name': 'mariadb-server',

'state': 'latest'

}

],

[

'name': 'start mariadb'

'service': {

'name': 'mariadb',

'state': 'started',

'enabled': 'yes'

}

]

]

### VariableManager

**from** ansible.vars.manager **import** VariableManager

管理变量的类,包括主机,组,扩展等变量

### InventoryManager

**from** ansible.inventory.manager **import** InventoryManager

用于创建主机清单,主机清单的源采用配置文件或是逗号分开主机名字符串

### play

**from** ansible.playbook.play **import** Play

用于创建play对象,能够通过play\_source提供的信息自动创建任务对象

### TaskQueueManager

**from** ansible.executor.task\_queue\_manager **import** TaskQueueManager

用于处理进程池中的多进程,队列管理器负责加载play策略插件,以便在选定的主机上执行任务

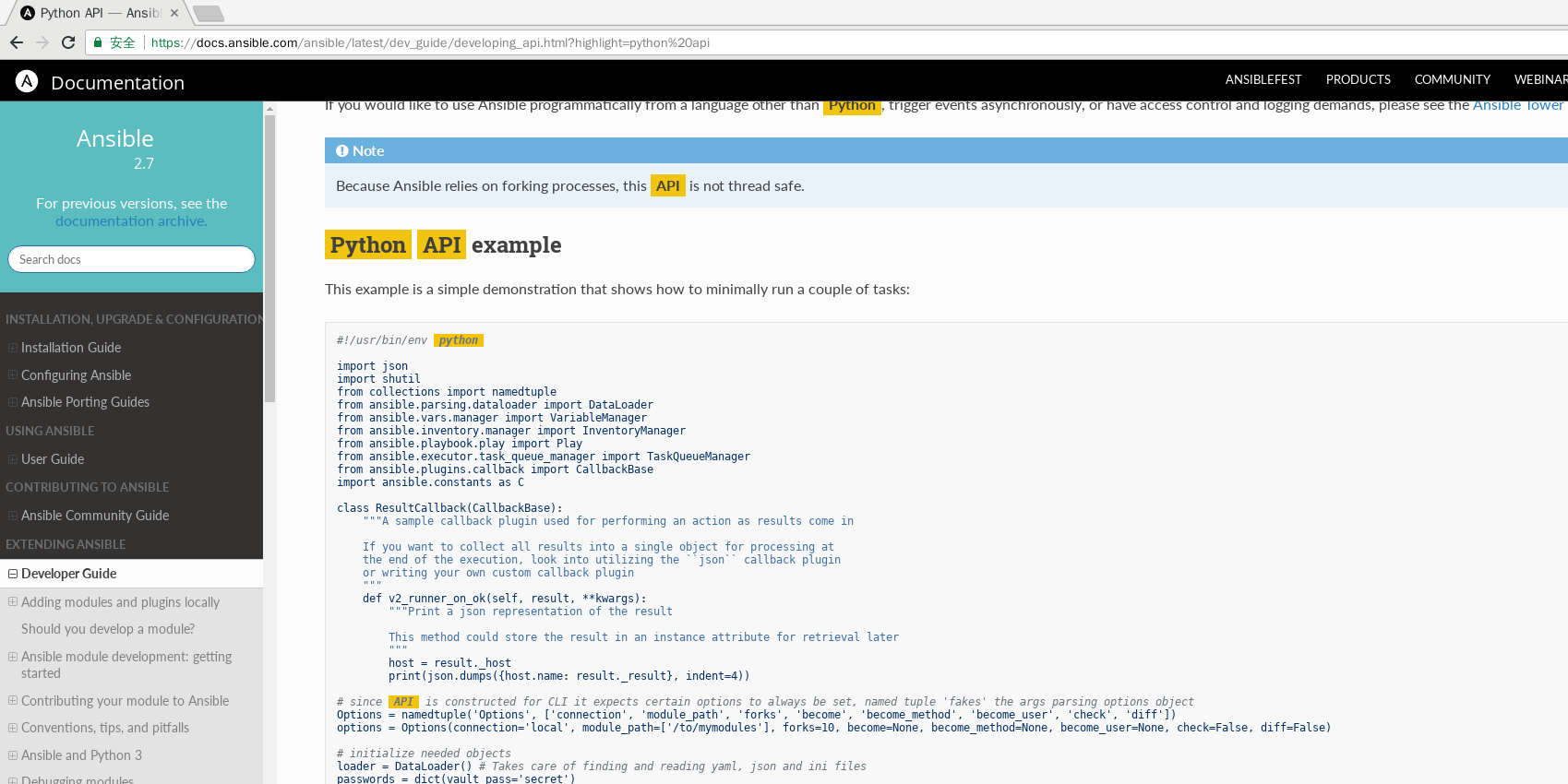
### C

**import** ansible.constants **as** C

存储ansible的一些预定义变量

## 修改官方源代码

https://docs.ansible.com #

相关

### 先删除自定义的类

#!/usr/bin/env python

import json

import shutil

from collections import namedtuple

from ansible.parsing.dataloader import DataLoader

.....

.....

from ansible.plugins.callback import CallbackBase

import ansible.constants as C

##删除自定义输出的类

class ResultCallback(CallbackBase):

"""A sample callback plugin used for performing an action as results come in

If you want to collect all results into a si相关ngle object for processing at

the end of the execution, look into utilizing the ``json`` callback plugin

or writing your own custom callback plugin

"""

def v2\_runner\_on\_ok(self, result, \*\*kwargs):

"""Print a json representation of the result

This method could store the result in an instance attribute for retrieval later

"""

host = result.\_host

print(json.dumps({host.name: result.\_result}, indent=4))

....

....

tqm = None

try:

tqm = TaskQueueManager(

inventory=inventory,

variable\_manager=variable\_manager,

loader=loader,

options=options,

passwords=passwords,

stdout\_callback=results\_callback, # Use our custom callback instead of the ``default`` callback plugin, which prints to stdout #调用上面的自定义输出,删除

)

result = tqm.run(play) # most interesting data for a play is actually sent to the callback's methods

....

....

相关

### 最终留下的原版

最终留下的原版,删除了所有的英文注释,反正自己也看不懂,实在想看去官网看

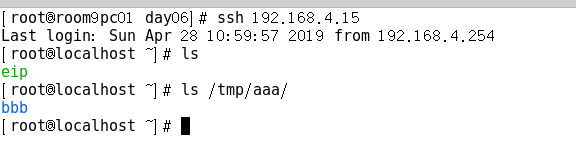
#!/usr/bin/env python  
**import** json  
**import** shutil  
**from** collections **import** namedtuple  
**from** ansible.parsing.dataloader **import** DataLoader  
**from** ansible.vars.manager **import** VariableManager  
**from** ansible.inventory.manager **import** InventoryManager  
**from** ansible.playbook.play **import** Play  
**from** ansible.executor.task\_queue\_manager **import** TaskQueueManager  
**from** ansible.plugins.callback **import** CallbackBase  
**import** ansible.constants **as** C  
  
**'''  
#connection='local' 连接方式:  
#module\_path=['/to/mymodules'] 定义模块位置,执行指令,例如-m shell除了搜索ansible默认的模块,还会到这个指定地方取搜索模块  
#forks=10 指定启动的进程数量  
#become=None 如果要使用普通用户远程登录服务器,指定要变成管理员方式,例如是由su还是sudo  
#check=False, 不真正执行,语言将要发生的变化  
#diff=False 执行  
'''**Options **=** namedtuple**('Options'**, **['connection'**, **'module\_path'**, **'forks'**, **'become'**, **'become\_method'**, **'become\_user'**, **'check'**, **'diff'])**options **=** Options**(**connection**='local'**, module\_path**=['/to/mymodules']**, forks**=**10, become**=None**, become\_method**=None**, become\_user**=None**, check**=False**, diff**=False)**#DataLoader() 用于查找并解析yml json ini文件,把他们转换成python的数据类型,----可查看笔记的yuml详细解析  
loader **=** DataLoader**()** # Takes care of finding and reading yaml, json and ini files  
passwords **=** dict**(**vault\_pass**='secret')**#定义主机清单,可以将各主机用逗号隔开的字符串,也可以指定文件路径列表  
#sources=['/root/ansible/hosts'] 主机列表文件目录  
inventory **=** InventoryManager**(**loader**=**loader, sources**=['/root/ansible/hosts'])**# inventory = InventoryManager(loader=loader, sources='localhost,') 原语句  
最终  
variable\_manager **=** VariableManager**(**loader**=**loader, inventory**=**inventory**)**play\_source **=** dict**(** #创建一个执行源  
 name **= "Ansible Play"**, #play的名字  
 hosts **= 'webservers'**, #指定在哪些主机上执行命令,实际中有webservers主机组  
 gather\_facts **= 'no'**,  
 tasks **= [** dict**(**action**=**dict**(**module**='shell'**, args**='ls')**, register**='shell\_out')**,  
 dict**(**action**=**dict**(**module**='debug'**, args**=**dict**(**msg**='{{shell\_out.stdout}}')))  
 ]  
 )**#通过上面的配置创建一个play,  
play **=** Play**()**.load**(**play\_source, variable\_manager**=**variable\_manager, loader**=**loader**)**#创建任务管理器,执行play  
tqm **= None  
try:** tqm **=** TaskQueueManager**(** inventory**=**inventory,  
 variable\_manager**=**variable\_manager,  
 loader**=**loader,  
 options**=**options,  
 passwords**=**passwords,  
 **)** result **=** tqm.run**(**play**)  
finally:  
 if** tqm **is not None:** tqm.cleanup**()** shutil.rmtree**(**C.DEFAULT\_LOCAL\_TMP, **True)**

### 最终版(详细请看ad-hoc模式)

#!/usr/bin/env python  
**import** json  
**import** shutil  
**from** collections **import** namedtuple  
**from** ansible.parsing.dataloader **import** DataLoader  
**from** ansible.vars.manager **import** VariableManager  
**from** ansible.inventory.manager **import** InventoryManager  
**from** ansible.playbook.play **import** Play  
**from** ansible.executor.task\_queue\_manager **import** TaskQueueManager  
**from** ansible.plugins.callback **import** CallbackBase  
**import** ansible.constants **as** C  
**'''  
最终版  
1,加上函数  
2,将一些目录,主机组,执行的命令改成变量  
'''  
def adhoc(**sources, hosts, module, args**):  
 '''  
 #connection='local' 连接方式:  
 #module\_path=['/to/mymodules'] 定义模块位置,执行指令,例如-m shell除了搜索ansible默认的模块,还会到这个指定地方取搜索模块  
 #forks=10 指定启动的进程数量  
 #become=None 如果要使用普通用户远程登录服务器,指定要变成管理员方式,例如是由su还是sudo  
 #check=False, 不真正执行,语言将要发生的变化  
 #diff=False 执行  
 '''** Options **=** namedtuple**('Options'**, **['connection'**, **'module\_path'**, **'forks'**, **'become'**, **'become\_method'**, **'become\_user'**, **'check'**, **'diff'])** options **=** Options**(**connection**='ssh'**, module\_path**=['/to/mymodules']**, forks**=**10, become**=None**, become\_method**=None**, become\_user**=None**, check**=False**, diff**=False)** #DataLoader() 用于查找并解析yml json ini文件,把他们转换成python的数据类型,----可查看笔记的yuml详细解析  
 loader **=** DataLoader**()** # Takes care of finding and reading yaml, json and ini files  
 passwords **=** dict**()** # passwords = dict(vault\_pass='secret')  
  
 #定义主机清单,可以将各主机用逗号隔开的字符串,也可以指定文件路径列表  
 #sources=['/root/ansible/hosts'] 主机列表文件目录  
 inventory **=** InventoryManager**(**loader**=**loader, sources**=**sources**)** # inventory = InventoryManager(loader=loader, sources=['/root/ansible/hosts'])  
 # inventory = InventoryManager(loader=loader, sources='localhost,') 原语句  
 variable\_manager **=** VariableManager**(**loader**=**loader, inventory**=**inventory**)** play\_source **=** dict**(** #创建一个执行源  
 name **= "Ansible Play"**, #play的名字  
 hosts **=** hosts, #指定在哪些主机上执行命令,实际中有webservers主机组  
 # hosts = 'webservers', # 指定在哪些主机上执行命令,实际中有webservers主机组  
 gather\_facts **= 'no'**,  
 tasks **= [** dict**(**action**=**dict**(**module**=**module, args**=**args**)**, register**='shell\_out')**,  
 # module = module #调用的模块  
 # args = args #执行的命令  
 # dict(action=dict(module='shell', args='ls'), register='shell\_out'),  
 # dict(action=dict(module='debug', args=dict(msg='{{shell\_out.stdout}}')))  
 **]  
 )** #通过上面的配置创建一个play,  
 play **=** Play**()**.load**(**play\_source, variable\_manager**=**variable\_manager, loader**=**loader**)**

#创建任务管理器,执行play  
 tqm **= None  
 try:** tqm **=** TaskQueueManager**(** inventory**=**inventory,  
 variable\_manager**=**variable\_manager,  
 loader**=**loader,  
 options**=**options,  
 passwords**=**passwords,  
 **)** result **=** tqm.run**(**play**)  
 finally:  
 if** tqm **is not None:** tqm.cleanup**()** shutil.rmtree**(**C.DEFAULT\_LOCAL\_TMP, **True)  
  
if** \_\_name\_\_ **== '\_\_main\_\_':最终** sources **= ['/root/ansible/hosts']** hosts **= 'webservers'** module **= 'shell'** args **= 'mkdir -p /tmp/aaa/bbb'** adhoc**(**sources, hosts, module, args**)**

最终结果



## ad-hoc模式

Ad-Hoc（点对点）模式：ad-hoc模式就和以前的直连双绞线概念一样，是P2P的连接，所以也就无法与其它网络沟通了。一般无线终端设备像PMP、PSP、DMA等用的就是ad-hoc模式。 在

### 使用TaskQueueManager

* 创建TaskQueueManager实例,用于管理子进程,,通过主机列表和任务建立对象
* TaskQueueManager 需要主机清单参数,主机变量参数,连接选项等

### 导入模块

不管是执行ad-hoc还是playbook都需要以下模块

**import** json  
**import** shutil  
**from** collections **import** namedtuple  
**from** ansible.parsing.dataloader **import** DataLoader  
**from** ansible.vars.manager **import** VariableManager  
**from** ansible.inventory.manager **import** InventoryManager  
**from** ansible.playbook.play **import** Play  
**from** ansible.executor.task\_queue\_manager **import** TaskQueueManager  
**from** ansible.plugins.callback **import** CallbackBase  
**import** ansible.constants **as** C

### 设置参数

运行命令时候有很多参数需要指定,这些参数往往很长,可以先把他们提前定义出来

**def adhoc(**sources, hosts, module, args**):  
 '''  
 #connection='local' 连接方式:  
 #module\_path=['/to/mymodules'] 定义模块位置,执行指令,例如-m shell除了搜索ansible默认的模块,还会到这个指定地方取搜索模块  
 #forks=10 指定启动的进程数量  
 #become=None 如果要使用普通用户远程登录服务器,指定要变成管理员方式,例如是由su还是sudo  
 #check=False, 不真正执行,语言将要发生的变化  
 #diff=False 执行  
 '''** Options **=** namedtuple**('Options'**, **['connection'**, **'module\_path'**, **'forks'**, **'become'**, **'become\_method'**, **'become\_user'**, **'check'**, **'diff'])** options **=** Options**(**connection**='ssh'**, module\_path**=['/to/mymodules']**, forks**=**10, become**=None**, become\_method**=None**, become\_user**=None**, check**=False**, diff**=False)** #DataLoader() 用于查找并解析yml json ini文件,把他们转换成python的数据类型,----可查看笔记的yuml详细解析  
 loader **=** DataLoader**()** # Takes care of finding and reading yaml, json and ini files  
 passwords **=** dict**()** # passwords = dict(vault\_pass='secret')  
  
 #定义主机清单,可以将各主机用逗号隔开的字符串,也可以指定文件路径列表  
 #sources=['/root/ansible/hosts'] 主机列表文件目录  
 inventory **=** InventoryManager**(**loader**=**loader, sources**=**sources**)** # inventory = InventoryManager(loader=loader, sources=['/root/ansible/hosts'])  
 # inventory = InventoryManager(loader=loader, sources='localhost,') 原语句  
 variable\_manager **=** VariableManager**(**loader**=**loader, inventory**=**inventory**)** play\_source **=** dict**(** #创建一个执行源  
 name **= "Ansible Play"**, #play的名字  
 hosts **=** hosts, #指定在哪些主机上执行命令,实际中有webservers主机组  
 # hosts = 'webservers', # 指定在哪些主机上执行命令,实际中有webservers主机组  
 gather\_facts **= 'no'**,  
 tasks **= [** dict**(**action**=**dict**(**module**=**module, args**=**args**)**, register**='shell\_out')**,  
 # module = module #调用的模块  
 # args = args #执行的命令  
 # dict(action=dict(module='shell', args='ls'), register='shell\_out'),  
 # dict(action=dict(module='debug', args=dict(msg='{{shell\_out.stdout}}')))  
 **]  
 )** #通过上面的配置创建一个play,  
 play **=** Play**()**.load**(**play\_source, variable\_manager**=**variable\_manager, loader**=**loader**)**

### 执行ad-hoc命令

#创建任务管理器,执行play  
tqm **= None  
try:** tqm **=** TaskQueueManager**(** inventory**=**inventory,  
 variable\_manager**=**variable\_manager,  
 loader**=**loader,  
 options**=**options,  
 passwords**=**passwords,  
 **)** result **=** tqm.run**(**play**)  
finally:  
 if** tqm **is not None:** tqm.cleanup**()** shutil.rmtree**(**C.DEFAULT\_LOCAL\_TMP, **True)**

### 主程序与传参

**if** \_\_name\_\_ **== '\_\_main\_\_':** sources **= ['/root/ansible/hosts']** hosts **= 'webservers'** module **= 'shell'** args **= 'mkdir -p /tmp/aaa/bbb'** adhoc**(**sources, hosts, module, args**)**

## 调用playbook

playbook编程概述

* playbook是Ansiblede配置,部署,;编排语言 cat
* 他们可以被描述为一个需要希望远程主机执行命令的方案,或者一组程序运行的命令集合
* playbook由一个到多个play组成
* 每个play可以指定哪些主机执行哪些任务
* 执行任务一般通过调用模块来实现
* 可以通过python编程的方式执行playbook

**from** collections **import** namedtuple  
**from** ansible.parsing.dataloader **import** DataLoader  
**from** ansible.vars.manager **import** VariableManager  
**from** ansible.inventory.manager **import** InventoryManager  
**from** ansible.executor.playbook\_executor **import** PlaybookExecutor  
 #PlaybookExecutor用于调用playbook的模块  
  
#具体参考ansible常用属性与官方源代码,这里是纵向排列,上面是横向排列而已  
Options **=** namedtuple**(  
 'Options'**,  
 **[**

**from collections import namedtuple\_x000B\_from ansible.parsing.dataloader import DataLoader\_x000B\_from ansible.vars.manager import VariableManager\_x000B\_from ansible.inventory.manager import InventoryManager\_x000B\_from ansible.executor.playbook\_executor import PlaybookExecutor\_x000B\_\_x000B\_Options = namedtuple(\_x000B\_ 'Options',\_x000B\_ [\_x000B\_ 'connection',\_x000B\_ 'remote\_user',\_x000B\_ 'ask\_sudo\_pass',\_x000B\_ 'verbosity',\_x000B\_ 'ask\_pass',\_x000B\_ 'module\_path',\_x000B\_ 'forks',\_x000B\_ 'become',\_x000B\_ 'become\_method',\_x000B\_ 'become\_user',\_x000B\_ 'check',\_x000B\_ 'listhosts',\_x000B\_ 'listtasks',\_x000B\_ 'listtags',**

**from collections import namedtuple\_x000B\_from ansible.parsing.dataloader import DataLoader\_x000B\_from ansible.vars.manager import VariableManager\_x000B\_from ansible.inventory.manager import InventoryManager\_x000B\_from ansible.executor.playbook\_executor import PlaybookExecutor\_x000B\_\_x000B\_Options = namedtuple(\_x000B\_ 'Options',\_x000B\_ [\_x000B\_ 'connection',\_x000B\_ 'remote\_user',\_x000B\_ 'ask\_sudo\_pass',\_x000B\_ 'verbosity',\_x000B\_ 'ask\_pass',\_x000B\_ 'module\_path',\_x000B\_ 'forks',\_x000B\_ 'become',\_x000B\_ 'become\_method',\_x000B\_ 'become\_user',\_x000B\_ 'check',\_x000B\_ 'listhosts',\_x000B\_ 'listtasks',\_x000B\_ 'listtags',\_x000B\_ 'syntax',\_x000B\_ 'sudo\_user',\_x000B\_ 'sudo',\_x000B\_ 'diff'\_x000B\_ ]\_x000B\_)\_x000B\_options = Options(\_x000B\_ connection='smart',\_x000B\_ remote\_user=None,\_x000B\_ ask\_pass=None,\_x000B\_ sudo\_user=None,\_x000B\_ forks=5,\_x000B\_ sudo=None,\_x000B\_ ask\_sudo\_pass=False,\_x000B\_ verbosity=5,\_x000B\_ module\_path=None,\_x000B\_ become=None,\_x000B\_ become\_method=None,\_x000B\_ become\_user=None,\_x000B\_ check=False,\_x000B\_ diff=False,\_x000B\_ listhosts=None,\_x000B\_ listtasks=None,\_x000B\_ listtags=None,\_x000B\_ syntax=None\_x000B\_)\_x000B\_loader = DataLoader()\_x000B\_passwords = dict()\_x000B\_inventory = InventoryManager(loader=loader, sources=['/root/ansible/hosts']) \_x000B\_ #定义相关参数,sources是hosts主机群文件\_x000B\_variable\_manager = VariableManager(loader=loader, inventory=inventory)\_x000B\_\_x000B\_def runpb(pb\_path):\_x000B\_ playbook = PlaybookExecutor(\_x000B\_ playbooks=pb\_path,\_x000B\_ inventory=inventory,\_x000B\_ variable\_manager=variable\_manager,\_x000B\_ loader=loader,\_x000B\_ options=options,\_x000B\_ passwords=passwords\_x000B\_ )\_x000B\_ result = playbook.run()\_x000B\_ return result\_x000B\_\_x000B\_if \_\_name\_\_ == '\_\_main\_\_':\_x000B\_ runpb(pb\_path=['/root/ansible/yum.yml']) #传入实现写好的yml文件绝对路径,\_x000B\_ \_x000B\_'''\_x000B\_此程序作用是定义主机hosts文件,定义yml文件(playbook文件),即可使用本程序来调用playbook来实现ansible管理**

**\_x000B\_ 'syntax',\_x000B\_ 'sudo\_user',\_x000B\_ 'sudo',\_x000B\_ 'diff'\_x000B\_ ]\_x000B\_)\_x000B\_options = Options(\_x000B\_ connection='smart',\_x000B\_ remote\_user=None,\_x000B\_ ask\_pass=None,\_x000B\_ sudo\_user=None,\_x000B\_ forks=5,\_x000B\_ sudo=None,\_x000B\_ ask\_sudo\_pass=False,\_x000B\_ verbosity=5,\_x000B\_ module\_path=None,\_x000B\_ become=None,\_x000B\_ become\_method=None,\_x000B\_ become\_user=None,\_x000B\_ check=False,\_x000B\_ diff=False,\_x000B\_ listhosts=None,\_x000B\_ listtasks=None,\_x000B\_ listtags=None,\_x000B\_ syntax=None\_x000B\_)\_x000B\_loader = DataLoader()\_x000B\_passwords = dict()\_x000B\_inventory = InventoryManager(loader=loader, sources=['/root/ansible/hosts']) \_x000B\_ #定义相关参数,sources是hosts主机群文件\_x000B\_variable\_manager = VariableManager(loader=loader, inventory=inventory)\_x000B\_\_x000B\_def runpb(pb\_path):\_x000B\_ playbook = PlaybookExecutor(\_x000B\_ playbooks=pb\_path,\_x000B\_ inventory=inventory,\_x000B\_ variable\_manager=variable\_manager,\_x000B\_ loader=loader,\_x000B\_ options=options,\_x000B\_ passwords=passwords\_x000B\_ )\_x000B\_ result = playbook.run()\_x000B\_ return result\_x000B\_\_x000B\_if \_\_name\_\_ == '\_\_main\_\_':\_x000B\_ runpb(pb\_path=['/root/ansible/yum.yml']) #传入实现写好的yml文件绝对路径,\_x000B\_ \_x000B\_'''\_x000B\_此程序作用是定义主机hosts文件,定义yml文件(playbook文件),即可使用本程序来调用playbook来实现ansible管理**

**'connection'**,  
 **'remote\_user'**,  
 **'ask\_sudo\_pass'**,  
 **'verbosity'**,  
 **'ask\_pass'**,  
 **'module\_path'**,  
 **'forks'**,  
 **'become'**,  
 **'become\_method'**,  
 **'become\_user'**,  
 **'check'**,  
 **'listhosts'**,  
 **'listtasks'**,  
 **'listtags'**,  
 **'syntax'**,  
 **'sudo\_user'**,  
 **'sudo'**,  
 **'diff'  
 ]  
)** #以上为选项  
options **=** Options**(** connection**='smart'**,  
 remote\_user**=None**,  
 ask\_pass**=None**,  
 sudo\_user**=None**,  
 forks**=**5,  
 sudo**=None**,  
 ask\_sudo\_pass**=False**,  
 verbosity**=**5,  
 module\_path**=None**,  
 become**=None**,  
 become\_method**=None**,  
 become\_user**=None**,  
 check**=False**,  
 diff**=False**,  
 listhosts**=None**,  
 listtasks**=None**,  
 listtags**=None**,  
 syntax**=None  
)** #以上为实例化  
loader **=** DataLoader**()** #分析yuml,ini等文件  
passwords **=** dict**()** #密码认证,免密模式  
inventory **=** InventoryManager**(**loader**=**loader, sources**=['/root/ansible/hosts'])** #InventoryManager定义主机清单参数,sources是hosts主机群文件  
variable\_manager **=** VariableManager**(**loader**=**loader, inventory**=**inventory**)  
  
def runpb(**pb\_path**):** #用于执行playbook函数  
 playbook **=** PlaybookExecutor**(** playbooks**=**pb\_path, #playbook的路径  
 inventory**=**inventory, #上面定义了主机清单  
 variable\_manager**=**variable\_manager, #意思是参数在哪?在上面定义的56行  
 loader**=**loader, #什么方式分析文件,前面已定义 52行  
 options**=**options, #选项,前面已定义  
 passwords**=**passwords  
 **)** #调用PlaybookExecutor用于执行playbook的函数  
 result **=** playbook.run**()  
 return** result  
  
**if** \_\_name\_\_ **== '\_\_main\_\_':** runpb**(**pb\_path**=['/root/ansible/yum.yml'])** #传入实现写好的yml文件绝对路径,  
  
**'''  
此程序作用是定义主机hosts文件,定义yml文件(playbook文件),即可使用本程序来调用playbook来实现ansible管理  
'''**

## 编辑ansible模块(AnsibleModule模块)

### 创建自定义模块路径

### 创建copy模块

from ansible.module\_utils.basic import AnsibleModule

import shutil

def main():

module = AnsibleModule(

argument\_spec=dict(

yuan=diPlaybookExecutorct(required=True, type='str'),

mudi=dict(required=True, type='str')

)

)

shutil.copy(module.params['yuan'], module.params['mudi'])

module.exit\_json(changed=True)

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

main()

### 创建wget模块

from ansible.module\_utils.basic import AnsibleModule

import wget

def main():

module = AnsibleModule(

argument\_spec=dict(

url=dict(required=True, type='str'),

dest=dict(required=True, type='str')

)

)

wget.download(module.params['url'], module.params['dest'])

module.exit\_jsoPlaybookExecutorn(changed=True)

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

main()

## ansible-cmdb:

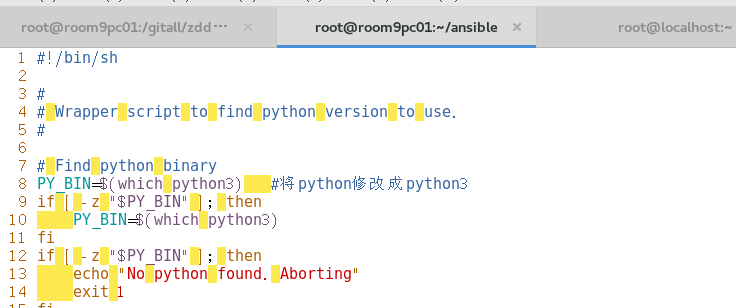
[root@room9pc01 ansible]# ansible all -m setup --tree /root/hostszdd

[root@room9pc01 ansible]# ls /root/hostszdd/

node14 node15 node16

[root@room9pc01 ansible]# pip3 install ansible-cmdb

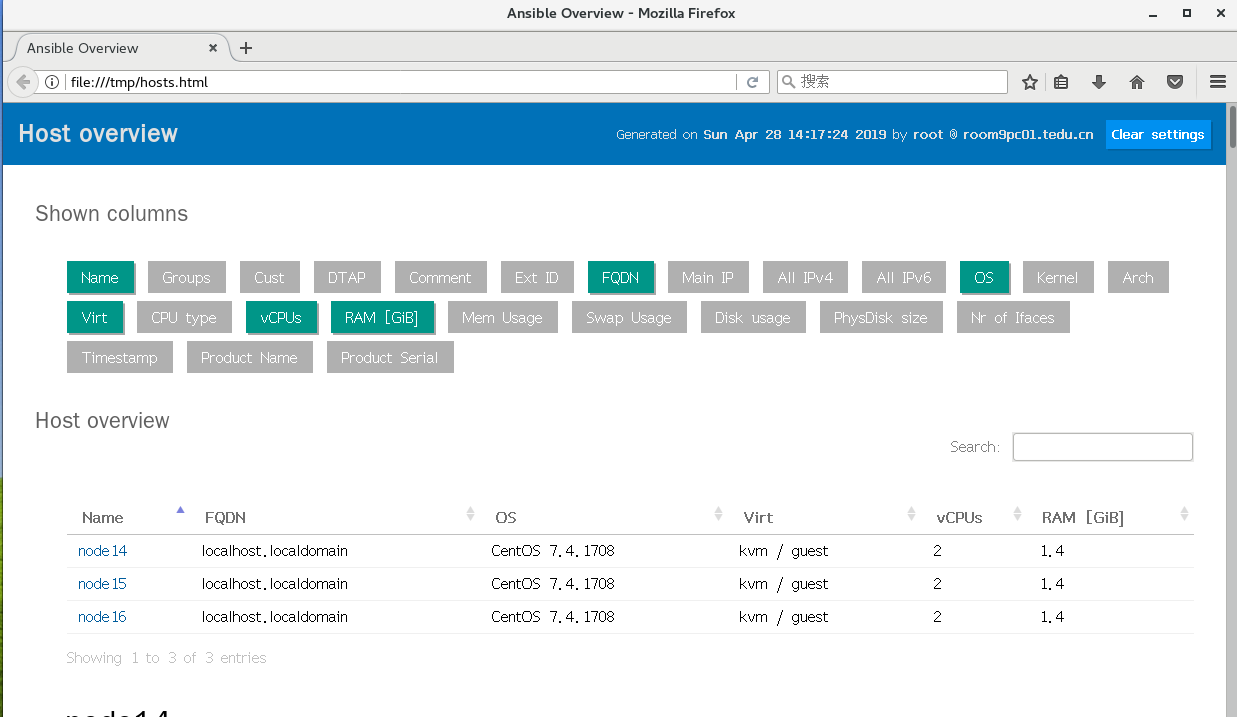
[root@room9pc01 ansible]# vim $(which ansible-cmdb)

PlaybookExecutor

[root@room9pc01 ansible]# ansible-cmdb /root/hostszdd/ >/tmp/hosts.html

[root@room9pc01 ansible]# firefox /tmp/hosts.html

最终效果图:



libraby

# WEB概述

<!--...--> 定义注释。

<!DOCTYPE html> #定义文档类型

<html lang="en">

<head>

<meta charset="UTF-8">

<title>my html</title>

</head>

<body>

<table border="1" width="100">

<tr>

<td rowspan="2">1</td>

<td colspan="2">1</td>

</tr>

<tr>

<td>1</td>

<td>1</td>

</tr>

</table>

<hr>

<table border="1" width="300">

<tr align="center">

<td>姓名</td>

<td>城市</td>

</tr>

<tr>

<td>吕磊</td>

<td>西安</td>

</tr>

<tr>

<td>吕子博</td>

<td>杭州</td>

</tr>

</table>

<hr>

<a href="#id1">hello1</a>

<hr>

<a href="http://www.163.com" target="\_blank">

<img width="300" src="https://img03.sogoucdn.com/app/a/100520021/a728006606fc2a983241998f831bb529">

</a>

<hr>

<a href="http://www.sogou.com" target="\_blank">搜狗搜索</a>

<hr>

<img width="600" height="600" src="https://img03.sogoucdn.com/app/a/100520021/a728006606fc2a983241998f831bb529" alt="如果图片不能显示，则显示此文字">

<hr>

你好。<b>你好</b>你好 <i>你好</i> 你好 <u>你好</u> 你好 <s>你好</s> <br>

2<sup>3</sup>=8 log<sub>2</sub>N

<hr>

hello&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;world! &lt;hr&gt; &copy;

&aacute; &bernou;

<hr>

<p>Hello World!</p>

<p>Hello World!</p>

<div>Hello World!</div>

<div>Hello World!</div>

<h1 id="id1">Hello World!</h1>

<a href="#">返回顶部</a>

<h2>Hello World!</h2>

<h3>Hello World!</h3>

<h4>Hello World!</h4>

<h5>Hello World!</h5>

<h6>Hello World!</h6>

</body>