****

**云计算**

**Python编程120例**

达内 ⬝ 云计算学院

2019年2月

目 录

[01-Hello World 6](#_Toc534039868)

[02-print 6](#_Toc534039869)

[03-基本运算 6](#_Toc534039870)

[04-input 6](#_Toc534039871)

[05-输入输出基础练习 7](#_Toc534039872)

[06-字符串使用基础 7](#_Toc534039873)

[07-列表基础 8](#_Toc534039874)

[08-元组基础 8](#_Toc534039875)

[09-字典基础 8](#_Toc534039876)

[10-基本判断 8](#_Toc534039877)

[11-条件表达式、三元运算符 9](#_Toc534039878)

[12-判断练习：用户名和密码是否正确 9](#_Toc534039879)

[13-成绩分类 9](#_Toc534039880)

[14-石头剪刀布 10](#_Toc534039881)

[15-改进的石头剪刀布 10](#_Toc534039882)

[16-猜数：基础实现 11](#_Toc534039883)

[17-猜数，直到猜对 11](#_Toc534039884)

[18-猜数，5次机会 11](#_Toc534039885)

[19-while循环，累加至100 12](#_Toc534039886)

[20-while-break 12](#_Toc534039887)

[21-while-continue 12](#_Toc534039888)

[22-for循环遍历数据对象 12](#_Toc534039889)

[23-range用法及数字累加 13](#_Toc534039890)

[24-列表实现斐波那契数列 13](#_Toc534039891)

[25-九九乘法表 13](#_Toc534039892)

[26-逐步实现列表解析 14](#_Toc534039893)

[27-三局两胜的石头剪刀布 14](#_Toc534039894)

[28-文件对象基础操作 14](#_Toc534039895)

[29-拷贝文件-1 15](#_Toc534039896)

[30-拷贝文件-2 16](#_Toc534039897)

[31-位置参数 16](#_Toc534039898)

[32-函数应用-斐波那契数列 16](#_Toc534039899)

[33-函数-拷贝文件 17](#_Toc534039900)

[34-函数-九九乘法表 17](#_Toc534039901)

[35-模块基础 17](#_Toc534039902)

[36-生成密码/验证码 18](#_Toc534039903)

[37-序列对象方法 18](#_Toc534039904)

[38-序列对象方法2 18](#_Toc534039905)

[39-字符串方法 19](#_Toc534039906)

[40-字符串格式化 19](#_Toc534039907)

[41-shutil模块常用方法 20](#_Toc534039908)

[42-练习：生成文本文件 20](#_Toc534039909)

[43-列表方法 21](#_Toc534039910)

[44-检查合法标识符 21](#_Toc534039911)

[45-动画程序：@从一行#中穿过 22](#_Toc534039912)

[46-创建用户，设置随机密码 22](#_Toc534039913)

[47-列表练习：模拟栈操作 22](#_Toc534039914)

[48-实现Linux系统中unix2dos功能 23](#_Toc534039915)

[49-字典基础用法 23](#_Toc534039916)

[50-字典常用方法 24](#_Toc534039917)

[51-集合常用方法 24](#_Toc534039918)

[52-集合实例：取出第二个文件有，第一个文件没有的行 25](#_Toc534039919)

[53-字典练习：模拟注册/登陆 25](#_Toc534039920)

[54-计算千万次加法运算时间 26](#_Toc534039921)

[55-时间相关模块常用方法 26](#_Toc534039922)

[56-变量作用域 26](#_Toc534039923)

[57-os模块常用方法 27](#_Toc534039924)

[58-pickle存储器 27](#_Toc534039925)

[59-异常处理基础 28](#_Toc534039926)

[60-异常处理完整语法 28](#_Toc534039927)

[61-自定义异常 28](#_Toc534039928)

[62-函数调用：参数使用注意事项 29](#_Toc534039929)

[63-参数个数不固定的函数 29](#_Toc534039930)

[64-偏函数基础应用 29](#_Toc534039931)

[65-偏函数应用：简单的图形窗口 30](#_Toc534039932)

[66-生成器基础 30](#_Toc534039933)

[67-生成器实例：每次取出文件的10行内容 30](#_Toc534039934)

[68-匿名函数和filter 31](#_Toc534039935)

[69-匿名函数和map 31](#_Toc534039936)

[70-函数练习：数学游戏 31](#_Toc534039937)

[71-数学游戏进阶 32](#_Toc534039938)

[72-递归函数计算阶乘 33](#_Toc534039939)

[73-递归函数练习：逐级列出目录内容 33](#_Toc534039940)

[74-递归函数练习：快速排序 34](#_Toc534039941)

[75-闭包的用法 34](#_Toc534039942)

[76-装饰器基础 35](#_Toc534039943)

[77-带有参数的装饰器 36](#_Toc534039944)

[78-装饰器，返回不同颜色的字体 36](#_Toc534039945)

[79-综合练习：记账小程序 36](#_Toc534039946)

[80-hashlib模块之计算md5值 38](#_Toc534039947)

[81-tarfile模块的基础应用 38](#_Toc534039948)

[82-OOP基础 38](#_Toc534039949)

[83-OOP之组合 39](#_Toc534039950)

[84-OOP之继承 39](#_Toc534039951)

[85-OOP之子类调用父类方法 40](#_Toc534039952)

[87-OOP之必需掌握的magic 40](#_Toc534039953)

[87-OOP之多重继承 41](#_Toc534039954)

[88-OOP之类方法和静态方法 41](#_Toc534039955)

[89-综合练习：备份程序 42](#_Toc534039956)

[90-OOP练习：实现unix2dos和dos2unix功能 43](#_Toc534039957)

[91-re模块基础用法 44](#_Toc534039958)

[92-re练习：匹配文件中指定模式 44](#_Toc534039959)

[93-re练习：模式匹配进阶写法 44](#_Toc534039960)

[94-socket基础：TCP服务器流程 45](#_Toc534039961)

[95-可重用的TCP服务器 45](#_Toc534039962)

[96-简单而完整的TCP服务器 46](#_Toc534039963)

[97-简单的TCP客户端 47](#_Toc534039964)

[98-简单的UDP服务器流程 47](#_Toc534039965)

[99-简单的UDP客户端流程 47](#_Toc534039966)

[100-百鸡百钱问题 48](#_Toc534039967)

[101-多进程基础 48](#_Toc534039968)

[102-多进程的ping 49](#_Toc534039969)

[103-多进程的效率 49](#_Toc534039970)

[104-僵尸进程 50](#_Toc534039971)

[105-解决僵尸进程问题 51](#_Toc534039972)

[106-基于多进程的时间消息服务器 51](#_Toc534039973)

[107-基于多线程的ping 52](#_Toc534039974)

[108-多线程的效率 52](#_Toc534039975)

[109-基于多线程的时间消息服务器 53](#_Toc534039976)

[110-并行批量管理远程服务器 54](#_Toc534039977)

[111-配置IP地址 54](#_Toc534039978)

[112-模拟字符串lstrip用法 55](#_Toc534039979)

[113-模拟字符串rstrip用法 56](#_Toc534039980)

[114-fork子进程解析 56](#_Toc534039981)

[115-可变与不可变对象的效率 57](#_Toc534039982)

[116-ip地址与10进制数的转换 58](#_Toc534039983)

[117-比较文件的差异 58](#_Toc534039984)

[118-打造vim为python IDE 59](#_Toc534039985)

[119-python交互解释器tab补全功能 60](#_Toc534039986)

[120-钉钉机器人 61](#_Toc534039987)

**01-Hello World**

python的语法逻辑完全靠缩进，建议缩进4个空格。

如果是顶级代码，那么必须顶格书写，哪怕只有一个空格也会有语法错误。

下面示例中，满足if条件要输出两行内容，这两行内容必须都缩进，而且具有相同的缩进级别。

print('hello world!')

if 3 > 0:

print('OK')

print('yes')

x = 3; y = 4 # 不推荐，还是应该写成两行

print(x + y)

**02-print**

print('hello world!')

print('hello', 'world!') # 逗号自动添加默认的分隔符：空格

print('hello' + 'world!') # 加号表示字符拼接

print('hello', 'world', sep='\*\*\*') # 单词间用\*\*\*分隔

print('#' \* 50) # \*号表示重复50遍

print('how are you?', end='') # 默认print会打印回车，end=''表示不要回车

**03-基本运算**

运算符可以分为：算术运算符、比较运算符和逻辑运算符。优先级是：算术运算符>比较运算符>逻辑运算符。不过呢，开始没背下来优先级，最好使用括号。这样不用背，也增加了代码的可读性。

print(5 / 2) # 2.5

print(5 // 2) # 丢弃余数，只保留商

print(5 % 2) # 求余数

print(5 \*\* 3) # 5的3次方

print(5 > 3) # 返回True

print(3 > 5) # 返回False

print(20 > 10 > 5) # python支持连续比较

print(20 > 10 and 10 > 5) # 与上面相同含义

print(not 20 > 10) # False

注意：True和False是关键字，区分大小写。

**04-input**

number = input("请输入数字: ") # input用于获取键盘输入

print(number)

print(type(number)) # input获得的数据是字符型

print(number + 10) # 报错，不能把字符和数字做运算

print(int(number) + 10) # int可将字符串10转换成数字10

print(number + str(10)) # str将10转换为字符串后实现字符串拼接

**05-输入输出基础练习**

username = input('username: ')

print('welcome', username) # print各项间默认以空格作为分隔符

print('welcome ' + username) # 注意引号内最后的空格

**06-字符串使用基础**

python中，单双引号没有区别，表示一样的含义。

sentence = 'tom\'s pet is a cat' # 单引号中间还有单引号，可以转义

sentence2 = "tom's pet is a cat" # 也可以用双引号包含单引号

sentence3 = "tom said:\"hello world!\""

sentence4 = 'tom said:"hello world"'

# 三个连续的单引号或双引号，可以保存输入格式，允许输入多行字符串

words = """

hello

world

abcd"""

print(words)

py\_str = 'python'

len(py\_str) # 取长度

py\_str[0] # 第一个字符

'python'[0]

py\_str[-1] # 最后一个字符

# py\_str[6] # 错误，下标超出范围

py\_str[2:4] # 切片，起始下标包含，结束下标不包含

py\_str[2:] # 从下标为2的字符取到结尾

py\_str[:2] # 从开头取到下标是2之前的字符

py\_str[:] # 取全部

py\_str[::2] # 步长值为2，默认是1

py\_str[1::2] # 取出yhn

py\_str[::-1] # 步长为负，表示自右向左取

py\_str + ' is good' # 简单的拼接到一起

py\_str \* 3 # 把字符串重复3遍

't' in py\_str # True

'th' in py\_str # True

'to' in py\_str # False

'to' not in py\_str # True

**07-列表基础**

列表也是序列对象，但它是容器类型，列表中可以包含各种数据

alist = [10, 20, 30, 'bob', 'alice', [1,2,3]]

len(alist)

alist[-1] # 取出最后一项

alist[-1][-1] # 因为最后一项是列表，列表还可以继续取下标

[1,2,3][-1] # [1,2,3]是列表，[-1]表示列表最后一项

alist[-2][2] # 列表倒数第2项是字符串，再取出字符下标为2的字符

alist[3:5] # ['bob', 'alice']

10 in alist # True

'o' in alist # False

100 not in alist # True

alist[-1] = 100 # 修改最后一项的值

alist.append(200) # 向列表中追加一项

**08-元组基础**

元组与列表基本上是一样的，只是元组不可变，列表可变。

atuple = (10, 20, 30, 'bob', 'alice', [1,2,3])

len(atuple)

10 in atuple

atuple[2]

atuple[3:5]

# atuple[-1] = 100 # 错误，元组是不可变的

**09-字典基础**

# 字典是key-value(键－值）对形式的，没有顺序，通过键取出值

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

len(adict)

'bob' in adict # False

'name' in adict # True

adict['email'] = 'bob@tedu.cn' # 字典中没有key，则添加新项目

adict['age'] = 25 # 字典中已有key，修改对应的value

**10-基本判断**

单个的数据也可作为判断条件。

任何值为0的数字、空对象都是False，任何非0数字、非空对象都是True。

if 3 > 0:

print('yes')

print('ok')

if 10 in [10, 20, 30]:

print('ok')

if -0.0:

print('yes') # 任何值为0的数字都是False

if [1, 2]:

print('yes') # 非空对象都是True

if ' ':

print('yes') # 空格字符也是字符，条件为True

**11-条件表达式、三元运算符**

a = 10

b = 20

if a < b:

smaller = a

else:

smaller = b

print(smaller)

s = a if a < b else b # 和上面的if-else语句等价

print(s)

**12-判断练习：用户名和密码是否正确**

import getpass # 导入模块

username = input('username: ')

# getpass模块中，有一个方法也叫getpass

password = getpass.getpass('password: ')

if username == 'bob' and password == '123456':

print('Login successful')

else:

print('Login incorrect')

**13-成绩分类**

score = int(input('分数: '))

if score >= 90:

print('优秀')

elif score >= 80:

print('好')

elif score >= 70:

print('良')

elif score >= 60:

print('及格')

else:

print('你要努力了')

**14-石头剪刀布**

import random

all\_choices = ['石头', '剪刀', '布']

computer = random.choice(all\_choices)

player = input('请出拳: ')

# print('Your choice:', player, "Computer's choice:", computer)

print("Your choice: %s, Computer's choice: %s" % (player, computer))

if player == '石头':

if computer == '石头':

print('平局')

elif computer == '剪刀':

print('You WIN!!!')

else:

print('You LOSE!!!')

elif player == '剪刀':

if computer == '石头':

print('You LOSE!!!')

elif computer == '剪刀':

print('平局')

else:

print('You WIN!!!')

else:

if computer == '石头':

print('You WIN!!!')

elif computer == '剪刀':

print('You LOSE!!!')

else:

print('平局')

**15-改进的石头剪刀布**

import random

all\_choices = ['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): """

computer = random.choice(all\_choices)

ind = int(input(prompt))

player = all\_choices[ind]

print("Your choice: %s, Computer's choice: %s" % (player, computer))

if player == computer:

print('\033[32;1m平局\033[0m')

elif [player, computer] in win\_list:

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

else:

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

**16-猜数：基础实现**

import random

num = random.randint(1, 10) # 随机生成1－10之间的数字

answer = int(input('guess a number: ')) # 将用户输入的字符转成整数

if answer > num:

print('猜大了')

elif answer < num:

print('猜小了')

else:

print('猜对了')

print('the number:', num)

**17-猜数，直到猜对**

import random

num = random.randint(1, 10)

running = True

while running:

answer = int(input('guess the number: '))

if answer > num:

print('猜大了')

elif answer < num:

print('猜小了')

else:

print('猜对了')

running = False

**18-猜数，5次机会**

import random

num = random.randint(1, 10)

counter = 0

while counter < 5:

answer = int(input('guess the number: '))

if answer > num:

print('猜大了')

elif answer < num:

print('猜小了')

else:

print('猜对了')

break

counter += 1

else: # 循环被break就不执行了，没有被break才执行

print('the number is:', num)

**19-while循环，累加至100**

因为循环次数是已知的，实际使用时，建议用for循环

sum100 = 0

counter = 1

while counter < 101:

sum100 += counter

counter += 1

print(sum100)

**20-while-break**

break是结束循环，break之后、循环体内代码不再执行。

while True:

yn = input('Continue(y/n): ')

if yn in ['n', 'N']:

break

print('running...')

**21-while-continue**

计算100以内偶数之和。

continue是跳过本次循环剩余部分，回到循环条件处。

sum100 = 0

counter = 0

while counter < 100:

counter += 1

# if counter % 2:

if counter % 2 == 1:

continue

sum100 += counter

print(sum100)

**22-for循环遍历数据对象**

astr = 'hello'

alist = [10, 20, 30]

atuple = ('bob', 'tom', 'alice')

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

for ch in astr:

print(ch)

for i in alist:

print(i)

for name in atuple:

print(name)

for key in adict:

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

**23-range用法及数字累加**

# range(10) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# >>> list(range(10))

# range(6, 11) # [6, 7, 8, 9, 10]

# range(1, 10, 2) # [1, 3, 5, 7, 9]

# range(10, 0, -1) # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

sum100 = 0

for i in range(1, 101):

sum100 += i

print(sum100)

**24-列表实现斐波那契数列**

列表中先给定两个数字，后面的数字总是前两个数字之和。

fib = [0, 1]

for i in range(8):

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

print(fib)

**25-九九乘法表**

for i in range(1, 10):

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

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

print()

# i=1 ->j: [1]

# i=2 ->j: [1,2]

# i=3 ->j: [1,2,3]

或者，由用户指定相乘到多少：

n = int(input('number: '))

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

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

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

print()

**26-逐步实现列表解析**

# 10+5的结果放到列表中

[10 + 5]

# 10+5这个表达式计算10次

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

# 10+i的i来自于循环

[10 + i for i in range(10)]

[10 + i for i in range(1, 11)]

# 通过if过滤，满足if条件的才参与10+i的运算

[10 + i for i in range(1, 11) if i % 2 == 1]

[10 + i for i in range(1, 11) if i % 2]

# 生成IP地址列表

['192.168.1.%s' % i for i in range(1, 255)]

**27-三局两胜的石头剪刀布**

import random

all\_choices = ['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): """

cwin = 0

pwin = 0

while cwin < 2 and pwin < 2:

computer = random.choice(all\_choices)

ind = int(input(prompt))

player = all\_choices[ind]

print("Your choice: %s, Computer's choice: %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']

**28-文件对象基础操作**

# 文件操作的三个步骤：打开、读写、关闭

# cp /etc/passwd /tmp

f = open('/tmp/passwd') # 默认以r的方式打开纯文本文件

data = f.read() # read()把所有内容读取出来

print(data)

data = f.read() # 随着读写的进行，文件指针向后移动。

# 因为第一个f.read()已经把文件指针移动到结尾了，所以再读就没有数据了

# 所以data是空字符串

f.close()

f = open('/tmp/passwd')

data = f.read(4) # 读4字节

f.readline() # 读到换行符\n结束

f.readlines() # 把每一行数据读出来放到列表中

f.close()

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

f = open('/tmp/passwd')

for line in f:

print(line, end='')

f.close()

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

f = open('图片地址', 'rb') # 打开非文本文件要加参数b

f.read(4096)

f.close()

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

f = open('/tmp/myfile', 'w') # 'w'打开文件，如果文件不存在则创建

f.write('hello world!\n')

f.flush() # 立即将缓存中的数据同步到磁盘

f.writelines(['2nd line.\n', 'new line.\n'])

f.close() # 关闭文件的时候，数据保存到磁盘

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

with open('/tmp/passwd') as f:

print(f.readline())

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

f = open('/tmp/passwd')

f.tell() # 查看文件指针的位置

f.readline()

f.tell()

f.seek(0, 0) # 第一个数字是偏移量，第2位是数字是相对位置。

# 相对位置0表示开头，1表示当前，2表示结尾

f.tell()

f.close()

**29-拷贝文件-1**

拷贝文件就是以r的方式打开源文件，以w的方式打开目标文件，将源文件数据读出后，写到目标文件。

以下是【不推荐】的方式，但是可以工作：

f1 = open('/bin/ls', 'rb')

f2 = open('/root/ls', 'wb')

data = f1.read()

f2.write(data)

f1.close()

f2.close()

**30-拷贝文件-2**

每次读取4K，读完为止：

src\_fname = '/bin/ls'

dst\_fname = '/root/ls'

src\_fobj = open(src\_fname, 'rb')

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

while True:

data = src\_fobj.read(4096)

if not data:

break

dst\_fobj.write(data)

src\_fobj.close()

dst\_fobj.close()

**31-位置参数**

注意：位置参数中的数字是字符形式的。

import sys

print(sys.argv) # sys.argv是sys模块里的argv列表

# python3 position\_args.py

# python3 position\_args.py 10

# python3 position\_args.py 10 bob

**32-函数应用-斐波那契数列**

def gen\_fib(l):

fib = [0, 1]

for i in range(l - len(fib)):

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

return fib # 返回列表，不返回变量fib

a = gen\_fib(10)

print(a)

print('-' \* 50)

n = int(input("length: "))

print(gen\_fib(n)) # 不会把变量n传入，是把n代表的值赋值给形参

**33-函数-拷贝文件**

import sys

def copy(src\_fname, dst\_fname):

src\_fobj = open(src\_fname, 'rb')

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

while True:

data = src\_fobj.read(4096)

if not data:

break

dst\_fobj.write(data)

src\_fobj.close()

dst\_fobj.close()

copy(sys.argv[1], sys.argv[2])

# 执行方式

# cp\_func.py /etc/hosts /tmp/zhuji.txt

**34-函数-九九乘法表**

def mtable(n):

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

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

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

print()

mtable(6)

mtable(9)

**35-模块基础**

每一个以py作为扩展名的文件都是一个模块。

star.py：

hi = 'hello world!'

def pstar(n=50):

print('\*' \* n)

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

pstar()

pstar(30)

在call\_star.py中调用star模块：

import star

print(star.hi)

star.pstar()

star.pstar(30)

**36-生成密码/验证码**

此文件名为：randpass.py。

思路：

1、设置一个用于随机取出字符的基础字符串，本例使用大小写字母加数字

2、循环n次，每次随机取出一个字符

3、将各个字符拼接起来，保存到变量result中

from random import choice

import string

all\_chs = string.ascii\_letters + string.digits # 大小写字母加数字

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))

print(gen\_pass(10))

**37-序列对象方法**

from random import randint

alist = list() # []

list('hello') # ['h', 'e', 'l', 'l', 'o']

list((10, 20, 30)) # [10, 20, 30] 元组转列表

astr = str() # ''

str(10) # '10'

str(['h', 'e', 'l', 'l', 'o']) # 将列表转成字符串

atuple = tuple() # ()

tuple('hello') # ('h', 'e', 'l', 'l', 'o')

num\_list = [randint(1, 100) for i in range(10)]

max(num\_list)

min(num\_list)

**38-序列对象方法2**

alist = [10, 'john']

# list(enumerate(alist)) # [(0, 10), (1, 'john')]

# a, b = 0, 10 # a->0 ->10

for ind in range(len(alist)):

print('%s: %s' % (ind, alist[ind]))

for item in enumerate(alist):

print('%s: %s' % (item[0], item[1]))

for ind, val in enumerate(alist):

print('%s: %s' % (ind, val))

atuple = (96, 97, 40, 75, 58, 34, 69, 29, 66, 90)

sorted(atuple)

sorted('hello')

for i in reversed(atuple):

print(i, end=',')

**39-字符串方法**

py\_str = 'hello world!'

py\_str.capitalize()

py\_str.title()

py\_str.center(50)

py\_str.center(50, '#')

py\_str.ljust(50, '\*')

py\_str.rjust(50, '\*')

py\_str.count('l') # 统计l出现的次数

py\_str.count('lo')

py\_str.endswith('!') # 以!结尾吗？

py\_str.endswith('d!')

py\_str.startswith('a') # 以a开头吗？

py\_str.islower() # 字母都是小写的？其他字符不考虑

py\_str.isupper() # 字母都是大写的？其他字符不考虑

'Hao123'.isdigit() # 所有字符都是数字吗？

'Hao123'.isalnum() # 所有字符都是字母数字？

' hello\t '.strip() # 去除两端空白字符，常用

' hello\t '.lstrip()

' hello\t '.rstrip()

'how are you?'.split()

'hello.tar.gz'.split('.')

'.'.join(['hello', 'tar', 'gz'])

'-'.join(['hello', 'tar', 'gz'])

**40-字符串格式化**

"%s is %s years old" % ('bob', 23) # 常用

"%s is %d years old" % ('bob', 23) # 常用

"%s is %d years old" % ('bob', 23.5) # %d是整数 常用

"%s is %f years old" % ('bob', 23.5)

"%s is %5.2f years old" % ('bob', 23.5) # %5.2f是宽度为5，2位小数

"97 is %c" % 97

"11 is %#o" % 11 # %#o表示有前缀的8进制

"11 is %#x" % 11

"%10s%5s" % ('name', 'age') # %10s表示总宽度为10，右对齐, 常用

"%10s%5s" % ('bob', 25)

"%10s%5s" % ('alice', 23)

"%-10s%-5s" % ('name', 'age') # %-10s表示左对齐, 常用

"%-10s%-5s" % ('bob', 25)

"%10d" % 123

"%010d" % 123

"{} is {} years old".format('bob', 25)

"{1} is {0} years old".format(25, 'bob')

"{:<10}{:<8}".format('name', 'age')

**41-shutil模块常用方法**

import shutil

with open('/etc/passwd', 'rb') as sfobj:

with open('/tmp/mima.txt', 'wb') as dfobj:

shutil.copyfileobj(sfobj, dfobj) # 拷贝文件对象

shutil.copyfile('/etc/passwd', '/tmp/mima2.txt')

shutil.copy('/etc/shadow', '/tmp/') # cp /etc/shadow /tmp/

shutil.copy2('/etc/shadow', '/tmp/') # cp -p /etc/shadow /tmp/

shutil.move('/tmp/mima.txt', '/var/tmp/') # mv /tmp/mima.txt /var/tmp/

shutil.copytree('/etc/security', '/tmp/anquan') # cp -r /etc/security /tmp/anquan

shutil.rmtree('/tmp/anquan') # rm -rf /tmp/anquan

# 将mima2.txt的权限设置成与/etc/shadow一样

shutil.copymode('/etc/shadow', '/tmp/mima2.txt')

# 将mima2.txt的元数据设置成与/etc/shadow一样

# 元数据使用stat /etc/shadow查看

shutil.copystat('/etc/shadow', '/tmp/mima2.txt')

shutil.chown('/tmp/mima2.txt', user='zhangsan', group='zhangsan')

**42-练习：生成文本文件**

import os

def get\_fname():

while True:

fname = input('filename: ')

if not os.path.exists(fname):

break

print('%s already exists. Try again' % fname)

return fname

def get\_content():

content = []

print('输入数据，输入end结束')

while True:

line = input('> ')

if line == 'end':

break

content.append(line)

return content

def wfile(fname, content):

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

fobj.writelines(content)

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

fname = get\_fname()

content = get\_content()

content = ['%s\n' % line for line in content]

wfile(fname, content)

**43-列表方法**

alist = [1, 2, 3, 'bob', 'alice']

alist[0] = 10

alist[1:3] = [20, 30]

alist[2:2] = [22, 24, 26, 28]

alist.append(100)

alist.remove(24) # 删除第一个24

alist.index('bob') # 返回下标

blist = alist.copy() # 相当于blist = alist[:]

alist.insert(1, 15) # 向下标为1的位置插入数字15

alist.pop() # 默认弹出最后一项

alist.pop(2) # 弹出下标为2的项目

alist.pop(alist.index('bob'))

alist.sort()

alist.reverse()

alist.count(20) # 统计20在列表中出现的次数

alist.clear() # 清空

alist.append('new')

alist.extend('new')

alist.extend(['hello', 'world', 'hehe'])

**44-检查合法标识符**

import sys

import keyword

import string

first\_chs = string.ascii\_letters + '\_'

all\_chs = first\_chs + string.digits

def check\_id(idt):

if keyword.iskeyword(idt):

return "%s is keyword" % idt

if idt[0] not in first\_chs:

return "1st invalid"

for ind, ch in enumerate(idt[1:]):

if ch not in all\_chs:

return "char in postion #%s invalid" % (ind + 2)

return "%s is valid" % idt

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

print(check\_id(sys.argv[1])) # python3 checkid.py abc@123

**45-动画程序：@从一行#中穿过**

\r是回车不换行.

import time

length = 19

count = 0

while True:

print('\r%s@%s' % ('#' \* count, '#' \* (length - count)), end='')

try:

time.sleep(0.3)

except KeyboardInterrupt:

print('\nBye-bye')

break

if count == length:

count = 0

count += 1

**46-创建用户，设置随机密码**

randpass模块参见[《37-生成密码/验证码》](<https://www.jianshu.com/p/b81b7ae19cb2>)。

import subprocess

import sys

from randpass import gen\_pass

def adduser(username, password, fname):

data = """user information:

%s: %s

"""

subprocess.call('useradd %s' % username, shell=True)

subprocess.call(

'echo %s | passwd --stdin %s' % (password, username),

shell=True

)

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

fobj.write(data % (username, password))

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

username = sys.argv[1]

password = gen\_pass()

adduser(username, password, '/tmp/user.txt')

# python3 adduser.py john

**47-列表练习：模拟栈操作**

stack = []

def push\_it():

item = input('item to push: ')

stack.append(item)

def pop\_it():

if stack:

print("from stack popped %s" % stack.pop())

def view\_it():

print(stack)

def show\_menu():

cmds = {'0': push\_it, '1': pop\_it, '2': view\_it} # 将函数存入字典

prompt = """(0) push it

(1) pop it

(2) view it

(3) exit

Please input your choice(0/1/2/3): """

while True:

# input()得到字符串，用strip()去除两端空白，再取下标为0的字符

choice = input(prompt).strip()[0]

if choice not in '0123':

print('Invalid input. Try again.')

continue

if choice == '3':

break

cmds[choice]()

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

show\_menu()

**48-实现Linux系统中unix2dos功能**

import sys

def unix2dos(fname):

dst\_fname = fname + '.txt'

with open(fname) as src\_fobj:

with open(dst\_fname, 'w') as dst\_fobj:

for line in src\_fobj:

line = line.rstrip() + '\r\n'

dst\_fobj.write(line)

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

unix2dos(sys.argv[1])

**49-字典基础用法**

adict = dict() # {}

dict(['ab', 'cd'])

bdict = dict([('name', 'bob'),('age', 25)])

{}.fromkeys(['zhangsan', 'lisi', 'wangwu'], 11)

for key in bdict:

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

print("%(name)s: %(age)s" % bdict)

bdict['name'] = 'tom'

bdict['email'] = 'tom@tedu.cn'

del bdict['email']

bdict.pop('age')

bdict.clear()

**50-字典常用方法**

adict = dict([('name', 'bob'),('age', 25)])

len(adict)

hash(10) # 判断给定的数据是不是不可变的，不可变数据才能作为key

adict.keys()

adict.values()

adict.items()

# get方法常用，重要

adict.get('name') # 取出字典中name对应的value，如果没有返回None

print(adict.get('qq')) # None

print(adict.get('qq', 'not found')) # 没有qq，返回指定内容

print(adict.get('age', 'not found'))

adict.update({'phone': '13455667788'})

**51-集合常用方法**

# 集合相当于是无值的字典，所以也用{}表示

myset = set('hello')

len(myset)

for ch in myset:

print(ch)

aset = set('abc')

bset = set('cde')

aset & bset # 交集

aset.intersection(bset) # 交集

aset | bset # 并集

aset.union(bset) # 并集

aset - bset # 差补

aset.difference(bset) # 差补

aset.add('new')

aset.update(['aaa', 'bbb'])

aset.remove('bbb')

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？

**52-集合实例：取出第二个文件有，第一个文件没有的行**

# cp /etc/passwd .

# cp /etc/passwd mima

# vim mima -> 修改，与passwd有些区别

with open('passwd') as fobj:

aset = set(fobj)

with open('mima') as fobj:

bset = set(fobj)

with open('diff.txt', 'w') as fobj:

fobj.writelines(bset - aset)

**53-字典练习：模拟注册/登陆**

import getpass

userdb = {}

def register():

username = input('username: ')

if username in userdb:

print('%s already exists.' % username)

else:

password = input('password: ')

userdb[username] = password

def login():

username = input('username: ')

password = getpass.getpass("password: ")

if userdb.get(username) != password:

print('login failed')

else:

print('login successful')

def show\_menu():

cmds = {'0': register, '1': login}

prompt = """(0) register

(1) login

(2) exit

Please input your choice(0/1/2): """

while True:

choice = input(prompt).strip()[0]

if choice not in '012':

print('Invalid inupt. Try again.')

continue

if choice == '2':

break

cmds[choice]()

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

show\_menu()

**54-计算千万次加法运算时间**

import time

result = 0

start = time.time() # 返回运算前时间戳

for i in range(10000000):

result += i

end = time.time() # 返回运算后时间戳

print(result)

print(end - start)

**55-时间相关模块常用方法**

import time

t = time.localtime() # 返回当前时间的九元组

time.gmtime() # 返回格林威治0时区当前时间的九元组

time.time() # 常用，与1970-1-1 8:00之间的秒数，时间戳

time.mktime(t) # 把九元组时间转成时间戳

time.sleep(1)

time.asctime() # 如果有参数，是九元组形式

time.ctime() # 返回当前时间，参数是时间戳，常用

time.strftime("%Y-%m-%d") # 常用

time.strptime('2018-07-20', "%Y-%m-%d") # 返回九元组时间格式

time.strftime('%H:%M:%S')

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

from datetime import datetime

from datetime import timedelta

datetime.today() # 返回当前时间的datetime对象

datetime.now() # 同上，可以用时区作参数

datetime.strptime('2018/06/30', '%Y/%m/%d') # 返回datetime对象

dt = datetime.today()

datetime.ctime(dt)

datetime.strftime(dt, "%Y%m%d")

days = timedelta(days=90, hours=3) # 常用

dt2 = dt + days

dt2.year

dt2.month

dt2.day

dt2.hour

**56-变量作用域**

x = 10 # 全局变量从定义开始到程序结束，一直可见可用

def foo():

print(x)

foo()

def bar():

x = 20 # 此处的x是局部变量，将全局变量遮盖住，不会影响全局变量的值

print(x)

bar() # x -> 20

print(x) # x -> 10

def aaa():

global x # 在局部引用全局变量

x = 100 # 将全局变量x重新赋值为100

print(x) # x -> 100

aaa()

print(x) # x -> 100

**57-os模块常用方法**

import os

os.getcwd() # 显示当前路径

os.listdir() # ls -a

os.listdir('/tmp') # ls -a /tmp

os.mkdir('/tmp/mydemo') # mkdir /tmp/mydemo

os.chdir('/tmp/mydemo') # cd /tmp/mydemo

os.listdir()

os.mknod('test.txt') # touch test.txt

os.symlink('/etc/hosts', 'zhuji') # ln -s /etc/hosts zhuji

os.path.isfile('test.txt') # 判断test.txt是不是文件

os.path.islink('zhuji') # 判断zhuji是不是软链接

os.path.isdir('/etc')

os.path.exists('/tmp') # 判断是否存在

os.path.basename('/tmp/abc/aaa.txt')

os.path.dirname('/tmp/abc/aaa.txt')

os.path.split('/tmp/abc/aaa.txt')

os.path.join('/home/tom', 'xyz.txt')

os.path.abspath('test.txt') # 返回当前目录test.txt的绝对路径

**58-pickle存储器**

import pickle

"""以前的文件写入，只能写入字符串，如果希望把任意数据对象(数字、列表等)写入文件，

取出来的时候数据类型不变，就用到pickle了

"""

# shop\_list = ["eggs", "apple", "peach"]

# with open('/tmp/shop.data', 'wb') as fobj:

# pickle.dump(shop\_list, fobj)

with open('/tmp/shop.data', 'rb') as fobj:

mylist = pickle.load(fobj)

print(mylist[0], mylist[1], mylist[2])

**59-异常处理基础**

try: # 把有可能发生异常的语句放到try里执行

n = int(input("number: "))

result = 100 / n

print(result)

except ValueError:

print('invalid number')

except ZeroDivisionError:

print('0 not allowed')

except KeyboardInterrupt:

print('Bye-bye')

except EOFError:

print('Bye-bye')

print('Done')

**60-异常处理完整语法**

try:

n = int(input("number: "))

result = 100 / n

except (ValueError, ZeroDivisionError):

print('invalid number')

except (KeyboardInterrupt, EOFError):

print('\nBye-bye')

else:

print(result) # 异常不发生时才执行else子句

finally:

print('Done') # 不管异常是否发生都必须执行的语句

# 常用形式有try-except和try-finally

**61-自定义异常**

def set\_age(name, age):

if not 0 < age < 120:

raise ValueError('年龄超过范围') # 自主决定触发什么样的异常

print("%s is %d years old" % (name, age))

def set\_age2(name, age):

assert 0 < age < 120, '年龄超过范围' # 断言异常

print("%s is %d years old" % (name, age))

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

set\_age('zhangsan', 20)

set\_age2('lisi', 200)

**62-函数调用：参数使用注意事项**

def get\_age(name, age):

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

get\_age('bob', 25) # 参数按顺序传递

get\_age(25, 'bob') # 没有语法错误，但是语义不对

get\_age(age=25, name='bob')

# get\_age() # Error，少参数

# get\_age('bob', 25, 100) # Error，多参数

# get\_age(age=25, 'bob') # 语法错误

# get\_age(25, name='bob') # 错误，参数按顺序传递，name得到多个值

get\_age('bob', age=25)

**63-参数个数不固定的函数**

def func1(\*args): # \*表示args是个元组

print(args)

def func2(\*\*kwargs): # \*\*表示kwargs是个字典

print(kwargs)

def func3(x, y):

print(x \* y)

def func4(name, age):

print("%s is %s years old" % (name, age))

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

func1()

func1(10)

func1(10, 'bob')

func2()

func2(name='bob', age=25)

func3(\*[10, 5]) # 调用的时候，\*表示拆开后面的数据类型

func4(\*\*{'name': 'bob', 'age': 25}) # name='bob', age=25

**64-偏函数基础应用**

偏函数可以理解为，将现有函数的某些参数固定下来，构造成一个新函数。新函数调用就不用写那么多参数了。

from functools import partial

def foo(a, b, c, d, f):

return a + b + c + d + f

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

print(foo(10, 20, 30, 40, 5))

print(foo(10, 20, 30, 40, 25))

print(foo(10, 20, 30, 40, 69))

print(foo(10, 20, 30, 40, 32))

add = partial(foo, a=10, b=20, c=30, d=40)

print(add(f=5)) # foo(10, 20, 30, 40, 5)

print(add(f=8)) # foo(10, 20, 30, 40, 8)

**65-偏函数应用：简单的图形窗口**

import tkinter

from functools import partial

root = tkinter.Tk()

lb = tkinter.Label(text="Hello world!")

b1 = tkinter.Button(root, fg='white', bg='blue', text='Button 1') # 不使用偏函数生成按钮

MyBtn = partial(tkinter.Button, root, fg='white', bg='blue') # 使用偏函数定义MyBtn

b2 = MyBtn(text='Button 2')

b3 = MyBtn(text='quit', command=root.quit)

lb.pack()

b1.pack()

b2.pack()

b3.pack()

root.mainloop()

**66-生成器基础**

生成器也是函数，只是常规函数通过return返回一个值，而生成器可以通过yield返回很多中间结果。

def mygen():

yield 'hello'

a = 10 + 20

yield a

yield [1, 2, 3]

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

m = mygen()

for i in m:

print(i)

for i in m:

print(i) # 无值，因为生成器对象只能用一次

**67-生成器实例：每次取出文件的10行内容**

def blocks(fobj):

block = []

counter = 0

for line in fobj:

block.append(line)

counter += 1

if counter == 10:

yield block # 返回中间结果，下次取值，从这里继续向下执行

block = []

counter = 0

if block: # 文件最后不够10行的部分

yield block

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

fobj = open('/tmp/passwd') # cp /etc/passwd /tmp

for lines in blocks(fobj):

print(lines)

print()

fobj.close()

**68-匿名函数和filter**

from random import randint

def func1(x):

return x % 2

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

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

print(alist)

# filter要求第一个参数是函数，该函数必须返回True或False

# 执行时把alist的每一项作为 func1的参数，返回真留下，否则过滤掉

# filter函数的参数又是函数，称作高阶函数

result = filter(func1, alist) # 不使用匿名函数

print(list(result))

result2 = filter(lambda x: x % 2, alist) # 匿名函数，不使用常规函数

print(list(result2))

**69-匿名函数和map**

from random import randint

def func(x):

return x \* 2 + 1

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

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

print(alist)

# map将第二个参数中的每一项交给func函数进行加工，保留加工后的结果

result = map(func, alist) # 使用常规则函数作为参数

result2 = map(lambda x: x \* 2 + 1, alist) # 使用匿名函数作为参数

print(list(result))

print(list(result2))

**70-函数练习：数学游戏**

随机生成100以内的两个数字，实现随机的加减法。如果是减法，结果不能是负数。

算错三次，给出正确答案。

from random import randint, choice

def add(x, y):

return x + y

def sub(x, y):

return x - y

def exam():

cmds = {'+': add, '-': sub}

nums = [randint(1, 100) for i in range(2)]

nums.sort(reverse=True) # 列表降序排列

op = choice('+-')

result = cmds[op](\*nums)

prompt = "%s %s %s = " % (nums[0], op, nums[1])

tries = 0

while tries < 3:

try:

answer = int(input(prompt))

except: # 简单粗暴地捕获所有异常

continue

if answer == result:

print('Very good!')

break

else:

print('Wrong answer.')

tries += 1

else: # 此得是while的else，全算错才给答案，算对了就不用给出答案了

print('%s%s' % (prompt, result))

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

while True:

exam()

try:

yn = input("Continue(y/n)? ").strip()[0]

except IndexError:

continue

except (KeyboardInterrupt, EOFError):

print()

yn = 'n'

if yn in 'nN':

break

**71-数学游戏进阶**

与前面例子[《71-函数练习：数学游戏》](https://www.jianshu.com/p/35aa85dfbba7)相同，只是加减法函数更换为匿名函数。

from random import randint, choice

def exam():

cmds = {'+': lambda x, y: x + y, '-': lambda x, y: x - y}

nums = [randint(1, 100) for i in range(2)]

nums.sort(reverse=True)

op = choice('+-')

result = cmds[op](\*nums)

prompt = "%s %s %s = " % (nums[0], op, nums[1])

tries = 0

while tries < 3:

try:

answer = int(input(prompt))

except:

continue

if answer == result:

print('Very good!')

break

else:

print('Wrong answer.')

tries += 1

else:

print('%s%s' % (prompt, result))

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

while True:

exam()

try:

yn = input("Continue(y/n)? ").strip()[0]

except IndexError:

continue

except (KeyboardInterrupt, EOFError):

print()

yn = 'n'

if yn in 'nN':

break

**72-递归函数计算阶乘**

递归函数就是在函数内部继续调用自己。

def func(n): # 5

if n == 1:

return n

return n \* func(n - 1)

# 5 \* func(4)

# 5 \* 4 \* func(3)

# 5 \* 4 \* 3 \* func(2)

# 5 \* 4 \* 3 \* 2 \* func(1)

# 5 \* 4 \* 3 \* 2 \* 1

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

print(func(5))

print(func(6))

**73-递归函数练习：逐级列出目录内容**

listdir.py:

import os

import sys

def list\_files(path):

if os.path.isdir(path):

print(path + ':')

content = os.listdir(path)

print(content)

for fname in content:

fname = os.path.join(path, fname)

list\_files(fname)

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

list\_files(sys.argv[1]) # python3 listdir.py /etc

**74-递归函数练习：快速排序**

思路：

1、假设列表中第一个数是中间值，比它小的数字放到smaller列表中，比它的大的数字放到larger列表中。再将这三项拼接起来。

2、因为smaller和larger仍然是无序列表，需要使用相同的方法继续分割。

3、如果列表的长度是0或1，那么就没有必要再排序了。

from random import randint

def quick\_sort(num\_list):

if len(num\_list) < 2:

return num\_list

middle = num\_list[0]

smaller = []

larger = []

for i in num\_list[1:]:

if i < middle:

smaller.append(i)

else:

larger.append(i)

return quick\_sort(smaller) + [middle] + quick\_sort(larger)

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

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

print(alist)

print(quick\_sort(alist))

**75-闭包的用法**

下面的代码用到了[《66-偏函数应用：简单的图形窗口》](https://www.jianshu.com/p/c5082104b16b)

图形窗口上的按钮有个command选项，其实它就是一个函数。如下：

import tkinter

from functools import partial

def hello():

lb.config(text="Hello China!")

def welcome():

lb.config(text="Hello Tedu!")

root = tkinter.Tk()

lb = tkinter.Label(text="Hello world!", font="Times 26")

MyBtn = partial(tkinter.Button, root, fg='white', bg='blue')

b1 = MyBtn(text='Button 1', command=hello)

b2 = MyBtn(text='Button 2', command=welcome)

b3 = MyBtn(text='quit', command=root.quit)

lb.pack()

b1.pack()

b2.pack()

b3.pack()

root.mainloop()

按下Button 1和Button 2就会执行hello和welcome两个函数。这两个函数非常类似，如果有10个按钮，并且都是类似的呢？

换成内部函数、闭包的的语法如下：

import tkinter

from functools import partial

def hello(word):

def welcome():

lb.config(text="Hello %s!" % word)

return welcome # hello函数的返回值还是函数

root = tkinter.Tk()

lb = tkinter.Label(text="Hello world!", font="Times 26")

MyBtn = partial(tkinter.Button, root, fg='white', bg='blue')

b1 = MyBtn(text='Button 1', command=hello('China'))

b2 = MyBtn(text='Button 2', command=hello('Tedu'))

b3 = MyBtn(text='quit', command=root.quit)

lb.pack()

b1.pack()

b2.pack()

b3.pack()

root.mainloop()

**76-装饰器基础**

def color(func):

def red():

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

return red

def hello():

return 'Hello World!'

@color

def welcome():

return 'Hello China!'

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

hello = color(hello) # 此种写法可以换成为welcome加上@color的写法

print(hello())

print(welcome()) # welcome因为有装饰器，所以调用时不是调用welcome函数

# 而是相当于color(welcome)()

# color(welcome)返回red，color(welcome)()

# 等价于red()

**77-带有参数的装饰器**

def color(func):

def red(\*args):

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

return red

@color

def hello(word):

return 'Hello %s!' % word

@color

def welcome():

return 'How are you?'

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

print(hello('China'))

print(welcome())

**78-装饰器，返回不同颜色的字体**

def colors(c):

def set\_color(func):

def red(\*word):

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

def green(\*word):

return '\033[32;1m%s\033[0m' % func(\*word)

adict = {'red': red, 'green': green}

return adict[c]

return set\_color

@colors('red')

def hello():

return 'Hello world!'

@colors('green')

def welcome(word):

return 'Hello %s' % word

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

print(hello()) # -> hello = set\_color(hello)

print(welcome('China'))

**79-综合练习：记账小程序**

1、记账时手头有一万块钱

2、可以记录花的钱、存的钱，以及收支明细

import pickle

import os

import time

def cost(wallet, record): # 记录花钱的函数

amount = int(input('amount: '))

comment = input('comment: ')

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

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

balance = pickle.load(fobj) - amount

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

pickle.dump(balance, fobj)

with open(record, 'a') as fobj:

fobj.write(

'%-12s%-8s%-8s%-10s%-20s\n' % (date, amount, '', balance, comment)

)

def save(wallet, record): # 记录存钱的函数

amount = int(input('amount: '))

comment = input('comment: ')

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

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

balance = pickle.load(fobj) + amount

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

pickle.dump(balance, fobj)

with open(record, 'a') as fobj:

fobj.write(

'%-12s%-8s%-8s%-10s%-20s\n' % (date, '', amount, balance, comment)

)

def query(wallet, record): # 查询收支明细的函数

print('%-12s%-8s%-8s%-10s%-20s' % ('date', 'cost', 'save', 'balace', 'comment'))

with open(record) as fobj:

for line in fobj:

print(line, end='')

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

balance = pickle.load(fobj)

print("Latest Balance: %d" % balance)

def show\_menu():

cmds = {'0': cost, '1': save, '2': query}

prompt = """(0) cost

(1) save

(2) query

(3) exit

Please input your choice(0/1/2/3): """

wallet = 'wallet.data'

record = 'record.txt'

if not os.path.exists(wallet):

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

pickle.dump(10000, fobj)

while True:

try:

choice = input(prompt).strip()[0]

except IndexError:

continue

except (KeyboardInterrupt, EOFError):

print()

choice = '3'

if choice not in '0123':

print('Invalid input. Try again.')

continue

if choice == '3':

break

cmds[choice](wallet, record)

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

show\_menu()

**80-hashlib模块之计算md5值**

check\_md5.py

import hashlib

import sys

def check\_md5(fname):

m = hashlib.md5()

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

while True:

data = fobj.read(4096)

if not data:

break

m.update(data)

return m.hexdigest()

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

print(check\_md5(sys.argv[1])) # python3 check\_md5.py /etc/passwd

**81-tarfile模块的基础应用**

import tarfile

# 压缩文件的方法

tar = tarfile.open('/tmp/demo.tar.gz', 'w:gz') # gzip压缩

tar.add('/etc/hosts')

tar.add('/etc/security')

tar.close()

# tar tvzf /tmp/demo.tar.gz

# 解压文件的方法

tar = tarfile.open('/tmp/demo.tar.gz', 'r:gz')

tar.extractall() # 解压所有文件到当前目录

tar.close()

**82-OOP基础**

为玩具厂创建一个玩具熊类。

玩具熊有名字、尺寸、颜色这些数据属性；还有唱歌、说话的行为。

class BearToy:

def \_\_init\_\_(self, nm, color, size):

"""\_\_init\_\_在实例化时自动执行，实例本身自动作为第一个参数传递给self

self只是习惯用的名字，不是必须使用

"""

self.name = nm

self.color = color # 绑定属性到实例

self.size = size

def sing(self):

print('lalala...')

def speak(self):

print('My name is %s' % self.name)

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

tidy = BearToy('Tidy', 'White', 'Large') # 调用\_\_init\_\_

print(tidy.color)

print(tidy.size)

tidy.sing()

tidy.speak()

**83-OOP之组合**

如果两个类有本质不同，其中一类的对象是另一个类对象的组件时，使用组合是最佳方案。

玩具熊还有生产厂商的信息，生产厂商的信息可以作为玩具熊的一个属性。

class Vendor:

def \_\_init\_\_(self, phone, email):

self.phone = phone

self.email = email

def call(self):

print('calling %s' % self.phone)

class BearToy:

def \_\_init\_\_(self, color, size, phone, email):

self.color = color # 绑定属性到实例

self.size = size

self.vendor = Vendor(phone, email)

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

bigbear = BearToy('Brown', 'Middle', '400-111-8989', 'sales@tedu.cn')

print(bigbear.color)

bigbear.vendor.call()

**84-OOP之继承**

如果两个类有很多相同之处，使用继承更为合理。

新品玩具熊增加了一个跑的行为，其他与原来的玩具熊一致。

class BearToy:

def \_\_init\_\_(self, nm, color, size):

self.name = nm

self.color = color # 绑定属性到实例

self.size = size

def sing(self):

print('lalala...')

def speak(self):

print('My name is %s' % self.name)

class NewBear(BearToy):

def run(self):

print('running...')

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

b1 = NewBear('venie', 'Brown', 'Small')

b1.sing()

b1.run()

**85-OOP之子类调用父类方法**

如果子类和父类具有同名的方法，那么父类方法将被遮盖住。

可以在子类中明确指明调用的是父类方法，而不是子类的同名方法。

class BearToy:

def \_\_init\_\_(self, nm, color, size):

self.name = nm

self.color = color # 绑定属性到实例

self.size = size

def sing(self):

print('lalala...')

def speak(self):

print('My name is %s' % self.name)

class NewBear(BearToy):

def \_\_init\_\_(self, nm, color, size, date):

# BearToy.\_\_init\_\_(self, nm, color, size) # 以下写法完全一样，更推荐下面写法

super(NewBear, self).\_\_init\_\_(nm, color, size)

self.date = date # 新品玩具熊增加玩具熊的生产日期

def run(self):

print('running...')

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

b1 = NewBear('venie', 'Brown', 'Small', '2018-07-20')

b1.sing()

b1.run()

**87-OOP之必需掌握的magic**

class Book:

def \_\_init\_\_(self, title, author, pages):

self.title = title

self.author = author

self.pages = pages

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

return '《%s》' % self.title

def \_\_call\_\_(self):

print('《%s》is written by %s' % (self.title, self.author))

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

py\_book = Book('Core Python', 'Wesley', 800) # 调用\_\_init\_\_()方法

print(py\_book) # 调用\_\_str\_\_

py\_book() # 调用\_\_call\_\_

**87-OOP之多重继承**

类的父类（基类）可以有很多个，子类可以调用所有父类的方法。

如果有重名方法，生效的顺序是自下而上，自左而右。当然最好不要出现重名方法。

class A:

def foo(self):

print('in A foo')

def hello(self):

print('A hello')

class B:

def bar(self):

print('in B bar')

def hello(self):

print('B hello')

class C(B, A):

pass

# def hello(self):

# print('C hello')

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

c = C()

c.foo()

c.bar()

c.hello()

**88-OOP之类方法和静态方法**

通过Date创建实例，也可以通过Date.create创建实例

class Date:

def \_\_init\_\_(self, year, month, date):

self.year = year

self.month = month

self.date = date

@classmethod # 类方法，不用创建实例即可调用

def create(cls, dstr): # cls表示类本身, class的缩写

y, m, d = map(int, dstr.split('-')) # map(int, ['2000', '5', '4'])

dt = cls(y, m, d) # 即Date(y, m, d)

return dt

@staticmethod # 静态方法，写在类的外面，可以独立成为一个函数，“愣”把它放到类中了

def is\_date\_valid(dstr):

y, m, d = map(int, dstr.split('-'))

return 1 <= d <= 31 and 1 <= m <=12 and y < 4000

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

bith\_date = Date(1995, 12, 3)

print(Date.is\_date\_valid('2000-5-4'))

day = Date.create('2000-5-4')

print(day)

**89-综合练习：备份程序**

1、既要可以实现完全备份，又要实现增量备份

2、完全备份时，将目录打个tar包，计算每个文件的md5值

3、增量备份时，备份有变化的文件和新增加的文件，更新md5值

import time

import os

import tarfile

import hashlib

import pickle

def check\_md5(fname):

m = hashlib.md5()

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

while True:

data = fobj.read(4096)

if not data:

break

m.update(data)

return m.hexdigest()

def full\_backup(src\_dir, dst\_dir, md5file):

fname = os.path.basename(src\_dir.rstrip('/'))

fname = '%s\_full\_%s.tar.gz' % (fname, time.strftime('%Y%m%d'))

fname = os.path.join(dst\_dir, fname)

md5dict = {}

tar = tarfile.open(fname, 'w:gz')

tar.add(src\_dir)

tar.close()

for path, folders, files in os.walk(src\_dir):

for each\_file in files:

key = os.path.join(path, each\_file)

md5dict[key] = check\_md5(key)

with open(md5file, 'wb') as fobj:

pickle.dump(md5dict, fobj)

def incr\_backup(src\_dir, dst\_dir, md5file):

fname = os.path.basename(src\_dir.rstrip('/'))

fname = '%s\_incr\_%s.tar.gz' % (fname, time.strftime('%Y%m%d'))

fname = os.path.join(dst\_dir, fname)

md5dict = {}

with open(md5file, 'rb') as fobj:

oldmd5 = pickle.load(fobj)

for path, folders, files in os.walk(src\_dir):

for each\_file in files:

key = os.path.join(path, each\_file)

md5dict[key] = check\_md5(key)

with open(md5file, 'wb') as fobj:

pickle.dump(md5dict, fobj)

tar = tarfile.open(fname, 'w:gz')

for key in md5dict:

if oldmd5.get(key) != md5dict[key]:

tar.add(key)

tar.close()

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

# mkdir /tmp/demo; cp -r /etc/security /tmp/demo

src\_dir = '/tmp/demo/security'

dst\_dir = '/var/tmp/backup' # mkdir /var/tmp/backup

md5file = '/var/tmp/backup/md5.data'

if time.strftime('%a') == 'Mon':

full\_backup(src\_dir, dst\_dir, md5file)

else:

incr\_backup(src\_dir, dst\_dir, md5file)

**90-OOP练习：实现unix2dos和dos2unix功能**

windows文本行结束标志是\r\n，非windows的是\n。

import os

class Convert:

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

self.fname = fname

def to\_linux(self):

dst\_fname = os.path.splitext(self.fname)[0] + '.linux'

with open(self.fname, 'r') as src\_fobj:

with open(dst\_fname, 'w') as dst\_fobj:

for line in src\_fobj:

line = line.rstrip() + '\n'

dst\_fobj.write(line)

def to\_windows(self):

dst\_fname = os.path.splitext(self.fname)[0] + '.windows'

with open(self.fname, 'r') as src\_fobj:

with open(dst\_fname, 'w') as dst\_fobj:

for line in src\_fobj:

line = line.rstrip() + '\r\n'

dst\_fobj.write(line)

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

c = Convert('/tmp/passwd') # cp /etc/passwd /tmp

c.to\_linux()

c.to\_windows()

**91-re模块基础用法**

import re

m = re.match('f..', 'food') # 匹配到返回对象

print(re.match('f..', 'seafood')) # 匹配不到返回None

m.group() # 返回匹配的值

m = re.search('f..', 'seafood')

m.group()

re.findall('f..', 'seafood is food') # 返回所有匹配项组成的列表

result = re.finditer('f..', 'seafood is food') # 返回匹配对象组成的迭代器

for m in result: # 从迭代器中逐个取出匹配对象

print(m.group())

re.sub('f..', 'abc', 'fish is food')

re.split('\.|-', 'hello-world.tar.gz') # 用.和-做切割符号

patt = re.compile('f..') # 先把要匹配的模式编译，提升效率

m = patt.search('seafood') # 指定在哪个字符串中匹配

m.group()

**92-re练习：匹配文件中指定模式**

import re

def count\_patt(fname, patt):

cpatt = re.compile(patt)

result = {}

with open(fname) as fobj:

for line in fobj:

m = cpatt.search(line) # 如果匹配不到，返回None

if m:

key = m.group()

result[key] = result.get(key, 0) + 1

return result

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

fname = 'access\_log' # apache日志文件

ip = '^(\d+\.){3}\d+' # 日志开头的ip地址

print(count\_patt(fname, ip))

br = 'Firefox|MSIE|Chrome' # 日志中客户端浏览器

print(count\_patt(fname, br))

**93-re练习：模式匹配进阶写法**

import re

from collections import Counter # Counter对象是有序的，字典无序

class CountPatt:

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

self.fname = fname

def count\_patt(self, patt):

cpatt = re.compile(patt)

result = Counter()

with open(self.fname) as fobj:

for line in fobj:

m = cpatt.search(line) # 如果匹配不到，返回None

if m:

result.update([m.group()])

return result

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

c = CountPatt('access\_log')

ip = '^(\d+\.){3}\d+'

br = 'Firefox|MSIE|Chrome'

a = c.count\_patt(ip)

print(a)

print(a.most\_common(3)) # 访问量最大的前三名

print(c.count\_patt(br))

**94-socket基础：TCP服务器流程**

服务器启动后，测试可以使用：telnet 127.0.0.1 12345

import socket

host = '' # 表示本机所有地址 0.0.0.0

port = 12345 # 应该大于1024

addr = (host, port)

s = socket.socket() # 默认值就是基于TCP的网络套接字

# 设置选项，程序结束之后可以立即再运行，否则要等60秒

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

s.bind(addr) # 绑定地址到套接字

s.listen(1) # 启动侦听进程

cli\_sock, cli\_addr = s.accept() # 等待客户端连接

print('Client connect from:', cli\_addr)

print(cli\_sock.recv(1024)) # 一次最多读1024字节数据

cli\_sock.send(b'I 4 C U\r\n') # 发送的数据要求是bytes类型

cli\_sock.close()

s.close()

**95-可重用的TCP服务器**

在[95-socket基础：TCP服务器流程](https://www.jianshu.com/p/b59008562c2a)中，TCP服务器只能一个客户端连接，客户端也只能发送一条消息。本例允许客户端发送多条消息，输入end结束。客户端退出后，服务器程序不再退出，可以为下一个客户端提供服务：

import socket

host = ''

port = 12345

addr = (host, port)

s = socket.socket()

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

s.bind(addr)

s.listen(1)

while True:

cli\_sock, cli\_addr = s.accept()

print('Client connect from:', cli\_addr)

while True:

data = cli\_sock.recv(1024)

if data.strip() == b'end':

break

print(data.decode('utf8')) # bytes类型转为string类型

data = input('> ') + '\r\n' # 获得的是string类型

cli\_sock.send(data.encode('utf8')) # 转成bytes类型发送

cli\_sock.close()

s.close()

**96-简单而完整的TCP服务器**

客户端可以通过telnet 127.0.0.1 12345来访问。

每发送一段文字，将会收到加上当前时间的文字。

import socket

from time import strftime

class TcpTimeServer:

def \_\_init\_\_(self, host='', port=12345):

self.addr = (host, port)

self.serv = socket.socket()

self.serv.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

self.serv.bind(self.addr)

self.serv.listen(1)

def chat(self, c\_sock):

while True:

data = c\_sock.recv(1024)

if data.strip() == b'quit':

break

data = '[%s] %s' % (strftime('%H:%M:%S'), data.decode('utf8'))

c\_sock.send(data.encode('utf8'))

c\_sock.close()

def mainloop(self):

while True:

cli\_sock, cli\_addr = self.serv.accept()

self.chat(cli\_sock)

self.serv.close()

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

s = TcpTimeServer()

s.mainloop()

**97-简单的TCP客户端**

客户端连接服务器的12345端口，在单独的一行输入end结束客户端程序。

import socket

host = '192.168.4.254' # 服务器IP地址

port = 12345 # 服务器端口

addr = (host, port)

c = socket.socket()

c.connect(addr)

while True:

data = input('> ') + '\r\n'

c.send(data.encode('utf8')) # 服务器收到end结束，所以要先发送再判断

if data.strip() == 'end':

break

data = c.recv(1024)

print(data.decode('utf8'))

c.close()

**98-简单的UDP服务器流程**

UDP是非面向连接的，不用listen、不用accept。

UDP不区分客户端，就算是同一客户端发来的多个数据包，udp服务器也不区分，与处理多个客户端发来的数据包等同对待。

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:

data, cli\_addr = s.recvfrom(1024)

clock = strftime('%H:%M:%S')

data = data.decode('utf8')

data = '[%s] %s' % (clock, data)

s.sendto(data.encode('utf8'), cli\_addr)

s.close()

**99-简单的UDP客户端流程**

UDP客户端非常简单，只要把数据发送到服务器地址就可以了。

import socket

host = '192.168.4.254'

port = 12345

addr = (host, port)

c = socket.socket(type=socket.SOCK\_DGRAM)

while True:

data = input('> ')

if data.strip() == 'quit':

break

c.sendto(data.encode('utf8'), addr)

print(c.recvfrom(1024)[0].decode('utf8'))

# print(c.recvfrom(1024))

c.close()

**100-百鸡百钱问题**

我国古代数学家张丘建在《算经》一书中提出的数学问题：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

思路：

1、答案不只一个

2、如果全是公鸡i，最多100/5只

3、如果全是母鸡j，最多100/3只

4、如果全是小鸡k，100块钱，可以买300只；但，所有的鸡最多是100只

5、鸡的数目i+j+k==100

6、鸡的价钱i \* 5 + j \* 3 + k / 3 == 100

for i in range(100//5 + 1): # //表示只留商，不要小数，舍弃余数

for j in range(100//3 + 1):

for k in range(100):

if i + j + k == 100 and i \* 5 + j \* 3 + k // 3 == 100:

print('公鸡:%s, 母鸡:%s,小鸡:%s' % (i, j, k))

**101-多进程基础**

fork()后会出现子进程，父子进程都打印Hello World!，所以会有两行相同的内容输出。

import os

print('starting...')

os.fork() # 生成子进程，后续代码同时在父子进程中执行

print('Hello World!')

可以根据fork()返回值判断是父进程，还是子进程

import os

print('starting...')

pid = os.fork() # 返回值是个数字，对于父进程，返回值是子进程PID，子进程是0

if pid:

print('In parent') # 父进程执行的代码

else:

print('In child') # 子进程执行的代码

print('Done') # 父子进程都会执行的代码

多进程编程时，要明确父子进程的工作。如：父进程只用于fork子进程；子进程做具体的工作，如果在循环结构中，做完后要退出，否则子进程还会再产生子进程、孙进程……子子孙孙无穷匮也，系统崩溃。

import os

for i in range(5):

pid = os.fork() # 父进程的工作是生成子进程

if not pid: # 如果是子进程，工作完后，结束，不要进入循环

print('hello')

exit() # 注释这一行执行，查看结果，分析原因

**102-多进程的ping**

没有多进程，ping一个网段的IP地址往往要花费几十分钟；使用多进程，几秒钟解决。

import subprocess

import os

def ping(host):

rc = subprocess.call(

'ping -c2 %s &> /dev/null' % host,

shell=True

)

if rc:

print('%s: down' % host)

else:

print('%s: up' % host)

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

ips = ('192.168.1.%s' % i for i in range(1, 255))

for ip in ips:

pid = os.fork()

if not pid:

ping(ip)

exit()

**103-多进程的效率**

没有多进程，即使CPU有多个核心，程序只是运行在一个核心上，无法利用多进程提升效率。5000万次加法，如果需要2.5秒，调用两次共花费5秒。

import time

def calc():

result = 0

for i in range(1, 50000001):

result += i

print(result)

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

start = time.time()

calc()

calc()

end = time.time()

print(end - start)

通过多进程，程序运行在多个核心上，同样的调用两次5000万次加法运算，时间仅为一半。

import time

import os

def calc():

result = 0

for i in range(1, 50000001):

result += i

print(result)

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

start = time.time()

for i in range(2):

pid = os.fork()

if not pid:

calc()

exit()

os.waitpid(-1, 0) # 挂起父进程，直到子进程结束才继续向下执行

os.waitpid(-1, 0) # 每个waitpid只能处理一个僵尸进程，两个子进程需要调用两次

end = time.time()

print(end - start)

**104-僵尸进程**

多进程编程要注意僵尸进程。子进程没有可执行代码后将变成僵尸进程，如果父进程一直运行，又没有处理僵尸进程的代码，僵尸进程也将一直存在，消耗资源。僵尸进程无法通过kill命令杀掉。

import os

import time

pid = os.fork()

if pid:

print('In parent. sleeping...')

time.sleep(60)

print('parent done.')

else:

print('in child. sleeping...')

time.sleep(10)

print('child done') # 10秒后，子进程变成了僵尸进程

# watch -n1 ps a 当子进程成为僵尸进程时，显示为Z

# kill 试图杀死僵尸进程、父进进程，查看结果

**105-解决僵尸进程问题**

os.waitpid()的第2个参数，0表示挂起父进程，1表示不挂起父进程。

import os

import time

pid = os.fork()

if pid:

print('In parent. sleeping...')

print(os.waitpid(-1, 1)) # 无僵尸进程可以处理，返回0

time.sleep(20)

print(os.waitpid(-1, 1)) # 处理僵尸进程，返回子进程PIP

time.sleep(60)

print('parent done.')

else:

print('in child. sleeping...')

time.sleep(10)

print('child done')

# watch -n1 ps a 当子进程成为僵尸进程时，显示为Z

# kill 试图杀死僵尸进程、父进进程，查看结果

**106-基于多进程的时间消息服务器**

1、支持多客户端同时访问

2、客户端向服务器发送消息后，服务器把消息加上时间发回客户端

3、每个客户端断开后会产生僵尸进程，新客户端连接时销毁所有的僵尸进程

import socket

import os

from time import strftime

class TcpTimeServer:

def \_\_init\_\_(self, host='', port=12345):

self.addr = (host, port)

self.serv = socket.socket()

self.serv.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

self.serv.bind(self.addr)

self.serv.listen(1)

def chat(self, c\_sock):

while True:

data = c\_sock.recv(1024)

if data.strip() == b'quit':

break

data = '[%s] %s' % (strftime('%H:%M:%S'), data.decode('utf8'))

c\_sock.send(data.encode('utf8'))

c\_sock.close()

def mainloop(self):

while True:

cli\_sock, cli\_addr = self.serv.accept()

pid = os.fork()

if pid:

cli\_sock.close()

while True:

result = os.waitpid(-1, 1)[0] # waitpid会优先处理僵尸进程

if result == 0:

break

else:

self.serv.close()

self.chat(cli\_sock)

exit()

self.serv.close()

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

s = TcpTimeServer()

s.mainloop()

**107-基于多线程的ping**

多线程与多进程类似，但是每个线程没有自己的资源空间，它们共用进程的资源。

多线程没有僵尸进程的问题。

import subprocess

import threading

def ping(host):

rc = subprocess.call(

'ping -c2 %s &> /dev/null' % host,

shell=True

)

if rc:

print('%s: down' % host)

else:

print('%s: up' % host)

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

ips = ['172.40.58.%s' % i for i in range(1, 255)]

for ip in ips:

# 创建线程，ping是上面定义的函数, args是传给ping函数的参数

t = threading.Thread(target=ping, args=(ip,))

t.start() # 执行ping(ip)

**108-多线程的效率**

python的多线程有一个GIL（全局解释器锁），使得多个线程，某一时刻只有一个线程发送给CPU处理。所以多线程不适用计算密集型应用，更适合IO密集型应用。

以下两次计算5000万次加法运算和不用多线程相比，没有效率的提升。因为CPU有上下文切换，甚至可能多线程更慢。

import time

import threading

def calc():

result = 0

for i in range(1, 50000001):

result += i

print(result)

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

start = time.time()

t1 = threading.Thread(target=calc)

t1.start()

t2 = threading.Thread(target=calc)

t2.start()

t1.join() # 挂起主进程，当t1线程执行完后才继续向下执行

t2.join()

end = time.time()

print(end - start)

**109-基于多线程的时间消息服务器**

与[106-基于多进程的时间消息服务器](https://www.jianshu.com/p/0c45df7ec53a)类似，只是换成了多线程。

import socket

import threading

from time import strftime

class TcpTimeServer:

def \_\_init\_\_(self, host='', port=12345):

self.addr = (host, port)

self.serv = socket.socket()

self.serv.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

self.serv.bind(self.addr)

self.serv.listen(1)

def chat(self, c\_sock):

while True:

data = c\_sock.recv(1024)

if data.strip() == b'quit':

break

data = '[%s] %s' % (strftime('%H:%M:%S'), data.decode('utf8'))

c\_sock.send(data.encode('utf8'))

c\_sock.close()

def mainloop(self):

while True:

cli\_sock, cli\_addr = self.serv.accept()

t = threading.Thread(target=self.chat, args=(cli\_sock,))

t.start()

self.serv.close()

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

s = TcpTimeServer()

s.mainloop()

**110-并行批量管理远程服务器**

脚本名为remote\_comm.py，执行方式如下：

python3 remote\_comm.py 服务器IP地址文件 "在远程服务器上要执行的命令"

如：

# python3 remote\_comm.py serverips.txt "useradd zhangsan"

远程服务器的密码以交互方式获得。

import sys

import getpass

import paramiko

import threading

import os

def remote\_comm(host, pwd, command):

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(hostname=host, username='root', password=pwd)

stdin, stdout, stderr = ssh.exec\_command(command)

out = stdout.read()

error = stderr.read()

if out:

print('[%s] OUT:\n%s' % (host, out.decode('utf8')))

if error:

print('[%s] ERROR:\n%s' % (host, error.decode('utf8')))

ssh.close()

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

if len(sys.argv) != 3:

print('Usage: %s ipaddr\_file "command"' % sys.argv[0])

exit(1)

if not os.path.isfile(sys.argv[1]):

print('No such file:', sys.argv[1])

exit(2)

fname = sys.argv[1]

command = sys.argv[2]

pwd = getpass.getpass()

with open(fname) as fobj:

ips = [line.strip() for line in fobj]

for ip in ips:

t = threading.Thread(target=remote\_comm, args=(ip, pwd, command))

t.start()

**111-配置IP地址**

RHEL7主机有四块网卡，名为eth0/eth1/eth2/eth3。为四块网卡配置IP地址。

#!/usr/bin/env python

import sys

import re

def configip(fname, ip\_addr, if\_ind):

content = """TYPE=Ethernet

BOOTPROTO=none

NAME=eth%s

DEVICE=eth%s

ONBOOT=yes

IPADDR=%s

PREFIX=24

""" % (if\_ind, if\_ind, ip\_addr)

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

fobj.write(content)

def check\_ip(ip\_addr): # 判断IP地址是不是X.X.X.X格式

m = re.match(r'(\d{1,3}\.){3}\d{1,3}$', ip\_addr)

if not m:

return False

return True

def show\_menu():

prompt = """Configure IP Address:

(0) eth0

(1) eth1

(2) eth2

(3) eth3

Your choice(0/1/2/3): """

try:

if\_ind = raw\_input(prompt).strip()[0]

except:

print 'Invalid input.'

sys.exit(1)

if if\_ind not in '0123':

print 'Wrong Selection. Use 0/1/2/3'

sys.exit(2)

fname = '/etc/sysconfig/network-scripts/ifcfg-eth%s' % if\_ind

ip\_addr = raw\_input('ip address: ').strip()

result = check\_ip(ip\_addr)

if not result:

print 'Invalid ip address'

sys.exit(3)

configip(fname, ip\_addr, if\_ind)

print '\033[32;1mConfigure ip address done. Please execute "systemctl restart NetworkManager"\033[0m'

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

show\_menu()

**112-模拟字符串lstrip用法**

思路：

1、取出字符串长度

2、通过range和字符串长度得到字符串下标

3、找到非空字符串下标，剩余部分取切片

4、如果字符串没有非空字符，返回空串

whitesps = ' \r\n\v\f\t'

def rmlsps(astr):

for i in range(len(astr)):

if astr[i] not in whitesps:

return astr[i:]

else: # 所有字符均为空，循环正常结束，返回空串

return ''

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

print(rmlsps(' \thello '))

**113-模拟字符串rstrip用法**

思路参考 [112-模拟字符串lstrip用法](https://www.jianshu.com/p/3e62945b4600)

whitesps = ' \r\n\v\f\t'

def rmrsps(astr):

for i in range(-1, -len(astr), -1): # 自右向左，下示为负

if astr[i] not in whitesps:

return astr[:i + 1] # 结束下标对应的字符不包含，所以加1

else:

return ''

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

print(rmrsps(''))

print(rmrsps(' \thello '))

**114-fork子进程解析**

当使用fork编写多进程的程序时，应该想清楚父子进程的工作各是什么。比如，让父进程生成子进程，子进程做具体的工作。当子进程执行完毕后，需要exit退出。如果不退出，它仍然在循环结构中，子进程还会再生成子进程。

以下代码：

import os

for i in range(3):

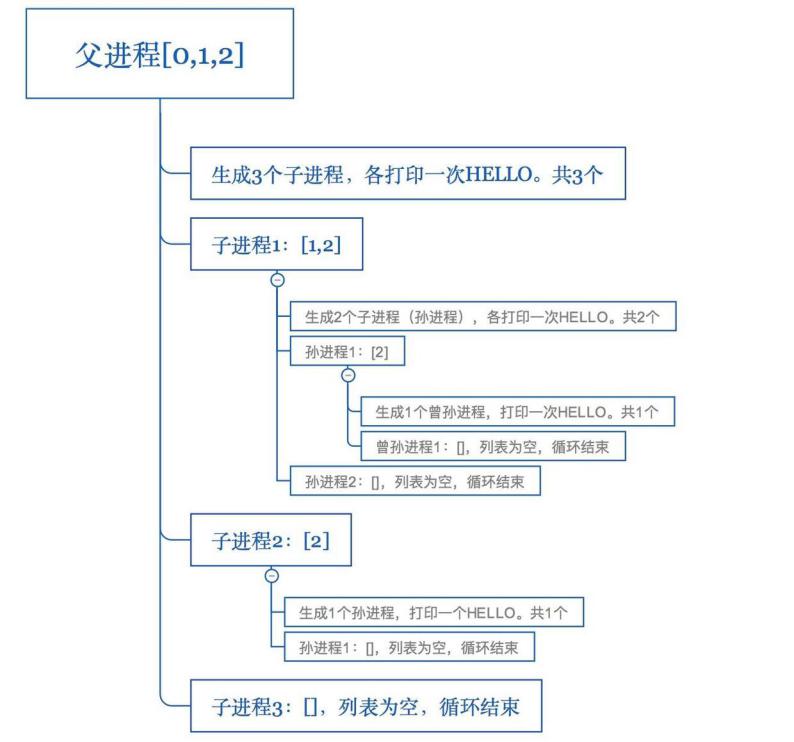
pid = os.fork()

if not pid:

print('hello')

执行时，屏幕上将打印7行hello。

分析如下：



fork.jpg

**115-可变与不可变对象的效率**

python是一种解释型的语言，执行效率要比C这样的编译型语言差得多，但是也应该注意它的效率。

python的各种数据类型，按更新模型可以分为可变类型（如列表、字典）和不可变类型（如数字、字符串和元组）。多使用可变类型，它的执行效率比不可变类型要高。

在《[37-生成密码/验证码](https://www.jianshu.com/p/b81b7ae19cb2)》中，将结果保存到了一个名为 result 的变量中。result 是字符串，字符串不可变，所以python在工作时，首先要申请一段内存储 result 的初值（空串''），随机取得一个字符后（如'a'），result += 'a'实际上是要重新申请一个新的内存，把新字符串存储进去。如此往复，有几次循环，就要重新分配几次内存。

如果变量 result 使用列表，只需要为其分配一次内存即可，因为列表是可变的。代码可以更改为以下样式：

from random import choice

import string

all\_chs = string.ascii\_letters + string.digits # 大小写字母加数字

def gen\_pass(n=8):

result = []

for i in range(n):

ch = choice(all\_chs)

result.append(ch)

return ''.join(result)

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

print(gen\_pass())

print(gen\_pass(4))

print(gen\_pass(10))

**116-ip地址与10进制数的转换**

我们先写个ping命令看看结果：

bogon:~ zhangzhigang$ ping -c2 2130706433

PING 2130706433 (127.0.0.1): 56 data bytes

64 bytes from 127.0.0.1: icmp\_seq=0 ttl=64 time=0.043 ms

64 bytes from 127.0.0.1: icmp\_seq=1 ttl=64 time=0.080 ms

--- 2130706433 ping statistics ---

2 packets transmitted, 2 packets received, 0.0% packet loss

round-trip min/avg/max/stddev = 0.043/0.061/0.080/0.018 ms

bogon:~ zhangzhigang$

当我们ping数字2130706433时，从127.0.0.1返回结果。为什么是这样呢？

IP地址是个32位的二进制数，表示成点分10进制，只是为了方便，如果把这32位二进制数转成10进制数，计算机也是认识的。

咱们就编写一个将10进制数转成2进制的代码吧。

思路：

1、仔细分析，IP地址的四个10进制数，实际上就是256进制

127.0.0.1 <=> 127 \* 256<sup>3</sup> + 0 \* 256<sup>2</sup> + 0 \* 256<sup>1</sup> + 1 \* 256<sup>0</sup>

2、10进制数除以256，余数就是IP地址右侧的数字，商继续除256即可

具代体码如下：

def int2ip(digit):

result = []

for i in range(4):

digit, mod = divmod(digit, 256)

result.insert(0, mod)

return '.'.join(map(str, result))

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

print(int2ip(2130706433))

map函数用法，参见《[70-匿名函数和map](<https://www.jianshu.com/p/ca5e5b92deeb)》>。

**117-比较文件的差异**

比较两个文件的差异，可以直接使用vim。

# vim -d /etc/passwd /etc/passwd-

python标准库提供了一个difflib，可以进行文件的比较，并且可以生成网页的形式。

import difflib

import webbrowser

import sys

import string

import os

from random import choice

def rand\_chs(n=8): # 默认生成8个随机字符

all\_chs = string.ascii\_letters + string.digits

result = [choice(all\_chs) for i in range(n)]

return ''.join(result)

# 函数接收两个相似的文件名，返回HTML形式的字符串

def make\_diff(lfile, rfile):

d = difflib.HtmlDiff()

# 将两个文件分别读到列表中

with open(lfile) as fobj:

ldata = fobj.readlines()

with open(rfile) as fobj:

rdata = fobj.readlines()

return d.make\_file(ldata, rdata) # 返回HTML格式内容

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

try:

lfile = sys.argv[1]

rfile = sys.argv[2]

except IndexError:

print('Usage: %s file1 file2' % sys.argv[0])

sys.exit(1)

if not os.path.isfile(lfile):

print('No such file:', lfile)

sys.exit(2)

if not os.path.isfile(rfile):

print('No such file:', rfile)

sys.exit(3)

data =make\_diff(lfile, rfile)

# 以下只是为说明内容增加中文显示，非必须项

data = data.replace(';Added', ';Added（增加）')

data = data.replace('>Changed', '>Changed（改变）')

data = data.replace('>Deleted', '>Deleted（被删除）')

data = data.replace('(f)irst change', '(f)irst change【第一处变更】')

data = data.replace('(n)ext change', '(n)ext change【下一处变更】')

data = data.replace('(t)op', '(t)op【回到顶部】')

html\_file = '/tmp/%s.html' % rand\_chs() # 用随机字符生成文件名

with open(html\_file, 'w') as fobj:

fobj.write(data)

webbrowser.open\_new\_tab('file:///%s' % html\_file) # 使用浏览器打开文件

**118-打造vim为python IDE**

pycharm之类的IDE很好，但是不能在无图形的终端下使用，另外它们是“重量级选手”。如果偶尔需要vim编辑python程序，能让vim支持python的语法提示不是更好！

实际上，vim支持python提示，就是把python以及各种各样模块的常用指令都写到一个字典里。

操作步骤如下：

1、创建vim插件工作目录

bogon:~ zhangzhigang$ mkdir -p ~/.vim/bundle/

2、下载插件

bogon:~ zhangzhigang$ cd ~/.vim/bundle/

bogon:bundle zhangzhigang$ git clone https://github.com/rkulla/pydiction.git

bogon:bundle zhangzhigang$ ls

pydiction

3、将pydiction目录中的after目录拷贝到 \~/.vim/目录。当vim执行时，会自动执行\~/.vim/目录中的内容

bogon:bundle zhangzhigang$ cp -r pydiction/after/ ~/.vim/

4、修改vim配置，设置打开以.py结尾的文件，按tab可以支持python语法补全

bogon:bundle zhangzhigang$ vim ~/.vimrc

filetype plugin on

let g:pydiction\_location = '~/.vim/bundle/pydiction/complete-dict'

set ai

set et

set ts=4

5、测试，注意文件名必须是以.py结尾，否则没有代码补全

bogon:bundle zhangzhigang$ vim a.py

imp<tab>

**119-python交互解释器tab补全功能**

进入python交互解释器后，按tab键默认是缩进功能，而不是代码补全。为了实现代码补全，可以采用如下操作：

1、创建指令补全文件

[root@room8pc16 ~]# vim /usr/local/bin/tab.py

from rlcompleter import readline

readline.parse\_and\_bind('tab: complete')

2、配置环境变量，在~/.bashrc中追加以下内容

[root@room8pc16 ~]# vim ~/.bashrc

export PYTHONSTARTUP='/usr/local/bin/tab.py'

3、source生效

[root@room8pc16 ~]# source ~/.bashrc

4、进入python解释器验证：

[root@room8pc16 ~]# python3

>>> pr<tab><tab> # 可实现语法提示

>>> pri<tab> # 补全

**120-钉钉机器人**

在很多情况下，如果能发消息到手机是一个非常实用的功能。比如，zabbix监控报警，可以采用执行脚本，向手机发送报警消息。

阿里巴巴的钉钉可以很方便的向用户发送消息，实际上就是建一个群，在群里面创建一个机器人。发消息就是通过脚本让机器人在群里说话而已。

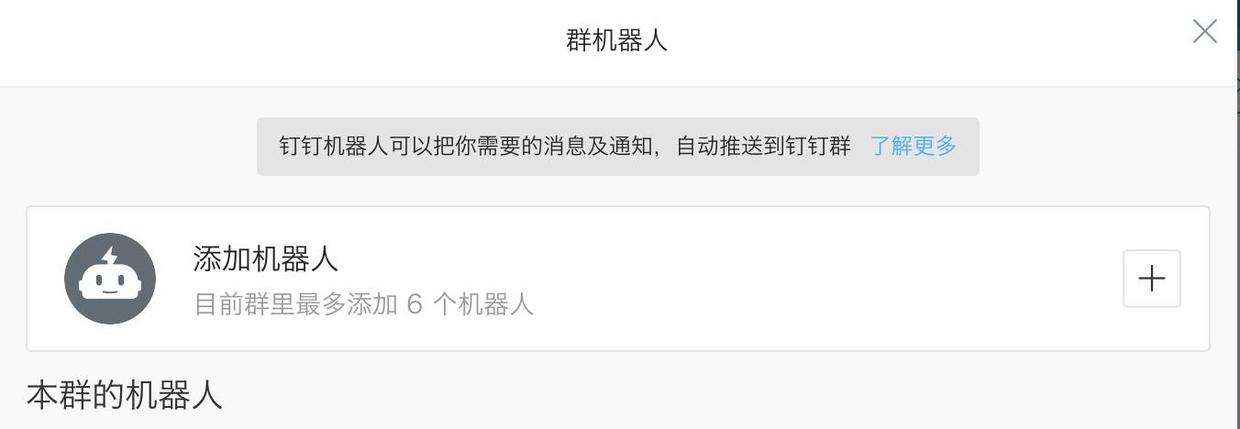
在群里创建机器人并设置，可以通过钉钉网页版，网址为：<https://im.dingtalk.com/>  
创建机器人的步骤如下：

1、在钉钉群聊的右上角点机器人



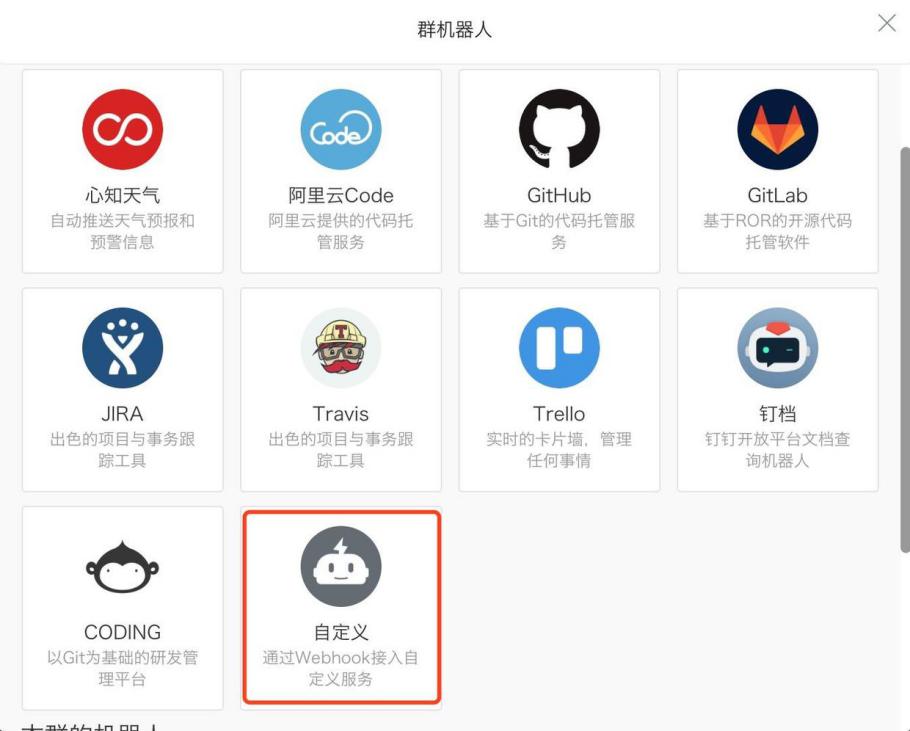
ding1.png

2、点击“+”添加机器人



ding2.png

3、选择机器人类型



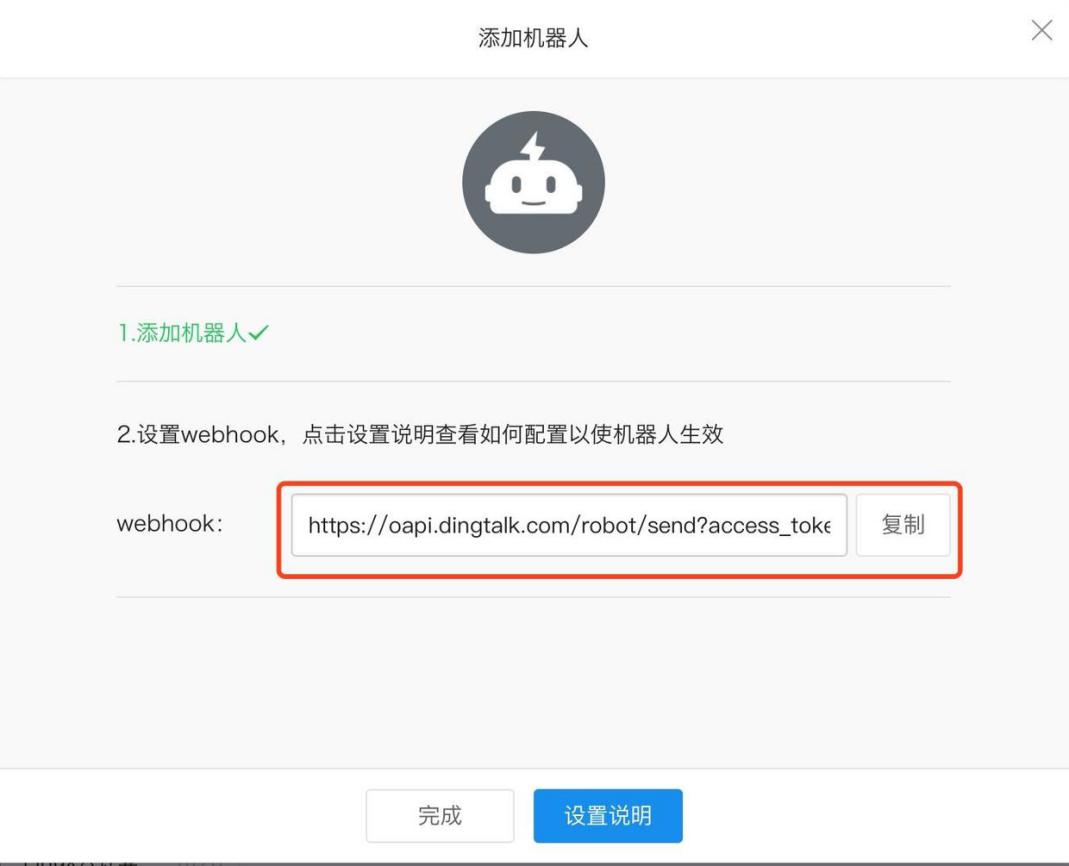
ding3.png

4、给机器人起名



ding4.png

5、将webhook内容保存下来，备用



ding5.png

编写脚本dingtalk.py：

#!/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,

"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 = ['15055667788'] # 特殊提醒要查看的人,就是@某人一下

url = 此处填写上面第5步webhook的内容

print(send\_msg(url, reminders, msg))

发送消息进行测试：

# python3 dingtalk.py "这是一个测试而已"

收到的消息如下：



ding6.png

钉钉机器人的设置，参见[《116-钉钉机器人》](https://www.jianshu.com/p/a3c62eb71ae3)。

zabbix服务器监控到异常，可以通过各种方式发送报警消息。配置步骤如下：  
1、创建报警脚本

[root@node2 ~]*# vi*m /usr/local/share/zabbix/alertscripts/dingalert.py

#!/usr/bin/env python

import json

import requests

import sys

def send\_msg(url, remiders, msg):

headers = {'Content-Type': 'application/json; charset=utf-8'}

data = {

"msgtype": "text",

"at": {

"atMobiles": remiders,

"isAtAll": False,

},

"text": {

"content": msg,

}

}

r = requests.post(url, data=json.dumps(data), headers=headers)

return r.text

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

msg = sys.argv[1]

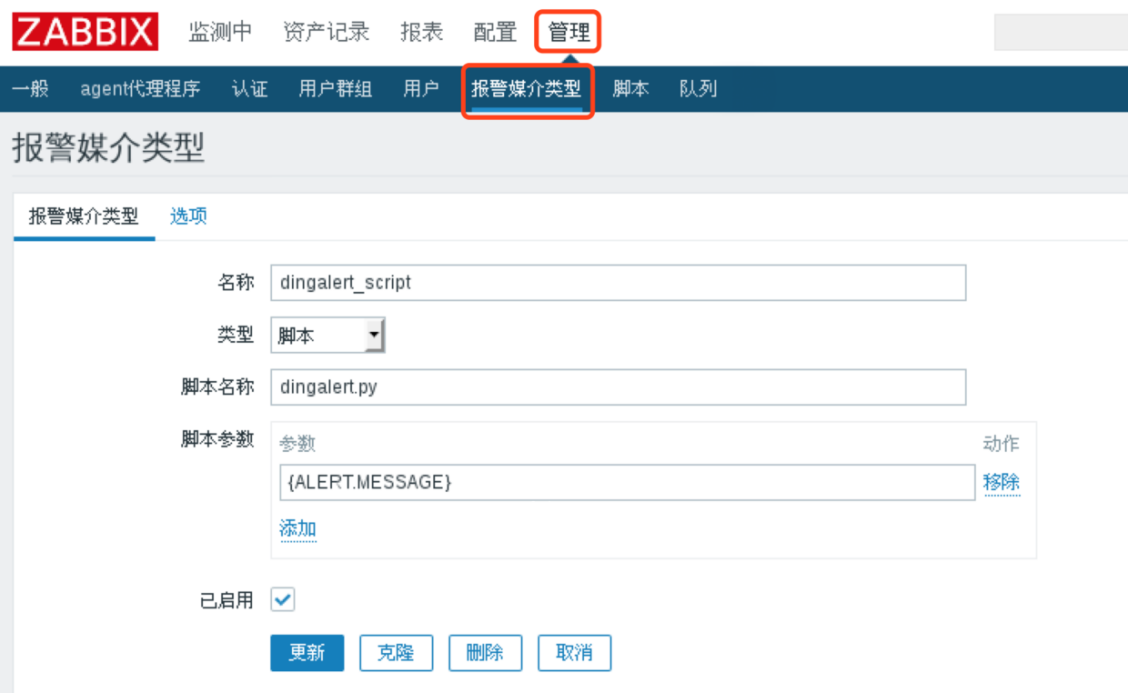
remiders = []

url = '钉钉机器人的URL'

print(send\_msg(url, remiders, msg))

[root@node2 ~]# chmod +x */usr/local/share/zabbix/alertscripts/dingalert.py*

2、添加报警媒介



z1.png

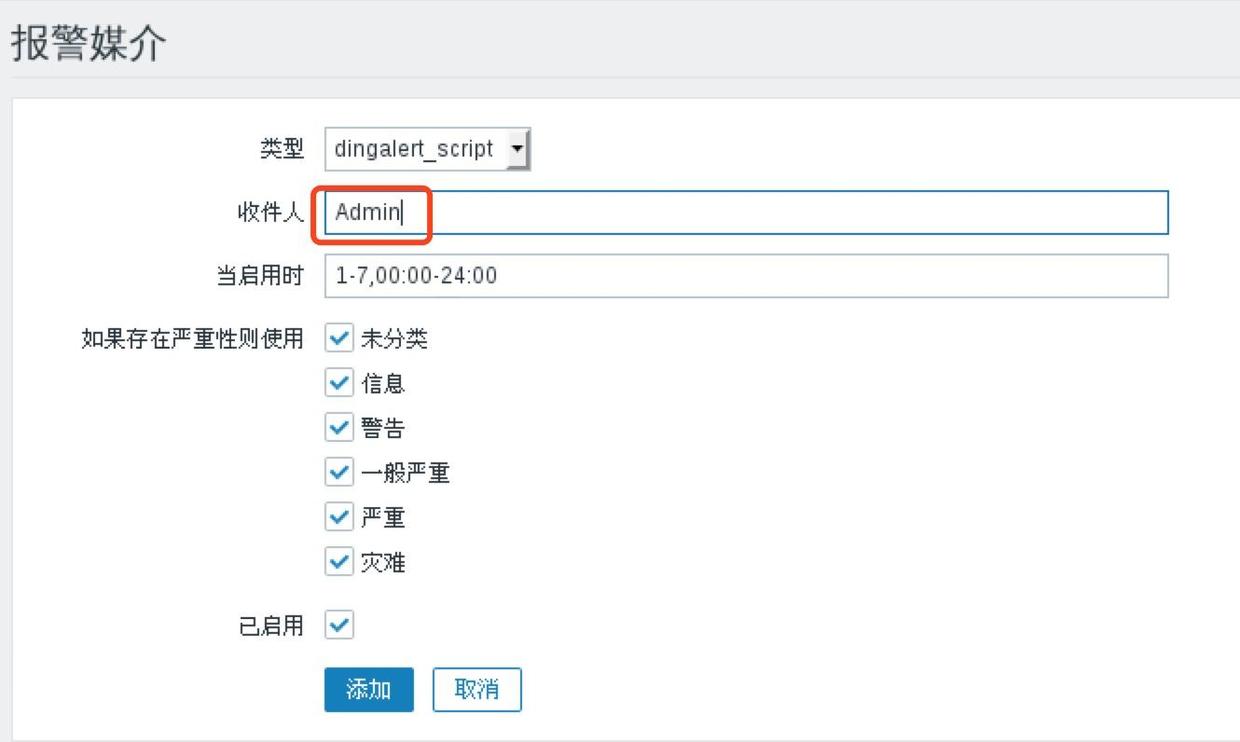
3、配置用户可以使用脚本报警



z2.png



z3.png



z4.png



z5.png

4、配置触发动作



z6.png



z7.png



z8.png



z9.png



z10.png

5、触发报警条件。如，本例中/boot分区空间不足20%将会发生报警。

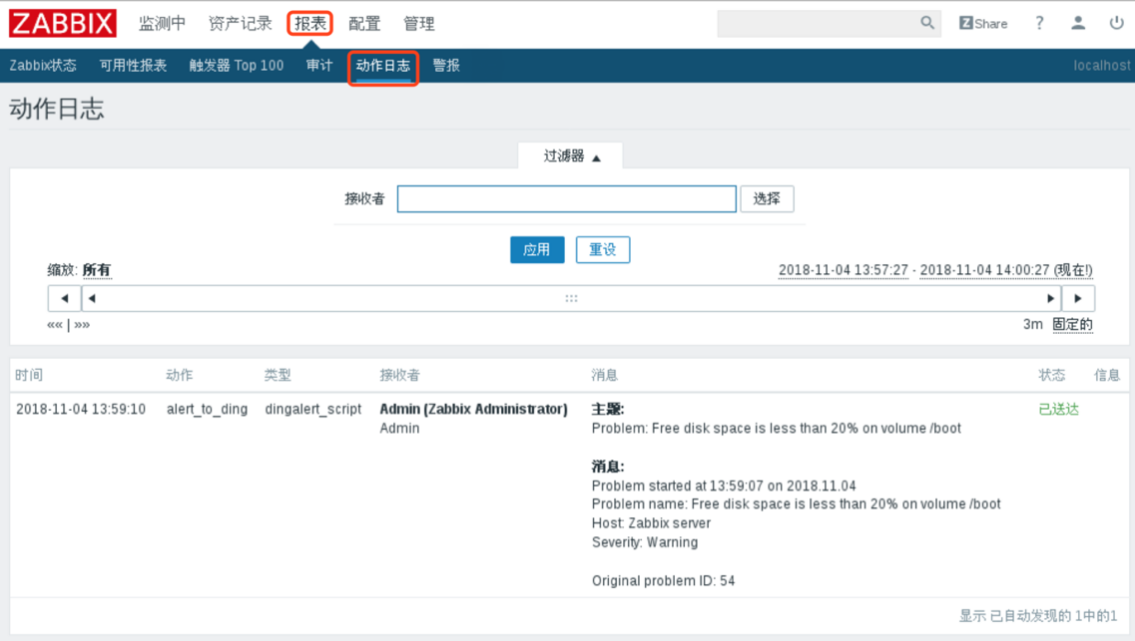
[root@node2 ~]# dd if=/dev/zero of=/boot/t.img bs=1M count=800

[root@node2 ~]# df -h /boot/

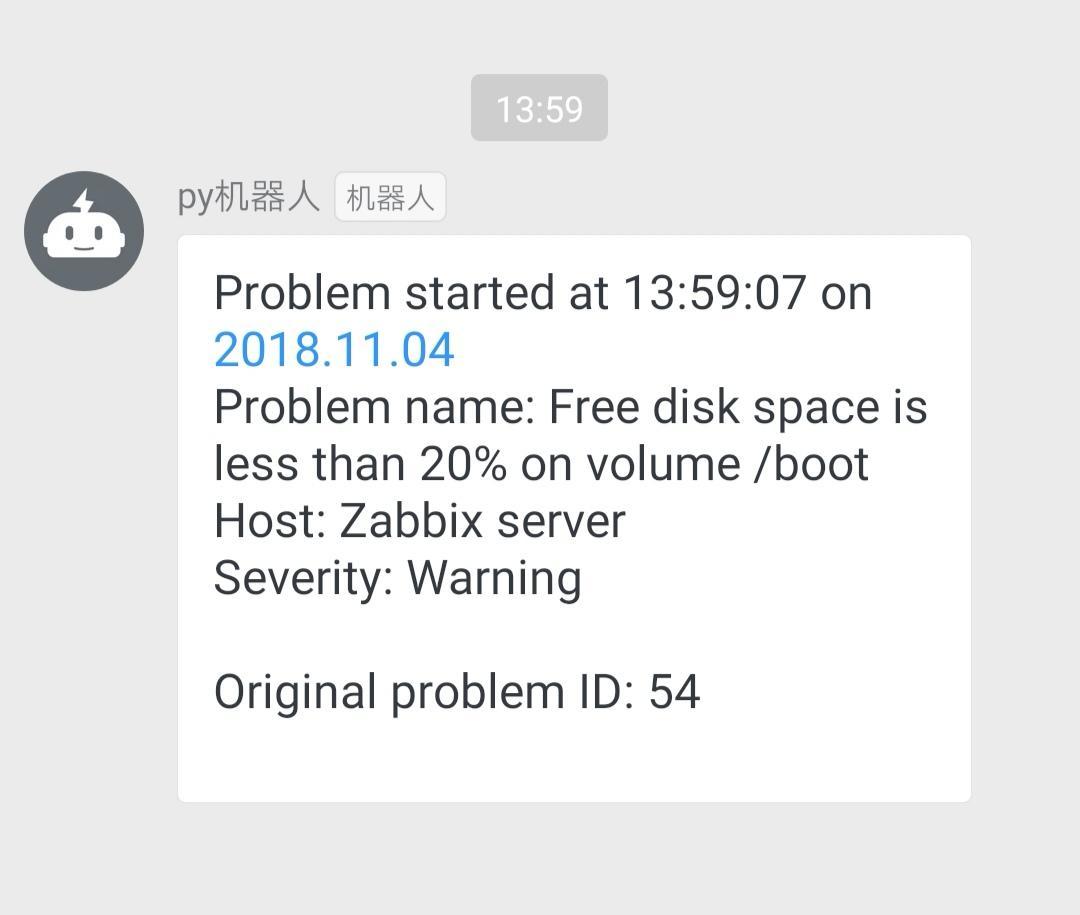
文件系统 容量 已用 可用 已用% 挂载点

/dev/vda1 1014M 961M 54M 95% /boot

6、查看动作日志及钉钉消息



z11.png



Screenshot\_20181104\_140234.jpg