



伟哥的私房菜 Python

51CTO.com
技术博客 Blog

王伟 著

Python具有易读、易学、易维护、可移植性、跨平台等特点。因而深受欢迎，并在最近几年迅速崛起。Python的作者有意的设计限制性很强的语法，使得不好的编程习惯都不能通过编译，python属于所想即所得的语言，实现功能简单浅显易懂。



机械工业出版社
CHINA MACHINE PRESS

伟哥的python私房菜

王伟 著

本书由作者授权北京华章图文信息有限公司在全球范围内以网络出版形式出版发行本作品中文版，未经出版者书面许可，本书的任何部分不得以任何方式抄袭，翻录或翻印。

策划编辑：杜正彬 责任编辑：李静

封面设计：梁杰

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @研发书局

腾讯微博 @yanfabook

目录

[图书简介](#)

[python join 和 split的常用使用方法](#)

[python中关于时间和日期函数的常用计算总结](#)

[Python 列表 list 数组 array 常用操作集锦](#)

[Python ConfigParser模块常用方法示例](#)

[python os.path模块常用方法详解](#)

[Python中执行系统命令常见的几种方法](#)

[python写的部署Windows下zabbix客户端脚本](#)

[Python字符串加密解密方法总结](#)

[python中用try来处理程序异常的集中常用方法](#)

[python监控单台多实例数据库服务器的数据库端口](#)

[python编程中常用的12种基础知识总结](#)

[python 中字符串大小写转换](#)

[zabbix自动添加删除主机的python脚本](#)

[python字符串按固定长度拆分](#)

[python脚本清除linux cron中过期的定时计划](#)

[Python用WMI模块获取windows系统信息](#)

[递归遍历目录中的所有文件](#)

[python wx 的wx.Frame框架属性](#)

[python写报警程序中的声音实现winsound](#)

[Errno 9: Bad file descriptor in python socket错误处理](#)

[python语言的基础规范](#)

[二、python脚本的规范：](#)

[python中用string.maketrans和translate巧妙替换字符串](#)

[python计算文件的行数和读取某一行内容的实现方法](#)

一、计算文件的行数

二、读取文件某一行的内容（测试过1G大小的文件，效率还可以）

三、用linecache读取文件内容（测试过1G大小的文件，效率还可以）

python linecache模块读取文件用法详解

python调用zabbix的api接口添加主机、查询组、主机、模板

python监控文件或目录变化

python用paramiko模块上传本地目录到远程目录

python写的分析mysql binlog日志工具

redis多实例重启脚本

计算mysql数据库目录中表文件大小并排序

监控redis多实例的负载情况

图书简介

Python具有易读、易学、易维护、可移植性、跨平台等特点而深受欢迎，并在最近几年迅速崛起。Python的笔者有意的设计限制性很强的语法，使得不好的编程习惯都不能通过编译，python属于所想即所得的语言，实现功能简单浅显易懂，本书主要是笔者在实践中所遇到的经常使用的技巧性文章，希望对读者有所帮助。

python join 和 split的常用使用方法

python join 和 split的使用方法不同, join用来连接字符串, split用来拆分字符串。

1.join用法示例

```
>>>li = ['my','name','is','bob']
>>>' '.join(li)
'my name is bob'
```

```
>>>'_'.join(li)
'my_name_is_bob'
```

```
>>> s = ['my','name','is','bob']
>>> ' '.join(s)
'my name is bob'
```

```
>>> '..'.join(s)
'my..name..is..bob'
```

2.split用法示例

```
>>> b = 'my..name..is..bob'
```

```
>>> b.split()
['my..name..is..bob']
```

```
>>> b.split("..")
['my', 'name', 'is', 'bob']
```

```
>>> b.split("..",0)
['my..name..is..bob']
```

```
>>> b.split("..",1)
['my', 'name..is..bob']
```

```
>>> b.split("..",2)
['my', 'name', 'is..bob']
```

```
>>> b.split("..",-1)
['my', 'name', 'is', 'bob']
```

可以看出 `b.split("..",-1)` 等价于 `b.split("..")`

python中关于时间和日期函数的常用计算总结

python 中关于时间和日期的函数有 time 和 datetime。

基本上常用的类有：datetime和timedelta两个。它们之间可以相互加减。每个类都有一些方法和属性可以查看具体的值，如 datetime可以查看天数(day)、小时数(hour)、星期几(weekday())等；timedelta可以查看天数(days)、秒数(seconds)等。

python中时间日期格式化符号：

%y	两位数的年份表示 (00-99)
%Y	四位数的年份表示 (000-9999)
%m	月份 (01-12)
%d	月内中的一天 (0-31)
%H	24小时制小时数 (0-23)
%I	12小时制小时数 (01-12)
%M	分钟数 (00-59)
%S	秒 (00-59)
%a	本地简化星期名称
%A	本地完整星期名称
%b	本地简化的月份名称
%B	本地完整的月份名称
%c	本地相应的日期表示和时间表示
%j	年内的一天 (001-366)
%p	本地A.M.或P.M.的等价符
%U	一年中的星期数 (00-53)，星期天为星期的开始
%w	星期 (0-6)，星期天为星期的开始
%W	一年中的星期数 (00-53)，星期一为星期的开始
%x	本地相应的日期表示
%X	本地相应的时间表示
%Z	当前时区的名称
%%	%号本身

Python 列表 list 数组 array 常用操作 集锦

Python中的列表(list)类似于C#中的可变数组(ArrayList)，用于顺序存储结构。

创建列表

```
sample_list = ['a', 1, ('a', 'b')]
```

Python 列表操作

```
sample_list = ['a', 'b', 0, 1, 3]
```

得到列表中的某一个值

```
value_start = sample_list[0]
```

```
end_value = sample_list[-1]
```

删除列表的第一个值

```
del sample_list[0]
```

在列表中插入一个值

```
sample_list[0:0] = ['sample value']
```

得到列表的长度

```
list_length = len(sample_list)
```

列表遍历

```
for element in sample_list:
```

```
print(element)
```

Python 列表高级操作/技巧

产生一个数值递增列表

```
num_inc_list = range(30)
```

```
#will return a list [0, 1, 2, ..., 29]
```

用某个固定值初始化列表

```
initial_value = 0
```

```
list_length = 5
```

```
sample_list = [ initial_value for i in  
range(10)]
```

```
sample_list = [initial_value]*list_length
```

```
# sample_list ==[0, 0, 0, 0, 0]
```

附：python内置类型

1、list：列表（即动态数组，C++标准库的vector，但可含不同类型的元素于一个list中）

a = ["I", "you", "he", "she"] #元素可为任何类型。

下标：按下标读写，就当作数组处理

以0开始，有负下标的使用

0第一个元素，-1最后一个元素，

-len第一个元素，len-1最后一个元素

取list的元素数量

len(list) #list的长度。实际该方法是调用了此对象的__len__(self)方法。

创建连续的list

L = range(1, 5) #即 L=[1, 2, 3, 4], 不含最后一个元素

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

list的方法

L.append(var) #追加元素

L.insert(index, var)

L.pop(var) #返回最后一个元素，并从list中将其删除

L.remove(var) #删除第一次出现的该元素

L.count(var) #该元素在列表中出现的个数

L.index(var) #该元素的位置, 无则抛异常

L.extend(list) #追加list，即合并list到L上

L.sort() #排序

L.reverse() #倒序

list 操作符":", "+", "*", 关键字del

`a[1:]` #片段操作符，用于子list的提取

`[1,2]+[3,4]` #为`[1,2,3,4]`。同`extend()`

`[2]*4` #为`[2,2,2,2]`

`del L[1]` #删除指定下标的元素

`del L[1:3]` #删除指定下标范围的元素

list的复制

`L1 = L` #L1为L的别名，用C来说就是指针地址相同，对L1操作即对L操作。函数参数就是这样传递的

`L1 = L[:]` #L1为L的克隆，即另一个拷贝

list comprehension

`[ <expr1> for k in L if <expr2> ]`

2、dictionary: 字典（即C++标准库的map）

`dict = {'ob1' : 'computer', 'ob2' : 'mouse', 'ob3' : 'printer'}`

每一个元素是pair，包含key、value两部分。key是Integer或string类型，value 是任意类型。

键是唯一的，字典只认最后一个赋的键值。

dictionary的方法

`D.get(key, 0)` #同`dict[key]`，多了个没有则返回缺省值，0。[]没有则抛异常

D.has_key(key) #有该键返回TRUE, 否则FALSE

D.keys() #返回字典键的列表

D.values()

D.items()

D.update(dict2) #增加合并字典

D.popitem() #得到一个pair, 并从字典中删除它。已空则抛异常

D.clear() #清空字典, 同del dict

D.copy() #拷贝字典

Dcmp(dict1, dict2) #比较字典, (优先级为元素个数、键大小、键值大小)

#第一个大返回1, 小返回-1, 一样返回0

dictionary的复制

dict1 = dict #别名

dict2=dict.copy() #克隆, 即另一个拷贝。

3、tuple: 元组 (即常量数组)

tuple = ('a', 'b', 'c', 'd', 'e')

可以用list的 [], :操作符提取元素。就是不能直接修改元素。

4、string: 字符串 (即不能修改的字符list)

```
str = "Hello My friend"
```

字符串是一个整体。如果你想直接修改字符串的某一部分，是不可能的。但我们能够读出字符串的某一部分。

子字符串的提取

```
str[:6]
```

字符串包含 判断操作符: in, not in

```
"He" in str
```

```
"she" not in str
```

string模块，还提供了很多方法，如

`S.find(substring, [start [,end]])` #可指范围
查找子串，返回索引值，否则返回-1

`S.rfind(substring, [start [,end]])` #反向查找

`S.index(substring, [start [,end]])` #同find，
只是找不到产生ValueError异常

`S.rindex(substring, [start [,end]])` #同上反向
查找

`S.count(substring, [start [,end]])` #返回找到
子串的个数

`S.lowercase()`

`S.capitalize()` #首字母大写

S.lower() #转小写

S.upper() #转大写

S.swapcase() #大小写互换

分 S.split(str, ' ') #将string转list, 以空格切

接 S.join(list, ' ') #将list转string, 以空格连

处理字符串的内置函数

len(str) #串长度

cmp("my friend", str) #字符串比较。第一个大, 返回1

max(' abcxyz') #寻找字符串中最大的字符

min(' abcxyz') #寻找字符串中最小的字符

string的转换

0.1 float(str) #变成浮点数, float("1e-1") 结果为

int(str) #变成整型, int("12") 结果为12

int(str, base) #变成base进制整型数,
int("11", 2) 结果为2

long(str) #变成长整型,

long(str, base) #变成base进制长整型,

字符串的格式化（注意其转义字符，大多如C语言的，略）

`str_format %`（参数列表）

Python ConfigParser模块常用方法示例

```
print "db_pass:", db_pass
print "thread:", threads
print "processor:", processors
```

#修改一个值，再写回去

```
cf.set("db", "db_pass", "zhaowei")
cf.write(open("test.conf", "w"))
```

#添加一个section。（同样要写回）

```
cf.add_section('liuqing')
cf.set('liuqing', 'int', '15')
cf.set('liuqing', 'bool', 'true')
cf.set('liuqing', 'float', '3.1415')
cf.set('liuqing', 'baz', 'fun')
cf.set('liuqing', 'bar', 'Python')
cf.set('liuqing', 'foo', '%(bar)s is %(baz)s!')
cf.write(open("test.conf", "w"))
```

#移除section 或者option 。（只要进行了修改就要写回的哦）

```
cf.remove_option('liuqing', 'int')
cf.remove_section('liuqing')
cf.write(open("test.conf", "w"))
```

以上就是对Python ConfigParser模块的相关应用方法的介绍,当然，这个模块还有许多其他的用法，有

兴 趣 的 可 以 去 官 方 网 站 看 看 :
[http://docs.python.org/2/library/configparser.h](http://docs.python.org/2/library/configparser.html)
tml。

python os.path模块常用方法详解

os.path模块主要用于文件的属性获取，在编程中经常用到，以下是该模块的几种常用方法。更多的方法可以去查看官方文档：
<http://docs.python.org/library/os.path.html>

1. os.path.abspath(path)

返回path规范化的绝对路径。

```
>>> os.path.abspath('test.csv')
'C:\\Python25\\test.csv'
>>> os.path.abspath('c:\\test.csv')
'c:\\test.csv'
>>> os.path.abspath('../csv\\test.csv')
'C:\\csv\\test.csv'
```

2. os.path.split(path)

将path分割成目录和文件名二元组返回。

```
>>> os.path.split('c:\\csv\\test.csv')
('c:\\csv', 'test.csv')
>>> os.path.split('c:\\csv\\')
('c:\\csv', '')
```

3. os.path.dirname(path)

返回path的目录。其实就是os.path.split(path)的第一个元素。

```
>>> os.path.dirname('c:\\csv\\test.csv')
'c:\\'
>>> os.path.dirname('c:\\csv')
'c:\\'
```

4. os.path.basename(path)

返回path最后的文件名。如果path以 / 或\结尾，那么就会返回空值。即os.path.split(path)的第二个元素。

```
>>> os.path.basename('c:\\test.csv')
'test.csv'
>>> os.path.basename('c:\\csv')
'csv' （这里csv被当作文件名处理了）
>>> os.path.basename('c:\\csv\\')
''
```

5. os.path.commonprefix(list)

返回list中，所有path共有的最长的路径。

如：

```
>>> os.path.commonprefix(['/home/td', '/home/td/ff', '/home/td/fff'])
'/home/td'
```

6. os.path.exists(path)

如果path存在，返回True；如果path不存在，返回False。

```
>>> os.path.exists('c:\\')
True
>>> os.path.exists('c:\\csv\\test.csv')
False
```

7. os.path.isabs(path)

如果path是绝对路径，返回True。

8. os.path.isfile(path)

如果path是一个存在的文件，返回True。否则返回False。

```
>>> os.path.isfile('c:\\boot.ini')
True
>>> os.path.isfile('c:\\csv\\test.csv')
False
>>> os.path.isfile('c:\\csv\\')
False
```

9. os.path.isdir(path)

如果path是一个存在的目录，则返回True。否则返回False。

```
>>> os.path.isdir('c:\\')
True
>>> os.path.isdir('c:\\csv\\')
False
>>> os.path.isdir('c:\\windows\\test.csv')
False
```

10. os.path.join(path1[, path2[, ...]])

将多个路径组合后返回，第一个绝对路径之前的参数将被忽略。

```
>>> os.path.join('c:\\', 'csv', 'test.csv')
'c:\\csv\\test.csv'
>>> os.path.join('windows\\temp', 'c:\\', 'csv', 'test.csv')
'c:\\csv\\test.csv'
>>> os.path.join('/home/aa', '/home/aa/bb', '/home/aa/bb/c')
'/home/aa/bb/c'
```

11. os.path.normcase(path)

在Linux和Mac平台上，该函数会原样返回path，在windows平台上会将路径中所有字符转换为小写，并将所有斜杠转换为饭斜杠。

```
>>> os.path.normcase('c:/windows\\system32\\')
'c:\\windows\\system32\\'
```

12. os.path.normpath(path)

规范化路径。

```
>>> os.path.normpath('c://windows\\System32\\../Temp/')
'c:\\windows\\Temp'
```

12. os.path.splitdrive(path)

返回 (drivename, fpath) 元组 。


```
>>> os.path.splitdrive('c:\\windows')  
('c:', '\\windows')
```

13. os.path.splitext(path)

分离文件名与扩展名；默认返回(fname, fextension)元组，可做分片操作。

```
>>> os.path.splitext('c:\\csv\\test.csv')  
('c:\\csv\\test', '.csv')
```

14. os.path.getsize(path)

返回path的文件的大小（字节）。

```
>>> os.path.getsize('c:\\boot.ini')  
299L
```

15. os.path.getatime(path)

返回path所指向的文件或者目录的最后存取时间。

16. os.path.getmtime(path)

返回path所指向的文件或者目录的最后修改时间。

Python中执行系统命令常见的几种方法

Python中执行系统命令常见的几种方法有：

(1) os.system

仅仅在一个子终端运行系统命令，而不能获取命令执行后的返回信息

如果再命令行下执行，结果直接打印出来

例如：

```
>>> import os
>>> os.system('ls')
chk_err_log.py CmdTool.log  install_log.txt  install_zabbix.
sh  manage_deploy.sh  MegaSAS.log
```

(2) os.popen

#该方法不但执行命令还返回执行后的信息对象

#好处在于：将返回的结果赋于一变量，便于程序的处理。

例如：

```
>>> import os
>>> tmp = os.popen('ls *.sh').readlines()
>>> tmp
['install_zabbix.sh\n', 'manage_deploy.sh\n', 'mysql_setup.s
h\n', 'python_manage_deploy.sh\n', 'setup.sh\n']
```

(3) 使用模块subprocess

使用方法:

```
>>> import subprocess
>>> subprocess.call(["cmd", "arg1", "arg2"], shell=True)
```

好处在于:运用对线程的控制和监控,将返回的结果赋于一变量,便于程序的处理。

如获取返回和输出:

```
import subprocess
p = subprocess.Popen('ls *.sh', shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
print p.stdout.readlines()
for line in p.stdout.readlines():
    print line,
retval = p.wait()
```

(4) 使用模块commands模块

常用的主要有两个方法: getoutput 和 getstatusoutput

```
>>> import commands

>>> commands.getoutput('ls *.sh')
'install_zabbix.sh\nmanage_deploy.sh\nmysql_setup.sh\npython_manage_deploy.sh\nsetup.sh'

>>> commands.getstatusoutput('ls *.sh')
(0, 'install_zabbix.sh\nmanage_deploy.sh\nmysql_setup.sh\npython_manage_deploy.sh\nsetup.sh')
```

注意： 当执行命令的参数或者返回中包含了中文文字，那么建议使用subprocess，如果使用os.popen则会出现错误。

python写的部署Windows下zabbix客户端脚本

```
#!/bin/env python
# -*- coding: utf-8 -*-
#####
# @This script is used to Install zabbix client for Windows(32/64)
# @Function:      Install zabbix client for Windows
# @Create Date:   2013-01-02
```

#打包官方 zabbix_agents_2.0.4.win.zip 为 rar 包，指定解压软件rar的路径。

```
#####
import os,re,sys,urllib,wmi
c = wmi.WMI ()
for s in c.Win32_Service ():
    if s.Caption == "Zabbix Agent":
        sys.exit("zabbix already install")
url = 'http://192.168.110.110/zabbix_agents_2.0.4.win.rar'
local = 'C:\\zabbix_agents_2.0.4.win.rar'
urllib.urlretrieve(url,local,)
url = 'http://192.168.110.110/rar.exe'
local = 'C:\\rar.exe'#本地如果安装了指定相应的目录
urllib.urlretrieve(url,local,)
os.popen('C:\\rar.exe x -y C:\\zabbix_agents_2.0.4.win.rar -
ed C:\\')
sProgramFiles = os.environ['PROGRAMFILES']
if "(86)" in sProgramFiles:
    os.popen('C:\\zabbix\\bin\\win64\\zabbix_agentd.exe -
c C:\\zabbix\\conf\\zabbix_agentd.win.conf -i')
else:
    os.popen('C:\\zabbix\\bin\\win32\\zabbix_agentd.exe -
```

```

c C:\\zabbix\\conf\\zabbix_agentd.win.conf -i')
#以下是添加windows网卡流量监控的自定key。注意，一般的执行命令或脚本自定义 key 格式为： UserParameter=keyname , commd , 网卡的
为 PerfCounter=keyname, ""
conm = os.popen('typeperf.exe -qx|find "Network Interface"|find "Bytes"|find /v "Total"|find /v "Loopback").readlines()
f=open('C:\\zabbix\\conf\\zabbix_agentd.win.conf','a+')
f.write('\\n')
e = 0
for i in range(len(conm)):
    c = re.search('Sent',conm[i])
    if c:
        b = "PerfCounter " + "=" + " eth"+str(e) + "_Out," +
        '""' + str(conm[i]).strip() + '",30'
        #print b
        f.write("%s \\n" %b)
        e += 1
e = 0
for i in range(len(conm)):
    c = re.search('Received',conm[i])
    if c:
        b = "PerfCounter " + "=" + " eth"+str(e) + "_In," +
        '""' + str(conm[i]).strip() + '",30'
        #print b
        f.write("%s \\n" %b)
        e += 1
f.close()
'''
f = open('C:\\zabbix\\zabbix_agentd.conf','r+')
ip = f.read()
ip = ip.replace('192.168.1.100',ipnew)
f.seek(0)
f.write(ip)
f.close()
'''
os.popen('net start "Zabbix Agent"')
os.popen('net stop "Zabbix Agent"')
os.popen('net start "Zabbix Agent"')
os.remove('C:\\\\rar.exe')
os.remove('C:\\\\zabbix_agents_2.0.4.win.rar')
sys.exit("zabbix install success !")

```

Python字符串加密解密方法总结

编程中经常会对字符串做加密解密处理，特别是涉及隐私的字符串，如密码等，这时候，就需要加密。加密解密方法大致有三种：base64，win32com.client和自己写加密解密算法，其中最安全的就是自己写加密解密算法了。

1. 最简单的方法是用base64

```
import base64

s1 = base64.encodestring('hello world')
s2 = base64.decodestring(s1)
print s1,s2

# aGVsbG8gd29ybGQ=\n
# hello world
```

注：这是最简单的方法了，但是不够保险，因为如果别人拿到你的密文，也可以自己解密来得到明文；不过可以把密文字符串进行处理，如字母转换成数字或是特殊字符等，自己解密的时候再替换回去，再进行base64.decodestring，这样要安全很多。

2. 第二种方法是使用win32com.client

```
import win32com.client
def encrypt(key,content): # key:密钥,content:明文
    EncryptedData = win32com.client.Dispatch('CAPICOM.EncryptedData')
    EncryptedData.Algorithm.KeyLength = 5
```

```

    EncryptedData.Algorithm.Name = 2
    EncryptedData.SetSecret(key)
    EncryptedData.Content = content
    return EncryptedData.Encrypt()

```



```

def decrypt(key, content): # key:密钥, content:密文
    EncryptedData = win32com.client.Dispatch('CAPICOM.EncryptedData')
    EncryptedData.Algorithm.KeyLength = 5
    EncryptedData.Algorithm.Name = 2
    EncryptedData.SetSecret(key)
    EncryptedData.Decrypt(content)
    str = EncryptedData.Content
    return str

```



```

s1 = encrypt('lovebread', 'hello world')
s2 = decrypt('lovebread', s1)
print s1,s2

```



```

# MGEGCSsGAQQBgjdYA6BUMFIGCisGAQQBgjdYAwGgRDBCAGMCAAECAMYBAG
FABAgq
# GpllWj9cswQQh/fnBUZ6ijwKDTH9DLZmBgQYmfaZ3VFyS/lq391oDtjlCR
FGnXpx
# lG7o
# hello world

```

注：这种方法也很方便，而且可以设置自己的密钥，比第一种方法更加安全，如果对安全级别要求不太高，这种方法是加密解密的首选之策！

3. 自己写加密解密算法

比如：

```

def encrypt(key, s):
    b = bytearray(str(s).encode("gbk"))
    n = len(b) # 求出 b 的字节数
    c = bytearray(n*2)
    j = 0
    for i in range(0, n):

```



```

        b1 = b[i]
        b2 = b1 ^ key # b1 = b2 ^ key
        c1 = b2 % 16
        c2 = b2 // 16 # b2 = c2*16 + c1
        c1 = c1 + 65
        c2 = c2 + 65 # c1,c2都是0~15之间的数,加上65就变成了A-
P 的字符的编码
        c[j] = c1
        c[j+1] = c2
        j = j+2
    return c.decode("gbk")
def decrypt(key, s):
    c = bytearray(str(s).encode("gbk"))
    n = len(c) # 计算 b 的字节数
    if n % 2 != 0 :
        return ""
    n = n // 2
    b = bytearray(n)
    j = 0
    for i in range(0, n):
        c1 = c[j]
        c2 = c[j+1]
        j = j+2
        c1 = c1 - 65
        c2 = c2 - 65
        b2 = c2*16 + c1
        b1 = b2 ^ key
        b[i]= b1
    try:
        return b.decode("gbk")
    except:
        return "failed"
key = 15
s1 = encrypt(key, 'hello world')
s2 = decrypt(key, s1)
print s1,'\n',s2

# HGKGDGDGAGPCIHAGNHDGLG
# hello world

```

注：这是从网上借鉴的一个简单的例子，大家可以自定义算法进行加密解密；还有许许多多复杂的加

密算法，大家可以自行查阅密码学的相关算法。

4. 把python源码文件编译成pyc二进制格式的文件

对于python来说，也可以把python源码文件编译成pyc二进制格式的文件，这样别人就看不到你的源码了，也算是一种加密方法，方法如下：

执 行 命 令 `python -m py_compile create_slave.py` 可 以 直 接 生 成 一 个 `create_slave.pyc` 文 件 ， 然 后 可 以 用 `create_slave.pyc`来替换`create_slave.py`作为脚本来执行。

python中用try来处理程序异常的集中常用方法

如果你在写python程序时遇到异常后想进行如下处理的话,一般用try来处理异常,假设有下面的一段程序:

```
try:
    语句1
    语句2
    .
    语句N
except .....:
    print .....
```

如果你并不知道“语句1至语句N”在执行时会出什么样的异常,但你还要做异常处理,且想把出现的异常打印出来,并不停止程序的运行,那么在“except”这句应怎样来写呢?

总结了一下至少3个方法:

方法一: 捕获所有异常

```
try:
    a=b
    b=c
except Exception,e:
    print Exception,":",e
```

方法二: 采用traceback模块查看异常

```
import traceback
try:
    a=b
    b=c
except:
    traceback.print_exc()
```

方法三：采用sys模块回溯最后的异常

```
import sys
try:
    a=b
    b=c
except:
    info=sys.exc_info()
    print info[0],":",info[1]
```

如果你还想把这些异常保存到一个日志文件中，来分析这些异常，那么采用下面的方法：

把`traceback.print_exc()`打印在屏幕上的信息保存到一个文本文件中

```
try:
    a=b
    b=c
except:
    f=open("c:log.txt",'a')
    traceback.print_exc(file=f)
    f.flush()
    f.close()
```

python监控单台多实例数据库服务器的数据库端口

数据库的服务器端口是什么，用来做什么的，拿资产数据库的端口和服务器本地运行的数据库端口进行对比，报出没有运行的mysql实例以及他的用途。一种方法是根据“ps auxww|grep mysqld|grep -v root|grep -v grep”这个命令抓取本地运行的数据库端口，也可以根据netstat命令来获取本地数据库实例的所有端口；还有一种方法就是从资产中得到这个服务器应该运行的所有mysql端口，用python的socket模块来检测端口是否存活，这种方法比较简单一些。笔者使用第一种方法是因为这段代码已经写过并用于其他用途，等于复用，图个省事。以下是代码内容：

```
#!/bin/env python
# -*- coding: utf-8 -*-

import os,sys,MySQLdb

def center(sql):#连接数据库
    try:
        center_ip = '192.168.1.100'
        center_user = 'root'
        center_passwd = 'xxxxxx'
        conn = MySQLdb.connect(host = center_ip,user = center_user,passwd = center_passwd,charset='utf8',connect_timeout=20)
        cursor = conn.cursor()
        cursor.execute(sql)
        alldata = cursor.fetchall()
        cursor.close()
        conn.close()
        return alldata
```

```

    except:
        return 0

class check_port():#在资产中获取本地IP中应该有多少个mysql实例端口

    def __init__(self):
        conn = "ip a|grep glob|grep -
v '192.168'|awk '{print $2}'"
        self.host = os.popen(conn).readlines()[0].split("/")
[0]

    def remot(self):
        sql = "SELECT PORT FROM center.host_info WHERE flag=
'1' AND del_info!=0 AND ip='%s' AND host_name IN ('a','b','c
')" % self.host
        alldata = center(sql)
        cent_port = []
        if alldata != 0:
            for i in alldata:
                cent_port.append(str(i[0]))
            return cent_port
        else:
            return cent_port

    def local(self):#获取本地mysql有多少个实例运行
        psinfo = os.popen("ps auxww|grep mysqld|grep -
v root|grep -v grep").readlines()
        local_port = []
        if not psinfo:
            return local_port
        for i in psinfo:
            for j in i.split("--"):
                if j.find("port") != -1:
                    port = j.split("=")[1].strip()
                    local_port.append(port)
        return local_port

    def main(self):
        local_port = self.local()
        cent_port = self.remot()
        cent_port.sort()
        local_port.sort()
        if local_port == cent_port and len(local_port) != 0
and len(cent_port) != 0:
            print 0

```

```

        else:
            error = ""
            diff_list = list(set(local_port) ^ set(cent_port
))
            for port in diff_list:
                sql = "SELECT CONCAT(a.main_name,'_',b.sub_n
ame,'_',c.app_name,'_',c.port) AS used FROM center_app.main_
category a, center_app.sub_category b, center_app.app_info c
WHERE a.id = b.main_id AND b.dist_id = c.dist_id AND b.main
_id = c.main_id AND b.main_id='2' AND c.flag='1' AND c.del_i
nfo!=0 AND c.ip='%s' and c.port='%s'" % (self.host,port)
                alldata = center(sql)
                if error == "":
                    error = error + alldata[0][0]
                else:
                    error = error + ";" + alldata[0][0]
            print error
    if __name__ == "__main__":
        boss = check_port()
        boss.main()

```

如果用第二种方法的话，很简单，用下面的函数可以实现这个端口测试：

```

import socket

def test_port():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(2)
    address = '127.0.0.1'
    port = 80
    try:
        s.connect((address,port))
        return True
    except Exception,e:
        return False
s.close()

```

python脚本判断一个数是否为素数的几种方法

质数又称素数。指在一个大于1的自然数中，除了1和此整数自身外，不能被其他自然数整除的数。素数在数论中有着很重要的地位。比1大但不是素数的数称为合数。1和0既非素数也非合数。质数是与合数相对立的两个概念，二者构成了数论当中最基础的定义之一。基于质数定义的基础之上而建立的问题有很多世界级的难题，如哥德巴赫猜想等。算术基本定理证明每个大于1的正整数都可以写成素数的乘积，并且这种乘积的形式是唯一的。这个定理的重要一点是，将1排斥在素数集合以外。如果1被认为是素数，那么这些严格的阐述就不得不加上一些限制条件。

前几天偶尔的有朋友问python怎么判断素数的方法，在网上查了查，总结了python脚本判断一个数是否为素数的几种方法：

```
#运用python的数学函数
import math

def isPrime(n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

#单行程序扫描素数
from math import sqrt
N = 100
```



```
[ p for p in range(2, N) if 0 not in [ p% d for d in range(2, int(sqrt(p))+1)] ]
```

#运用python的itertools模块

```
from itertools import count
```

```
def isPrime(n):  
    if n <= 1:  
        return False  
    for i in count(2):  
        if i * i > n:  
            return True  
        if n % i == 0:  
            return False
```

#不使用模块的两种方法

```
def isPrime(n):  
    if n <= 1:  
        return False  
    i = 2  
    while i*i <= n:  
        if n % i == 0:  
            return False  
        i += 1  
    return True
```

```
def isPrime(n):  
    if n <= 1:  
        return False  
    if n == 2:  
        return True  
    if n % 2 == 0:  
        return False  
    i = 3  
    while i * i <= n:  
        if n % i == 0:  
            return False  
        i += 2  
    return True
```

python编程中常用的12种基础知识总结

python编程中常用的12种基础知识总结：正则表达式替换，遍历目录方法，列表按列排序、去重，字典排序，字典、列表、字符串互转，时间对象操作，命令行参数解析(getopt)，print 格式化输出，进制转换，Python调用系统命令或者脚本，Python 读写文件。

1、正则表达式替换

目标：将字符串line中的 overview.gif 替换成其他字符串

```
>>> line = '<IMG ALIGN="middle" SRC=\'#\'' /span>
>>> mo=re.compile(r'(?<=SRC=)"([\w+\.]+)"',re.I)
>>> mo.sub(r'\1****"',line)
'<IMG ALIGN="middle" SRC=\'#\'' /span>
>>> mo.sub(r'replace_str_\1',line)
'<IMG ALIGN="middle" replace_str_overview.gif BORDER="0" ALT="">'< /span>
>>> mo.sub(r'"testetstset"',line)
'<IMG ALIGN="middle" SRC=\'#\'' /span>
```

注意：其中 \1 是匹配到的数据，可以通过这样的方式直接引用。

2、遍历目录方法

在某些时候，我们需要遍历某个目录找出特定的文件列表，可以通过os.walk方法来遍历，非常方便。

```
import os
```

```

fileList = []
rootdir = "/data"
for root, subFolders, files in os.walk(rootdir):
    if '.svn' in subFolders: subFolders.remove('.svn') # 排除特
    定目录
    for file in files:
        if file.find(".t2t") != -1: # 查找特定扩展名的文件
            file_dir_path = os.path.join(root, file)
            fileList.append(file_dir_path)
print fileList

```

3、列表按列排序(list sort)

如果列表的每个元素都是一个元组(tuple), 我们要根据元组的某列来排序, 可参考如下方法,

下面的例子是根据元组的第2列和第3列数据来排序的, 而且是倒序(reverse=True)。

```

>>> a = [('2011-03-17', '2.26', 6429600, '0.0'), ('2011-03-
16', '2.26', 12036900, '-3.0'), ('2011-03-
15', '2.33', 15615500, '-19.1')]
>>> print a[0][0]
2011-03-17
>>> b = sorted(a, key=lambda result: result[1], reverse=True)
>>> print b
[('2011-03-15', '2.33', 15615500, '-19.1'), ('2011-03-
17', '2.26', 6429600, '0.0'), ('2011-03-
16', '2.26', 12036900, '-3.0')]
>>> c = sorted(a, key=lambda result: result[2], reverse=True)
>>> print c
[('2011-03-15', '2.33', 15615500, '-19.1'), ('2011-03-
16', '2.26', 12036900, '-3.0'), ('2011-03-
17', '2.26', 6429600, '0.0')]

```

4、列表去重(list uniq)

有时候需要将list中重复的元素删除，就要使用如下方法

```
>>> lst= [(1, 'sss'), (2, 'fsdf'), (1, 'sss'), (3, 'fd')]
>>> set(lst)
set([(2, 'fsdf'), (3, 'fd'), (1, 'sss')])
>>>
>>> lst = [1, 1, 3, 4, 4, 5, 6, 7, 6]
>>> set(lst)
set([1, 3, 4, 5, 6, 7])
```

5、字典排序(dict sort)

一般来说，我们都是根据字典的key来进行排序，但是我们如果想根据字典的value值来排序，就使用如下方法，

```
>>> from operator import itemgetter
>>> aa = {"a": "1", "sss": "2", "ffdf": "5", "ffff2": "3"}
>>> sort_aa = sorted(aa.items(), key=itemgetter(1))
>>> sort_aa
[('a', '1'), ('sss', '2'), ('ffff2', '3'), ('ffdf', '5')]
```

从上面的运行结果看到，按照字典的value值进行排序的。

6、字典, 列表, 字符串互转

以下是生成数据库连接字符串，从字典转换到字符串，

```
>>> params = {"server": "mpilgrim", "database": "master", "uid": "sa", "pwd": "secret"}
>>> ["%s=%s" % (k, v) for k, v in params.items()]
['server=mpilgrim', 'uid=sa', 'database=master', 'pwd=secret']
```

```
']  
>>> ";".join(["%s=%s" % (k, v) for k, v in params.items()])  
'server=mpilgrim;uid=sa;database=master;pwd=secret'
```

下面的例子 是将字符串转化为字典

```
>>> a = 'server=mpilgrim;uid=sa;database=master;pwd=secret'  
>>> aa = {}  
>>> for i in a.split(';'):aa[i.split('=')[0]] = i.split('=')[1]  
...  
>>> aa  
{'pwd': 'secret', 'database': 'master', 'uid': 'sa', 'server': 'mpilgrim'}
```

7、时间对象操作

将时间对象转换成字符串

```
>>> import datetime  
>>> datetime.datetime.now().strftime("%Y-%m-%d %H:%M")  
'2011-01-20 14:05'
```

时间大小比较

```
>>> import time  
>>> t1 = time.strptime('2011-01-20 14:05', "%Y-%m-%d %H:%M")  
>>> t2 = time.strptime('2011-01-20 16:05', "%Y-%m-%d %H:%M")  
>>> t1 > t2  
False  
>>> t1 < t2  
True
```

时间差值计算, 计算8小时前的时间

```
>>> datetime.datetime.now().strftime("%Y-%m-%d %H:%M")  
'2011-01-20 15:02'  
>>> (datetime.datetime.now() - datetime.timedelta(hours=8)).  
strftime("%Y-%m-%d %H:%M")  
'2011-01-20 07:03'
```

将字符串转换成时间对象

```
>>> endtime=datetime.datetime.strptime('20100701', "%Y%m%d")
```

```
>>> type(endtime)
<type 'datetime.datetime'>
>>> print endtime
2010-07-01 00:00:00
```

将从 1970-01-01 00:00:00 UTC 到现在的秒数，格式化输出

```
>>> import time
>>> a = 1302153828
>>> time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(a))
'2011-04-07 13:23:48'
```

8、命令行参数解析(getopt)

通常在编写一些运维脚本时，需要根据不同的条件，输入不同的命令行选项来实现不同的功能。

Python中提供的getopt模块很好的实现了命令行参数的解析，下面具体说明。请看如下程序，

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys,os,getopt
def usage():
    print '''
Usage: analyse_stock.py [options...]
Options:
-e : Exchange Name
-c : User-Defined Category Name
-f : Read stock info from file and save to db
-d : delete from db by stock code
-n : stock name
-s : stock code
-h : this help info
test.py -s haha -n "HA Ha"
'''
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'he:c:f:d:n:s:')
    except getopt.GetoptError:
        usage()
    sys.exit()
    if len(opts) == 0:
```

```

usage()
sys.exit()

for opt, arg in opts:
if opt in ('-h', '--help'):
    usage()
    sys.exit()
elif opt == '-d':
    print "del stock %s" % arg
elif opt == '-f':
    print "read file %s" % arg
elif opt == '-c':
    print "user-defined %s " % arg
elif opt == '-e':
    print "Exchange Name %s" % arg
elif opt == '-s':
    print "Stock code %s" % arg
elif opt == '-n':
    print "Stock name %s" % arg
    sys.exit()

```

9、print 格式化输出

9.1、格式化输出字符串

截取字符串输出,下面例子将只输出字符串的前3个字母

```

>>> str="abcdefg"
>>> print "%.3s" % str
abc

```

按固定宽度输出,不足使用空格补全,下面例子输出宽度为10

```

>>> str="abcdefg"
>>> print "%10s" % str
abcdefg

```

截取字符串,按照固定宽度输出

```

>>> str="abcdefg"
>>> print "%10.3s" % str
abc

```

浮点类型数据位数保留

```

>>> import fpformat
>>> a= 0.00300000000005
>>> b=fpformat.fix(a,6)

```

```
>>> print b
0.003000
对浮点数四舍五入,主要使用到round函数
>>> from decimal import *
>>> a = "2.26"
>>> b = "2.29"
>>> c = Decimal(a) - Decimal(b)
>>> print c
-0.03
>>> c / Decimal(a) * 100
Decimal('-1.327433628318584070796460177')
>>> Decimal(str(round(c / Decimal(a) * 100, 2)))
Decimal('-1.33')
```

9.2、进制转换

有些时候需要做不同进制转换，可以参考下面的例子(%x 十六进制,%d 十进制,%o 十进制)

```
>>> num = 10
>>> print "Hex = %x,Dec = %d,Oct = %o" %(num,num,num)
Hex = a,Dec = 10,Oct = 12
```

10、Python调用系统命令或者脚本

使用 `os.system()` 调用系统命令，程序中无法获得输出和返回值

```
>>> import os
>>> os.system('ls -l /proc/cpuinfo')
>>> os.system("ls -l /proc/cpuinfo")
-r--r--r-- 1 root root 0 3月 29 16:53 /proc/cpuinfo
0
```

使用 `os.popen()` 调用系统命令，程序中可以获得命令输出，但是不能得到执行的返回值

```
>>> out = os.popen("ls -l /proc/cpuinfo")
>>> print out.read()
-r--r--r-- 1 root root 0 3月 29 16:59 /proc/cpuinfo>
```


使用 `commands.getstatusoutput()` 调用系统命令，程序中可以获得命令输出和执行的返回值

```
>>> import commands
>>> commands.getstatusoutput('ls /bin/ls')
(0, '/bin/ls')
```

11、Python 捕获用户 Ctrl+C ,Ctrl+D 事件

有些时候，需要在程序中捕获用户键盘事件，比如ctrl+c退出，这样可以更好的安全退出程序

```
try:
    do_some_func()
except KeyboardInterrupt:
    print "User Press Ctrl+C,Exit"
except EOFError:
    print "User Press Ctrl+D,Exit"
```

12、Python 读写文件

一次性读入文件到列表，速度较快，适用文件比较小的情况下

```
track_file = "track_stock.conf"
fd = open(track_file)
content_list = fd.readlines()
fd.close()
for line in content_list:
    print line
```

逐行读入，速度较慢，适用没有足够内存读取整个文件(文件太大)

```
fd = open(file_path)
fd.seek(0)
title = fd.readline()
keyword = fd.readline()
uuid = fd.readline()
fd.close()
```

写文件 write 与 writelines 的区别

`Fd.write(str)` : 把str写到文件中, `write()` 并不会在str后加上一个换行符

`Fd.writelines(content)` : 把content的内容全部写到文件中, 原样写入, 不会在每行后面加上任何东西

python 中字符串大小写转换

python中字符串的大小写转换和判断字符串大小写的函数小结：

一、python字符串的大小写转换，常用的有以下几种方法：

1、对字符串中所有字符 (仅对字母有效) 的大小写转换, 有两个方法：

```
print 'just to test it'.upper() #所有字母都转换成大写  
JUST TO TEST IT
```

```
print 'JUST TO TEST IT'.lower() #所有字母都转换成小写  
just to test it
```

2、对字符串中的字符 (仅对字母有效) 部分大小写转换：

```
print 'JUST TO TEST IT'.capitalize() #字符串的首字母转换成大写，  
其余转换成小写  
Just to test it
```

```
print 'JUST TO TEST IT'.title() #字符串中所有单词的首字母转换成大写，  
其余转换成小写  
Just To Test It
```

二、判断字符串大小写函数：

```
print 'JUST TO TEST IT'.isupper()  
True
```

```
print 'JUST TO TEST IT'.islower()  
False
```

```
print 'JUST TO TEST IT'.istitle()  
False
```

zabbix自动添加删除主机的python脚本

python写的zabbix自动添加和删除主机的脚本，原理是模拟登陆zabbix

web页面中添加、删除主机的操作。如果添加或是删除多台主机，可以写多个脚本循环或是多线程来调用此脚本删除添加即可。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#zabbix的自动发现功能有时由于网络等原因自动添加主机比较慢
#或者不能添加地址段内所有的主机，基于此，写此脚本根据单IP
#和单IP应用来添加删除主机。
#在zabbix2.0.3/4上测试通过

import re,urllib,urllib2,cookielib
import MySQLdb

zabbix_server = '192.168.1.2'

class web_form:
    post_data=""#登陆提交的参数
    def __init__(self):
        '''初始化类，并建立cookies值'''
        cj = cookielib.CookieJar()
        opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
        opener.addheaders = [('User-agent', 'Mozilla/5.0 (Windows NT 6.2; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0')]
        urllib2.install_opener(opener)

    def login(self,loginurl):
        '''模拟登陆,获取认证的后的session'''
        req = urllib2.Request(loginurl,self.post_data)
        _response = urllib2.urlopen(req)
        for i in str(_response.info()).split('\n'):
            if 'Set-Cookie' in i:
                sid = i.split("=")[1].split(";")[0][16:]
                break
        _d=_response.read()
        return _d,sid
```

```

def getpagehtml(self,pageurl):
    '''获取目标网站任意一个页面的html代码'''
    req2=urllib2.Request(pageurl,self.post_data)
    _response2=urllib2.urlopen(req2)
    status = _response2.code
    _d2=_response2.read()
    return _d2

def center(sql):
    try:
        center_ip = zabbix_server
        center_user = 'root'
        center_passwd = '123456789'
        conn = MySQLdb.connect(host = center_ip,user = center_user,passwd = center_passwd,connect_timeout=10)
        cursor = conn.cursor()
        cursor.execute(sql)
        alldata = cursor.fetchall()
        cursor.close()
        conn.close()
        return alldata
    except Exception,e:
        return '0'

def operation(action,hostip,post_dir):
    boss = web_form()
    #传递一个post参数
    url = "http://%s/zabbix" % zabbix_server
    boss.post_data = urllib.urlencode({"autologin":"1","enter":"Sign in","name":"admin","password":"zabbix"})
    login,sid = boss.login("%s/index.php" % url)#首先登陆zabbix, 获取认证的sid
    if action == 'add':#添加主机
        sql = "SELECT hostid FROM zabbix.hosts WHERE host='%s'" % hostip
        hostid = center(sql)
        if len(hostid) != 0:
            print "Server %s is already in zabbix status !" % hostip
        else:
            post_form = {'form':'创建主机','form_refresh':'2','host':hostip,'interfaces[1][dns]':'','interfaces[1][interfaceid]':'1','interfaces[1][ip]':hostip,'interfaces[1][isNew]':'true','interfaces[1]

```

```

[port]':'10050','interfaces[1][type]':'1','interfaces[1]
[useip]':'1','inventory_mode':'-1','ipmi_authtype':'-1','ipm
i_password':'','ipmi_privilege':'2','ipmi_username':'','macr
os[0][macro]':'','macros[0]
[value]':'','mainInterfaces[1]':'1','newgroup':'','proxy_hos
tid':'0','save':'
存档','sid':sid,'status':'0','visiblename':hostip}
        post_form = dict(post_form,**post_dir)
        boss.post_data=urlllib.urlencode(post_form)
        add_host = boss.getpagehtml("%s/hosts.php" % url
)
        sql = "SELECT hostid FROM zabbix.hosts WHERE hos
t='%s'" % hostip
        hostid = center(sql)
        if len(hostid) != 0:
            print "Add %s in zabbix success !" % hostip
        else:
            print "Add %s in zabbix failure !" % hostip
    elif action == 'del':#删除主机
        sql = "SELECT hostid FROM zabbix.hosts WHERE host='%
s'" % hostip
        hostid = center(sql)#从zabbix服务器端数据库取出要删除ip的
hostid号
        if len(hostid) != 0:
            hostid = hostid[0][0]
            a = 'hosts[%s]' % hostid
            boss.post_data=urlllib.urlencode({'form_refresh':
'1','go':'delete','goButton':'
确认 (1)','a:hostid,'sid':sid})
            del_host = boss.getpagehtml("%s/hosts.php" % url
)
            hostid = center(sql)
            sql = "SELECT hostid FROM zabbix.hosts WHERE hos
t='%s'" % hostip
            if len(hostid) != 0:
                print "Delete %s from zabbix failure !" % ho
stip
            else:
                print "Delete %s from zabbix success !" % ho
stip
        else:
            print "Server %s not in zabbix status !" % hosti
p

```

```
def main(block,action,hostip):#根据IP的应用，确定需要添加到zabbix中的组和需要应用的模板
```

```
    if block == "adb" or block == "ldb" or block == "ccs":
        sql = "SELECT groupid,name FROM zabbix.groups WHERE name IN ('WD_ADB_LDB','Linux servers')"
```

```
        groupdata = center(sql)
        sql = "SELECT hostid,HOST FROM zabbix.hosts WHERE HOST IN ('linux_Server','MySQL_status','Linux_disk_io')"
```

```
        templates = center(sql)
        elif "_s" in block:
            sql = "SELECT groupid,name FROM zabbix.groups WHERE name IN ('Linux servers','WD_SLAVE')"
```

```
            groupdata = center(sql)
            sql = "SELECT hostid,HOST FROM zabbix.hosts WHERE HOST IN ('linux_Server','MySQL_status','Slave_Status','Linux_disk_io')"
```

```
            templates = center(sql)
        else:
            sql = "SELECT groupid,name FROM zabbix.groups WHERE name IN ('Linux servers','WD_GS')"
```

```
            groupdata = center(sql)
            sql = "SELECT hostid,HOST FROM zabbix.hosts WHERE HOST IN ('linux_Server','Linux_disk_io')"
```

```
            templates = center(sql)
            templatesdirc = {}
            groupdirc = {}
            for i in groupdata:
                groups = 'groups[%s]' % str(i[0])
                groupdirc[groups] = str(i[0])
            for i in templates:
                groups = 'templates[%s]' % str(i[0])
                templatesdirc[groups] = str(i[1])
            post_dirc = dict(groupdirc,**templatesdirc)
            operation(action,hostip,post_dirc)
```

```
if __name__=="__main__":
    block = 'adb_s'
    hostip = '6.6.6.6'
    action = 'add'
    action = 'del'
    main(block,action,hostip)
```

python字符串按固定长度拆分

将 mac 地址 更改 成 一定 格式 ， 如 mac='902B345FB021' 改为 mac='90-2B-34-5F-B0-21'，写一个小python脚本，就可以处理了。其实就是字符串按照固定长度拆分。

文件mac.txt中的mac地址如下：

50E549E32ECB

902B3413EFA6

50E549ECBA1C

902B3457B16F

1C6F65296DF9

902B34131A14

50E549E3E2F8

50E5493A2696

902B345FB021

902B34131574

实现的方法有两种，如下：

方法一：

```
A = open('mac.txt','r')  
a = A.readlines()
```



```

for aa in a:
    b = list(aa.strip())
    c=''
    for i in range(len(b)):
        if i !=0:
            if i%2 == 0:
                c=c+'-'+b[i]
            else:
                c=c+b[i]
        else:
            c=c+b[i]
    print c
A.close()

```

这种方法比较简陋，刚开始想到这个。

方法二：

```

import re
A = open('mac.txt','r')
a = A.readlines()
for aa in a:
    b=re.findall(r'.{2}',aa)
    c='-'.join(b)
    print c
A.close()

```

这种就是用python的正则表达式来实现，比较方便，执行效率比较高。

处理结果如下：

50-E5-49-E3-2E-CB

90-2B-34-13-EF-A6

50-E5-49-EC-BA-1C

90-2B-34-57-B1-6F

1C-6F-65-29-6D-F9

90-2B-34-13-1A-14

50-E5-49-E3-E2-F8

50-E5-49-3A-26-96

90-2B-34-5F-B0-21

90-2B-34-13-15-74

90-2B-34-18-43-BF

00-24-1D-0E-25-8D

python脚本清除linux cron中过期的定时计划

由于服务器需要定时的去执行一些任务，精确到月、日、时、分，这样cron文件中的条目就比较多了，单个的去删除比较麻烦，写了个脚本，去维护cron文件中的计划任务，删除已经过期的，保留需要执行的。

以apache用户的cron文件为例，脚本内容如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time

def del_cront():
    f = '/var/spool/cron/apache'
    read = open(f, 'r')
    cront = read.readlines() #读取apache用户cron文件的内容
    read = open(f, 'w')
    read.write("") #清除当前文件的内容
    for line in cront:
        if '#' not in line:
            a = ' '.join(line.strip().split()[0:4]) #截取cron中的月、日、时、分字段
            aa = str(time.localtime()[0]) + ':' + ':'.join(a.split()[::-1]) + ':00' #格式化截取到的时间字段为%Y:%m:%d:%H:%M:%S格式
            cront_time = time.mktime(time.strptime(aa, '%Y:%m:%d:%H:%M:%S')) #标准时间格式转化为时间戳
            print a, aa, cront_time
            now = time.time() #当前的时间的的时间戳
            if now < cront_time:
                read.write(line) #大于当前时间的任务回写到cron文件中，保留
            else:
                read.write(line) #注释之类的保留到文件中
```

```
read.close  
  
if __name__ == "__main__":  
    del_cront()
```

执行脚本前文件/var/spool/cron/apache中的内容如下：

```
[root@cacti soft]# vim cront_apache  
#apache  
55 08 13 03 * /usr/bin/python /home/soft/games.py 1 XWB  
00 11 13 03 * /usr/bin/python /home/soft/games.py 0 XWB  
20 13 14 03 * /usr/bin/python /home/soft/games.py 1 SSJX  
30 14 14 03 * /usr/bin/python /home/soft/games.py 0 SSJX  
25 18 15 03 * /usr/bin/python /home/soft/games.py 1 SSJX  
42 19 16 03 * /usr/bin/python /home/soft/games.py 0 SSJX
```

执行脚本截图如下：

```
[root@cacti soft]# python del_cront.py  
55 08 13 03 2013:03:13:08:55:00 1363136100.0  
00 11 13 03 2013:03:13:11:00:00 1363143600.0  
20 13 14 03 2013:03:14:13:20:00 1363238400.0  
30 14 14 03 2013:03:14:14:30:00 1363242600.0  
25 18 15 03 2013:03:15:18:25:00 1363343100.0  
42 19 16 03 2013:03:16:19:42:00 1363434120.0
```

执行完脚本文件/var/spool/cron/apache中的内容如下：

```
[root@cacti soft]# cat cront_apache
#apache
30 14 14 03 * /usr/bin/python /home/soft/games.py 0 SSJX
25 18 15 03 * /usr/bin/python /home/soft/games.py 1 SSJX
42 19 16 03 * /usr/bin/python /home/soft/games.py 0 SSJX
```

当前系统时间为:

```
[root@cacti soft]# date
Thu Mar 14 13:53:03 CST 2013
```

Python用WMI模块获取windows系统信息

Python用WMI模块获取windows系统的硬件信息：硬盘分区、使用情况，内存大小，CPU型号，当前运行的进程，自启动程序及位置，系统的版本等信息。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import wmi
import os
import sys
import platform
import time

def sys_version():
    c = wmi.WMI ()
    #获取操作系统版本
    for sys in c.Win32_OperatingSystem():
        print "Version:%s" % sys.Caption.encode("UTF8"), "Ver
num:%s" % sys.BuildNumber
        print sys.OSArchitecture.encode("UTF8") #系统是32位还
是64位的
        print sys.NumberOfProcesses #当前系统运行的进程总数

def cpu_mem():
    c = wmi.WMI ()
    #CPU类型和内存
    for processor in c.Win32_Processor():
        #print "Processor ID: %s" % processor.DeviceID
        print "Process Name: %s" % processor.Name.strip()
    for Memory in c.Win32_PhysicalMemory():
        print "Memory Capacity: %.fMB" %
(int(Memory.Capacity)/1048576)

def cpu_use():
    #5s取一次CPU的使用率
    c = wmi.WMI()
    while True:
        for cpu in c.Win32_Processor():
```

```

        timestamp = time.strftime('%a, %d %b %Y %H:%M:%S', time.localtime())
        print '%s | Utilization: %s: %d %%' % (timestamp, cpu.DeviceID, cpu.LoadPercentage)
        time.sleep(5)
        def disk():
            c = wmi.WMI ()
            #获取硬盘分区
            for physical_disk in c.Win32_DiskDrive ():
                for partition in physical_disk.associators ("Win32_DiskDriveToDiskPartition"):
                    for logical_disk in partition.associators ("Win32_LogicalDiskToPartition"):
                        print physical_disk.Caption.encode("UTF8"), partition.Caption.encode("UTF8"), logical_disk.Caption
                        #获取硬盘使用百分情况
                        for disk in c.Win32_LogicalDisk (DriveType=3):
                            print disk.Caption, "%0.2f%% free" % (100.0 * long (disk.FreeSpace) / long (disk.Size))
            def network():
                c = wmi.WMI ()
                #获取MAC和IP地址
                for interface in c.Win32_NetworkAdapterConfiguration (IPEnabled=1):
                    print "MAC: %s" % interface.MACAddress
                    for ip_address in interface.IPAddress:
                        print "ip_add: %s" % ip_address
                print
                #获取自启动程序的位置
                for s in c.Win32_StartupCommand ():
                    print "[%s] %s

```

递归遍历目录中的所有文件

python递归遍历目录中的所有文件，打印出所有文件，有两种方法：

```
import os
rootDir = 'C:\\zabbix\\'
def Test1(rootDir):
    for root,dirs,files in os.walk(rootDir):
        for filepath in files:
            print os.path.join(root,filepath)
import os
def Test2(rootDir):
    for lists in os.listdir(rootDir):
        path = os.path.join(rootDir, lists)
        print path
        if os.path.isdir(path):
            Test2(path)
```

```
----- python2.7_win -----
C:\zabbix\disk.py
C:\zabbix\zabbix_agentd.log
C:\zabbix\bin\win32\zabbix_agentd.exe
C:\zabbix\bin\win32\zabbix_get.exe
C:\zabbix\bin\win32\zabbix_sender.exe
C:\zabbix\bin\win64\zabbix - 快捷方式.lnk
C:\zabbix\bin\win64\zabbix_agentd.exe
C:\zabbix\bin\win64\zabbix_get.exe
C:\zabbix\bin\win64\zabbix_sender.exe
C:\zabbix\conf\zabbix_agentd.51CTO.com
输出完成 (耗时 0 秒) - 正常终止技术博客 Blog
```


wxPython中工具栏、状态栏、菜单实现

wxPython是python可视化编程中的一个很好的模块，以下的代码主要讲述工具栏、状态栏、菜单、菜单事件的实现（可参考：<http://www.czug.org/python/wxpythoninaction/>）：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import wx
import wx.py.images
class ToolbarFrame(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Toolbars', size
= (600, 400))
        panel = wx.Panel(self)
        panel.SetBackgroundColour('White')
        #创建状态栏
        statusBar = self.CreateStatusBar()
        #创建工具栏
        toolbar = self.CreateToolBar()
        #增加一个工具
        toolbar.AddSimpleTool(wx.NewId(), wx.py.images.getPy
Bitmap(), "New", "Long help for 'New'")
        toolbar.AddSimpleTool(wx.NewId(), wx.py.images.getPy
Bitmap(), "Edit", "Long help for 'Edit'")
        #准备显示
        toolbar.Realize()
        #创建菜单
        menuBar = wx.MenuBar()
        menu1 = wx.Menu()
        menuBar.Append(menu1, u"&文件") #菜单项目1
        self.close = menu1.Append(wx.NewId(), u"退出")
```

```

(&X)", "")
    menu2 = wx.Menu()
    # 菜单内容&表示随后的字符为热键，参数3为在状态栏上显示的菜单项
说明
    self.Copy = menu2.Append(wx.NewId(), "&Copy", "Copy
in status bar")
    self.Cut = menu2.Append(wx.NewId(), "C&ut", "")
    self.Paste = menu2.Append(wx.NewId(), "Paste", "")
    menu2.AppendSeparator()
    self.Options = menu2.Append(wx.NewId(), "&Options...
", "Display Options")
    self.Edit = menuBar.Append(menu2, "&Edit")
    self.SetMenuBar(menuBar)
    # 调用菜单下拉的退出事件
    self.Bind(wx.EVT_MENU, self.OnClose, self.close)
    def OnClose(self, event): # 退出事件
        self.Close()
if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = ToolbarFrame(parent = None, id = -1)
    frame.Show()
    app.MainLoop()

```

python paramiko模块中设置执行命令超时值

经常使用paramiko工具对几百台设备进行管理，但是由于服务器本身或是网络原因，有时返回值回不来，使程序一直等待，这个时候需要设置一个超时值。paramiko模块中执行命令代码如下：

```
stdin, stdout, stderr =  
s.exec_command(command)
```

这个地方在模块中只有一个参数，paramiko默认在这里并不能设置超时值。

paramiko本身是可以在这个地方设置超时值的，只是默认情况下没有这个选项，需要在paramiko的安装目录中修改他的源代码，使其支持，在代码中有这个接口。在设计之初没有这个超时值，开发方考虑有些命令可能执行的时间比较长，比如大文件的压缩等，超时值如果设置的话，有可能会中断命令的执行，索性留下接口，并不设置超时值。但是我们用这个模块批量的去操作多台设备的话，有时超时值是很有必要的。

修改paramiko源代码方法如下：

找到 C:\Python27\Lib\site-packages\paramiko 目录，下面有个client.py文件，文件中找到这段代码：

```
def exec_command(self, command, bufsize=-1):  
    """
```

```

    Execute a command on the SSH server. A new L{Channel} i
s opened and
    the requested command is executed. The command's input
and output
    streams are returned as python C{file}-
like objects representing
    stdin, stdout, and stderr.
    @param command: the command to execute
    @type command: str
    @param bufsize: interpreted the same way as by the built
-in C{file()} function in python
    @type bufsize: int
    @return: the stdin, stdout, and stderr of the executing
command
    @rtype: tuple(L{ChannelFile}, L{ChannelFile}, L{ChannelF
ile}))
    @raise SSHException: if the server fails to execute the
command
    """
    chan = self._transport.open_session()
    chan.exec_command(command)
    stdin = chan.makefile('wb', bufsize)
    stdout = chan.makefile('rb', bufsize)
    stderr = chan.makefile_stderr('rb', bufsize)
    return stdin, stdout, stderr

```

修改为:

```

def exec_command(self, command, bufsize=-1, timeout = None):
    """
    Execute a command on the SSH server. A new L{Channel} i
s opened and
    the requested command is executed. The command's input
and output
    streams are returned as python C{file}-
like objects representing
    stdin, stdout, and stderr.
    @param command: the command to execute
    @type command: str
    @param bufsize: interpreted the same way as by the built
-in C{file()} function in python
    @type bufsize: int
    @return: the stdin, stdout, and stderr of the executing
command
    @rtype: tuple(L{ChannelFile}, L{ChannelFile}, L{ChannelF
ile}))

```

```
@raise SSHException: if the server fails to execute the
command
"""
    chan = self._transport.open_session()
    if timeout is not None:
        chan.settimeout(timeout)
    chan.exec_command(command)
    stdin = chan.makefile('wb', bufsize)
    stdout = chan.makefile('rb', bufsize)
    stderr = chan.makefile_stderr('rb', bufsize)
    return stdin, stdout, stderr
```

主要修改了两个地方：

1 、 def exec_command(self, command, bufsize=-1, timeout = None) 定义时加一个 timeout = None;

2、在 chan = self._transport.open_session() 下面添加一个判断

```
if timeout is not None:
    chan.settimeout(timeout)
```

python wx 的wx.Frame框架属性

最近用python的wx模块写了一些窗口，其中wx.Frame是一个最重要的窗口框架，其常用的属性用法如下：

```
wx.Frame(parent,          id=-1,          title="",  
pos=wx.DefaultPosition,   size=wx.DefaultSize,  
style=wx.DEFAULT_FRAME_STYLE, name="frame")
```

框架的形状和尺寸标记

wx.FRAME_NO_TASKBAR：一个完全标准的框架，但在Windows系统和别的支持这个特性的系统下，它不显示在任务栏中。当最小化时，该框架图标化到桌面而非任务栏。

wx.FRAME_SHAPED：非矩形的框架。框架的确切形状使用SetShape()方法来设置。窗口的形状将在本章后面部分讨论。

wx.FRAME_TOOL_WINDOW：该框架的标题栏比标准的小些，通常用于包含多种工具按钮的辅助框架。在Windows操作系统下，工具窗口将不显示在任务栏中。

wx.ICONIZE：窗口初始时将被最小化显示。这个样式仅在Windows系统中起作用。

wx.MAXIMIZE：窗口初始时将被最大化显示（全屏）。这个样式仅在Windows系统中起作用。

wx.MINIMIZE：与wx.ICONIZE相同。

窗口漂浮行为的样式

`wx.FRAME_FLOAT_ON_PARENT`: 框架将漂浮在其父窗口（仅其父窗口）的上面。（很明显，要使用这个样式，框架需要有一个父窗口）。其它的框架可以遮盖这个框架。

`wx.STAY_ON_TOP`: 该框架将始终位于系统中其它框架的上面。（如果你有多个框架使用了这个样式，那么它们将相互重叠，但对于系统中其它的框架，它们仍在上面。）

装饰窗口的样式

`wx.CAPTION`: 给窗口一个标题栏。如果要放置最大化框、最小化框、系统菜单和上下文帮助，则必须包括该样式。

`wx.FRAME_EX_CONTEXTHELP`: 这是用于Windows操作系统的，它在标题栏的右角放置问号帮助图标。这个样式是与`wx.MAXIMIZE_BOX`和`wx.MINIMIZE_BOX`样式互斥的。它是一个扩展的样式，并且必须使用两步来创建，稍后说明。

`wx.FRAME_EX_METAL`: 在Mac OS X上，使用这个样式的框架有一个金属质感的外观。这是一个附加样式，必须使用`SetExtraStyle`方法来设置。

`wx.MAXIMIZE_BOX`: 在标题栏的标准位置放置一个最大化框。

`wx.MINIMIZE_BOX`: 在标题栏的标准位置放置一个最小化框。

`wx.CLOSE_BOX`: 在标题栏的标准位置放置一个关闭框。

`wx.RESIZE_BORDER`: 给框架一个标准的可以手动调整尺寸的边框。

`wx.SIMPLE_BORDER`: 给框架一个最简单的边框，不能调整尺寸，没有其它装饰。该样式与所有其它装饰样式是互斥的。

`wx.SYSTEM_MENU`: 在标题栏上放置一个系统菜单。这个系统菜单的内容与你所使用的装饰样式有关。例如，如果你使用`wx.MINIMIZE_BOX`，那么系统菜单项就有“最小化”选项。

`wx.Frame`的公共属性

`GetBackgroundColor()`

`SetBackgroundColor(wx.Color)`: 背景色是框架中没有被其子窗口部件覆盖住的那部分的颜色。你可以传递一个`wx.Color`或颜色名给设置方法。任何传递给需要颜色的`wxPython`方法的字符串，都被解释为对函数`wx.NamedColour()`的调用。

`GetId()`、`SetId(int)`: 返回或设置窗口部件的标识符。

`GetMenuBar()`、`SetMenuBar(wx.MenuBar)`: 得到或设置框架当前使用的菜单栏对象，如果没有菜单

栏，则返回None。

`GetPosition()`、`GetPositionTuple()`、`SetPosition(wx.Point)`：以一个`wx.Point`或Python元组的形式返回窗口左上角的x,y的位置。对于顶级窗口，该位置是相对于显示区域的坐标，对于子窗口，该位置是相对于父窗口的坐标。

`GetSize()`、`GetSizeTuple()`、`SetSize(wx.Size)`：C++版的`get*`或`set*`方法被覆盖。默认的`get*`或`set*`使用一个`wx.Size`对象。`GetSizeTuple()`方法以一个Python元组的形式返回尺寸。也可以参看访问该信息的另外的方法`SetDimensions()`。

`GetTitle()`、`SetTitle(String)`：得到或设置框架标题栏的字符串。

wx.Frame的方法

`Center(direction=wx.BOTH)`：框架居中（注意，非美语的拼写Centre，也被定义了的）。参数的默认值是`wx.BoTH`，在此情况下，框是在两个方向都居中的。参数的值若是`wx.HORIZONTAL`或`wx.VERTICAL`，表示在水平或垂直方向居中。

`Enable(enable=true)`：如果参数为true，则框架能够接受用户的输入。如果参数为False，则用户不能在框架中输入。相对应的方法是`Disable()`。

`GetBestSize()`：对于`wx.Frame`，它返回框架能容纳所有子窗口的最小尺寸。

`Iconize(iconize)`: 如果参数为true, 最小化该框架为一个图标 (当然, 具体的行为与系统有关)。如果参数为False, 图标化的框架恢复到正常状态。

`IsEnabled()`: 如果框架当前有效, 则返回True。

`IsFullScreen()`: 如果框架是以全屏模式显示的, 则返回True, 否则False。细节参看`ShowFullScreen`。

`IsIconized()`: 如果框架当前最小化为图标了, 则返回True, 否则False。

`IsMaximized()`: 如果框架当前是最大化状态, 则返回True, 否则False。

`IsShown()`: 如果框架当前可见, 则返回True。

`IsTopLevel()`: 对于顶级窗口部件如框架或对话框, 总是返回True, 对于其它类型的窗口部件返回False。

`Maximize(maximize)`: 如果参数为True, 最大化框架以填充屏幕 (具体的行为与系统有关)。这与敲击框架的最大化按钮所做的相同, 这通常放大框架以填充桌面, 但是任务栏和其它系统组件仍然可见。

`Refresh(erase=True, rect=None)`: 触发该框架的重绘事件。如果rect是none, 那么整个框架被重画。如果指定了一个矩形区域, 那么仅那个矩形区域被重画。如果eraseBackground为True, 那么这个窗口的北影也将被重画, 如果为False, 那么背景将不被重画。

`SetDimensions(x, y, width, height, sizeFlags=wx.SIZE_AUTO)`: 使你能够在方法调用中设置窗口的尺寸和位置。位置由参数`x`和`y`决定, 尺寸由参数`width`和`height`决定。前四个参数中, 如果有的为-1, 那么这个-1将根据参数`sizeFlags`的值作相应的解释。表8.6包含了参数`sizeFlags`的可能取值。

`Show(show=True)`: 如果参数值为`True`, 导致框架被显示。如果参数值为`False`, 导致框架被隐藏。`Show(False)`等同于`Hide()`。

`ShowFullScreen(show, style=wx.FULLSCREEN_ALL)`: 如果布尔参数是`True`, 那么框架以全屏的模式被显示——意味着框架被放大到填充整个显示区域, 包括桌面上的任务栏和其它系统组件。如果参数是`False`, 那么框架恢复到正常尺寸。`style`参数是一个位掩码。默认值`wx.FULLSCREEN_ALL`指示`wxPython`当全屏模式时隐藏所有窗口的所有样式元素。后面的这些值可以通过使用按位运算符来组合, 以取消全屏模式框架的部分装饰:
`wx.FULLSCREEN_NOBORDER`,
`wx.FULLSCREEN_NOCAPTION`, `wx.FULLSCREEN_NOMENUBAR`,
`wx.FULLSCREEN_NOSTATUSBAR`, `wx.FULLSCREEN_NOTOOLBAR`。

SetDimensions方法的尺寸标记

`wx.ALLOW_MINUS_ONE`: 一个有效的位置或尺寸。

`wx.SIZE_AUTO`: 转换为一个`wxPython`默认值。

`wx.SIZE_AUTO_HEIGHT`: 一个有效的高度，或一个wxPython默认高度。

`wx.SIZE_AUTO_WIDTH`: 一个有效的宽度，或一个wxPython默认宽度。

`wx.SIZE_USE_EXISTING`: 使用现有的尺寸。

python写报警程序中的声音实现winsound

写windows下的报警程序，有一个报警声音的实现，在python中有个winsound模块可以来实现，方法也很简单：

```
import time
import winsound
def play_music():
    winsound.PlaySound('alert', winsound.SND_ASYNC)
    time.sleep(3)
```

```
>import winsound
```

```
PlaySound(sound, flags)
```

sound是声音文件名字，该文件为wav格式的。
flags为其播放的一些参数，如：

SND_LOOP

重复地播放声音。SND_ASYNC标识也必须被用来避免堵塞。不能用 SND_MEMORY。

SND_MEMORY

提供给PlaySound()的 sound 参数是一个 WAV 文件的内存映像(memory image)，作为一个字符串。

注意：这个模块不支持从内存映像中异步播放，因此这个标识和 SND_ASYNC 的组合将挂起

RuntimeError。

SND_PURGE

停止播放所有指定声音的实例。

SND_ASYNC

立即返回，允许声音异步播放。

SND_NODEFAULT

指定的声音没有找到，不播放系统缺省的声音。

SND_NOSTOP

不中断当前播放的声音。

SND_NOWAIT

如果声音驱动忙立即返回。

MB_ICONASTERISK

播放 SystemDefault 声音。

MB_ICONEXCLAMATION

播放 SystemExclamation 声音。

MB_ICONHAND

播放 SystemHand 声音。

MB_ICONQUESTION

播放 SystemQuestion 声音。

MB_OK

播放 SystemDefault 声音。

python蜂鸣，通过python让电脑发声：

```
import winsound
```

```
winsound.Beep(37, 2000)
```

37是频率(Hz)，2000是蜂鸣持续多少毫秒(ms)。

第一个参数frequency表示分贝数，大小在37到32767之间。第二个参数是持续时间，以毫秒为单位。

Errno 9: Bad file descriptor in python socket错误处理

写了一个循环检测端口的程序，循环的次数多了，会报Errno 9: Bad file descriptor in python socket错误。程序如下：

```
def Scan_port(host,port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #socket.setdefaulttimeout(10)
    s.settimeout(10)
    t = 1
    while 1:
        try:
            s.connect((host,port))
            s.close()
            break
        except socket.error, e:
            s.close()
            if t > 3:
                print e
                break
            else:
                t = t + 1
                time.sleep(1)
```

socket连接超时时间为10s，超时之后重试3次，3次依然连接不到或是超时，则报错。这个函数加入到多线程，多个主机时就会报Errno 9: Bad file descriptor in python socket错误。经查，是s.close()在循环里面是主线问题的主要原因，不管s.close()在循环中的哪个位置都会出现这个错误。修改了这段代码如下：


```
def Scan_port(host,port):  
    t = 1  
    while 1:  
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM  
)  
        #socket.setdefaulttimeout(10)  
        s.settimeout(10)  
        try:  
            s.connect((host,port))  
            s.close()  
            break  
        except socket.error, e:  
            s.close()  
            if t > 3:  
                print e  
                break  
            else:  
                t = t + 1  
                time.sleep(1)
```

把初始化socket提进循环中，经测试2000台服务器的端口不再出错，运行测试了一天没有问题。

python语言的基础规范

本节主要讲述python的基础：python脚本的规范、缩进、编写功能函数时注意事项等，这些都是笔者自己编程过程中的心得体会。

一、python脚本的规范:

每个脚本都有自己的规范，以下的规范不是强制的，但可以使你的脚本规范、易懂、方便使用。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

这个写在开头，定义脚本编码。现在多数都是UTF8格式，所以写脚本尽量用这个编码，遇到中文可以做编码处理，字符串编码处理主要就是encode和decode。

```
import os,urllib,MySQLdb,time,platform
```

导入需要的模块。

```
main():
    pass
```

定义函数

```
if __name__ == "__main__":
    main()
```

这个就是说脚本从这里往下执行，如果是其他的脚本调用这个脚本，这个脚本不至于执行其他的部分。

提示：以上是整个脚本中的规范，大家在写脚本的时候尽量这么做。

二、python的缩进

python的对缩进要求很严格，缩进不对，就会报语法错误；python中缩进就是一个tab键或是4个空格，4个空格比较麻烦，直接一个tab键简单，所以没有特别的需求，缩进一般用tab键。缩进类似于分层，同一缩进就是相同的层次。见如下实例：

```
if a==0:
    print a

else:
    print b
```

三、每一个功能对应一个函数

这一点最重要，每一个功能就写一个函数，脚本清晰易懂，其他复用这个功能也方便，脚本也不冗余。不建议一个函数里面有好多功能，使函数模块化。

四、系统命令的引用

引用系统命令的时候，特别是linux命令，一定要写命令的全路径，比如：

```
os.popen("/sbin/ifconfig eth0").read()
```

如果写成如下命令：

```
os.popen("ifconfig eth0").read()
```

这样也是没有问题的，手动执行脚本时会执行，但是脚本做cron的时候，就不会执行了。所以要特别注意要写命令的全路径。

五、异常处理

```
try:  
    pass  
except Exception,e:  
    print e
```

其中e就是错误信息。try的异常处理这么写就足够了，还有其他的方法，不常用。

下例是一个获取本地ip地址，从数据库查询ip的用途，去连接一个URL，判断这个URL是否可以用，并写日志。主要讲python操作数据库的常用用法。

```
#!/usr/bin/env python  
# -*- coding: utf-8 -*-  
import os,urllib,MySQLdb,time,platform  
def log_w(text):  
    logfile = "/tmp/websocket.log"  
    if os.path.isfile(logfile):  
        if (os.path.getsize(logfile)/1024/1024) > 100:  
            os.remove(logfile)
```

```

    now = time.strftime("%Y-%m-%d %H:%M:%S")
    tt = str(now) + "\t" + str(text) + "\n"
    f = open(logfile, 'a+')
    f.write(tt)
    f.close()
def get_idcname(ip):
    try:
        conn = MySQLdb.connect(host = '192.168.8.43', port=3306, user = 'read_app', passwd = '123456', charset='utf8', connect_timeout=20)
        cursor = conn.cursor() # 查询出的结果是元组形式，元组和列表基本一样。
        # cursor = conn.cursor(cursorclass = MySQLdb.cursors.DictCursor) # 查询结果是字典形式。
        sql = "select host, user from mysql.user where host='%s'" % ip # python中执行sql语句一次只能是一个sql语句，一次只执行一条，如果用分号分开写多条的话是会报错的，如果是多条sql语句可以多写几个sql和cursor.execute()来分开执行。
        cursor.execute(sql) # 执行sql语句。
        # cursor.executemany("""insert into dist_sniffer.sniffer_order_day values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s) """ , values) # 执行组合插入数据库的时候可以用这个，每个%s代表一个数据库字段，values是一个元组或是一个列表。
        alldata = cursor.fetchall() # 接收sql执行结果，如果是写操作的，这个就不用了。
        # conn.commit() 如果是写操作，需要这个去提交。
        cursor.close()
        conn.close() # 关闭数据库回话。
        return alldata[0][0].encode('UTF8') # 如果是写操作的话就没有返回值了。
    except Exception, e:
        return 0
def get_ip():
    os = platform.system()
    if os == "Linux":
        ip = os.popen("/sbin/ifconfig eth0 | grep 'inet addr'").read().strip().split(":")[1].split()[0]
    elif os == "Windows":
        import wmi
        c = wmi.WMI()
        network = c.Win32_NetworkAdapterConfiguration (IPEnabled=1)
        for interface in network:
            if interface.DefaultIPGateway:
                ip = interface.IPAddress[0]

```

```

        return ip
        #print interface.IPAddress[0],interface.MACAd
ddress,interface.IPSubnet[0],interface.DefaultIPGateway[0],i
nterface.DNSServerSearchOrder[0],interface.DNSServerSearchOr
der[1]
        #获取出网的ip地址、MAC地址、子网掩码、默认网关、DNS
def web_status():
    ip = get_ip()
    idc_name = get_idcname(ip)
    url = "http://www.text.com/index.php?idc_ip=%s&idc_name=%s" % (ip,idc_name)
    get = urllib.urlopen(url)
    if get.getcode() == 200:
        aa = int(get.read().strip())
        if aa == 1:
            text = "Webservice return OK"
        else:
            text = "Webservice return Error"
    else:
        text = "Conect webservice Error"
    print text
    log_w(text)
if __name__ == "__main__":
    web_status()

```

一开始就要养成一个好习惯，这样对以后python编程是十分有益的。

python中用string.maketrans和translate巧妙替换字符串

python中用string.maketrans和translate巧妙替换字符串

将nginx日志中字符串 [2013-07-03T00:29:40-05:00] HTTP 格式化为: "2013-07-03 00:29:40-05:00"

整条日志如下:

```
92.82.22.46 - - [2013-07-03T00:29:40-05:00] "GET /images/mask_bg.png HTTP/1.1" 200 195 "http://www.chlinux.net/" "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)" "-"
```

将 [2013-07-03T00:29:40-05:00] 替换成为: "2013-07-03 00:29:40-05:00"

把[]换成",然后把T替换成空格

做法如下:

```
>>> s='''92.82.22.46 - - [2013-07-03T00:29:40-05:00] "GET /images/mask_bg.png HTTP/1.1" 200 195 "http://www.chlinux.net/" "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)" "-"'
>>> table = string.maketrans('[]',', ')
>>> s.translate(table)
'92.82.22.46 - - "2013-07-03T00:29:40-05:00" "GET /images/mask_bg.png HTTP/1.1" 200 195 "http://www.chlinux.net/" "Mozilla/5.0 (compatible; MSIE 10.0; Windows
```



```

NT 6.1; WOW64; Trident/6.0)" "-"'
>>> s.translate(table).replace('T', ' ',1)#替换掉第一个T为空格
'92.82.22.46      -      -      "2013-07-03      00:29:40-
05:00" "GET /images/mask_bg.png HTTP/1.1" 200 195 "http://ww
w.chlinux.net/" "Mozilla/5.0 (compatible; MSIE 10.0; Windows
NT 6.1; WOW64; Trident/6.0)" "-"'

```

也可以这样:

```

>>> table = re.sub('\[|\]', '', s).replace('T', ' ',1)
>>> print table
'92.82.22.46      -      -      "2013-07-03      00:29:40-
05:00" "GET /images/mask_bg.png HTTP/1.1" 200 195 "http://w
ww.chlinux.net/" "Mozilla/5.0 (compatible; MSIE 10.0; Window
s NT 6.1; WOW64; Trident/6.0)" "-"'

```

python计算文件的行数和读取某一行内容的实现方法

一、计算文件的行数

最简单的办法是把文件读入一个大的列表中, 然后统计列表的长度。如果文件的路径是以参数的形式 filepath 传递的, 那么只用一行代码就可以完成我们的需求了, 如下:

```
count = len(open(filepath, 'rU').readlines())
```

如果是非常大的文件, 上面的方法可能很慢, 甚至失效。此时, 可以使用循环来处理:

```
count = -1
for count, line in enumerate(open(thefilepath, 'rU')):
    pass
count += 1
```

另外一种处理大文件比较快的方法是统计文件中换行符的个数 '\n' (或者包含 '\n' 的字串, 如在 windows 系统中):

```
count = 0
thefile = open(thefilepath, 'rb')
while True:
    buffer = thefile.read(8192*1024)
    if not buffer:
        break
```

```
count += buffer.count('\n')  
thefile.close( )
```

参数'rb'是必须的, 否则在windows系统中, 上面的代码会非常慢。

linecache是专门支持读取大文件, 而且支持行式读取的函数库。 linecache预先把文件读入缓存起来, 后面再访问该文件时就不再从硬盘读取。

二、读取文件某一行的内容（测试过1G大小的文件，效率还可以）

```
import linecache  
count = linecache.getline(filename, linenum)
```

三、用linecache读取文件内容（测试过1G大小的文件，效率还可以）

```
str = linecache.getlines(filename)
```

str为列表形式，每一行为列表中的一个元素

python linecache模块读取文件用法详解

linecache模块允许从任何文件里得到任何的行，并且使用缓存进行优化，常见的情况是从单个文件读取多行。

```
linecache.getlines(filename)
```

从名为filename的文件中得到全部内容，输出为列表格式，以文件每行为列表中的一个元素，并以linenum-1为元素在列表中的位置存储。

```
linecache.getline(filename, lineno)
```

从名为filename的文件中得到第lineno行。这个函数从不会抛出一个异常-产生错误时它将返回”（换行符将包含在找到的行里）。

如果文件没有找到，这个函数将会在sys.path搜索。

```
linecache.clearcache()
```

清除缓存。如果你不再需要先前从getline()中得到的行。

```
linecache.checkcache(filename)
```

检查缓存的有效性。如果在缓存中的文件在硬盘上发生了变化，并且你需要更新版本，使用这个函数。如果省略filename，将检查缓存里的所有条目。

```
linecache.updatecache(filename)
```

更新文件名为filename的缓存。如果filename文件更新了，使用这个函数可以更新linecache.getlines(filename)返回的列表。

用法举例：

```
# cat a.txt
```

```
1a
```

```
2b
```

```
3c
```

```
4d
```

```
5e
```

```
6f
```

```
7g
```

1、获取a.txt文件的内容

```
>>> a=linecache.getlines('a.txt')
>>> a
['1a\n', '2b\n', '3c\n', '4d\n', '5e\n', '6f\n', '7g\n']
```

2、获取a.txt文件中第1-4行的内容

```
>>> a=linecache.getlines('a.txt')[0:4]
>>> a
['1a\n', '2b\n', '3c\n', '4d\n']
```

3、获取a.txt文件中第4行的内容

```
>>> a=linecache.getline('a.txt',4)
>>> a
'4d\n'
```

注意：使用`linecache.getlines('a.txt')`打开文件的内容之后，如果`a.txt`文件发生了改变，当再次用`linecache.getlines`获取的内容，不是文件的最新内容，而是之前的内容，此时有两种方法：

1、使用`linecache.checkcache(filename)`来更新文件在硬盘上的缓存，然后再执行`linecache.getlines('a.txt')`就可以获取到`a.txt`的最新内容；

2、直接使用`linecache.updatecache('a.txt')`，即可获取最新的`a.txt`的内容

另：在读取文件之后，如果不需要继续使用文件的缓存时，需要最后清理一下缓存，使`linecache.clearcache()`释放缓存。

这个模块是使用内存来缓存文件内容，所以需要耗费内存，打开文件的大小和打开速度与内存大小有关系。

python调用zabbix的api接口添加主机、查询组、主机、模板

zabbix有一个API接口，可以调用这些接口来自动添加主机，查询zabbix中监控的主机、监控的模板、监控的主机组等信息，使用也非常的方便。以下是用python调用zabbix的API接口来实现上述功能：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import json
import urllib2
import sys
class zabbixtools:
    def __init__(self):
        self.url = "http://192.168.100.200/zabbix/api_jsonrpc.php"
        self.header = {"Content-Type": "application/json"}
        self.authID = self.user_login()
    def user_login(self):
        data = json.dumps(
            {
                "jsonrpc": "2.0",
                "method": "user.login",
                "params": {
                    "user": "Admin",
                    "password": "zabbix"
                },
                "id": 0
            })
        request = urllib2.Request(self.url,data)
        for key in self.header:
            request.add_header(key,self.header[key])
        try:
            result = urllib2.urlopen(request)
        except URLError as e:
            print "Auth Failed, Please Check Your Name And Password:",e.code
```

```

        else:
            response = json.loads(result.read())
            result.close()
            authID = response['result']
            return authID
        def get_data(self,data,hostip=""):
            request = urllib2.Request(self.url,data)
            for key in self.header:
                request.add_header(key,self.header[key])
            try:
                result = urllib2.urlopen(request)
            except URLError as e:
                if hasattr(e, 'reason'):
                    print 'We failed to reach a server.'
                    print 'Reason: ', e.reason
                elif hasattr(e, 'code'):
                    print 'The server could not fulfill the request.'
                    print 'Error code: ', e.code
                return 0
            else:
                response = json.loads(result.read())
                result.close()
                return response
        def host_get(self,hostip):
            #hostip = raw_input("\033[1;35;40m%s\033[0m" % 'Enter Your Check Host:Host_ip :')
            data = json.dumps(
                {
                    "jsonrpc": "2.0",
                    "method": "host.get",
                    "params": {
                        "output":
["hostid","name","status","host"],
                        "filter": {"host": [hostip]}
                    },
                    "auth": self.authID,
                    "id": 1
                })
            res = self.get_data(data)['result']
            if (res != 0) and (len(res) != 0):
                #for host in res:
                host = res[0]
                if host['status'] == '1':
                    print "\t","\033[1;31;40m%s\033[0m" % "Host_"

```

```

IP:", "\033[1;31;40m%s\033[0m" % host['host'].ljust(15), '\t',
"\033[1;31;40m%s\033[0m" % "Host_Name:", "\033[1;31;40m%s\033[0m" % host['name'].encode('GBK'), '\t', "\033[1;31;40m%s\033[0m" % u'未在监控状态'.encode('GBK')
        return host['hostid']
    elif host['status'] == '0':
        print "\t", "\033[1;32;40m%s\033[0m" % "Host_IP:", "\033[1;32;40m%s\033[0m" % host['host'].ljust(15), '\t',
"\033[1;32;40m%s\033[0m" % "Host_Name:", "\033[1;32;40m%s\033[0m" % host['name'].encode('GBK'), '\t', "\033[1;32;40m%s\033[0m" % u'在监控状态'.encode('GBK')
        return host['hostid']
    print
    else:
        print '\t', "\033[1;31;40m%s\033[0m" % "Get Host Error or cannot find this host, please check !"
        return 0
    def host_del(self):
        hostip = raw_input("\033[1;35;40m%s\033[0m" % 'Enter Your Check Host:Host_ip :')
        hostid = self.host_get(hostip)
        if hostid == 0:
            print '\t', "\033[1;31;40m%s\033[0m" % "This host cannot find in zabbix, please check it !"
            sys.exit()
        data = json.dumps(
            {
                "jsonrpc": "2.0",
                "method": "host.delete",
                "params": [{"hostid": hostid}],
                "auth": self.authID,
                "id": 1
            })
        res = self.get_data(data)['result']
        if 'hostids' in res.keys():
            print "\t", "\033[1;32;40m%s\033[0m" % "Delet Host:%s success !" % hostip
        else:
            print "\t", "\033[1;31;40m%s\033[0m" % "Delet Host:%s failure !" % hostip
    def hostgroup_get(self):
        data = json.dumps(
            {
                "jsonrpc": "2.0",
                "method": "hostgroup.get",

```

```

        "params": {
            "output": "extend",
        },
        "auth": self.authID,
        "id": 1,
    })
    res = self.get_data(data)
    if 'result' in res.keys():
        res = res['result']
        if (res !=0) or (len(res) != 0):
            print "\033[1;32;40m%s\033[0m" % "Number Of
Group: ", "\033[1;31;40m%d\033[0m" % len(res)
            for host in res:
                print "\t", "HostGroup_id:", host['groupid
'], "\t", "HostGroup_Name:", host['name'].encode('GBK')
            print
        else:
            print "Get HostGroup Error, please check !"
    def template_get(self):
        data = json.dumps(
            {
                "jsonrpc": "2.0",
                "method": "template.get",
                "params": {
                    "output": "extend",
                },
                "auth": self.authID,
                "id": 1,
            })
        res = self.get_data(data)#[['result']]
        if 'result' in res.keys():
            res = res['result']
            if (res !=0) or (len(res) != 0):
                print "\033[1;32;40m%s\033[0m" % "Number Of
Template: ", "\033[1;31;40m%d\033[0m" % len(res)
                for host in res:
                    print "\t", "Template_id:", host['template
id'], "\t", "Template_Name:", host['name'].encode('GBK')
                print
            else:
                print "Get Template Error, please check !"
    def host_create(self):
        hostip = raw_input("\033[1;35;40m%s\033[0m" % 'Enter
your:Host_ip :')
        groupid = raw_input("\033[1;35;40m%s\033[0m" % 'Ente

```

```

r your:Group_id :')
    templateid = raw_input("\033[1;35;40m%s\033[0m" % 'Enter your:Template_id :')
    g_list=[]
    t_list=[]
    for i in groupid.split(','):
        var = {}
        var['groupid'] = i
        g_list.append(var)
    for i in templateid.split(','):
        var = {}
        var['templateid'] = i
        t_list.append(var)
    if hostip and groupid and templateid:
        data = json.dumps(
            {
                "jsonrpc": "2.0",
                "method": "host.create",
                "params": {
                    "host": hostip,
                    "interfaces": [
                        {
                            "type": 1,
                            "main": 1,
                            "useip": 1,
                            "ip": hostip,
                            "dns": "",
                            "port": "10050"
                        }
                    ],
                    "groups": g_list,
                    "templates": t_list,
                },
                "auth": self.authID,
                "id": 1,
            })
        res = self.get_data(data,hostip)
        if 'result' in res.keys():
            res = res['result']
            if 'hostids' in res.keys():
                print "\033[1;32;40m%s\033[0m" % "Create host success"
            else:
                print "\033[1;31;40m%s\033[0m" % "Create host failure: %s" % res['error']['data']

```

```

        else:
            print "\033[1;31;40m%s\033[0m" % "Enter Error: i
p or groupid or tempateid is NULL,please check it !"
def main():
    test = zabbixtools()
    #test.template_get()
    #test.hostgroup_get()
    #test.host_get()
    test.host_del()
    #test.host_create()
if __name__ == "__main__":
    main()

```

相关的材料的可以参考官方文档。这个只是一些功能模块，包含获取主机、主机组、模板、删除主机等功能，可以根据需要进行调整，实现zabbix的批量化和自动化管理。因为是在linux中运行，所以设置了输出终端的字体颜色，方便区分，如果不需要，自行删除即可。

python监控文件或目录变化

在监控一个文件或目录的变化时，如果有变化，把文件上传备份至备份主机，并且监控上传过程是否有问题等，根据此需求，编写如下脚本实现这个监控功能：

```
#!/usr/bin/env python
#coding=utf-8
#####
#
#Status wd gs/ccs sql file changed
#date:2013-08-26 王伟
#文件有变化上传至备份主机，上传之后验证文件是否正确
#
#####
import paramiko,os,sys,datetime,time,MySQLdb
from pyinotify import WatchManager, Notifier, ProcessEvent,
IN_DELETE, IN_CREATE,IN_MODIFY
'''
CREATE TABLE `wddel_log.status_sql` (
  `ip` varchar(16) NOT NULL COMMENT '机器IP',
  `tar_name` varchar(50) NOT NULL COMMENT '备份文件名字',
  `md5` varchar(50) NOT NULL COMMENT '备份文件MD5',
  `flag` int(2) NOT NULL COMMENT '0:成功;1:失败',
  `error_log` varchar(100) NOT NULL COMMENT '错误日志',
  `uptime` datetime NOT NULL COMMENT '更新时间',
  KEY `ip` (`ip`),
  KEY `uptime` (`uptime`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8'''#日志表创建脚本
GM_path='/home/asktao/'
center_hostname='192.168.1.100'
center_username='root'
center_password='123456'
center_port=63008
def log2db(ip,tar_name,md5,flag,error='0'):#删除日志入库
    try:
        tar_name = os.path.split(tar_name)[1]
        now = time.strftime("%Y-%m-%d %H:%M:%S")
```

```

        conn = MySQLdb.connect(host = '192.168.1.104',user =
'root',passwd = '1q2w3e4r',charset='utf8',connect_timeout=2
0)
        cursor = conn.cursor()
        sql = "SELECT ip FROM wddel_log.status_sql WHERE ip=
'%s'" % ip
        cursor.execute(sql)
        res = cursor.fetchall()
        if len(res)==0:
            inster_sql = "insert into wddel_log.status_sql V
ALUES('%s','%s','%s',%s,'%s','%s')" % (ip,tar_name,md5,flag,
error,now)
            cursor.execute(inster_sql)
            conn.commit()
        else:
            update_sql = "UPDATE wddel_log.status_sql SET md
5='%s',flag='%s',error_log='%s',uptime='%s' WHERE ip='%s'" %
(md5,flag,error,now,ip)
            cursor.execute(update_sql)
            conn.commit()
            cursor.close()
            conn.close()
    except Exception,e:
        print e
def find_ip():#获取本地eth0的IP地址
    ip = os.popen("/sbin/ip a|grep 'global eth0']").readlines
()[0].split()[1].split("/")[0]
    if "192.168." in ip:
        ip = os.popen("/sbin/ip a|grep 'global eth1']").readl
ines()[0].split()[1].split("/")[0]
    return ip
def md5sum(file_name):#验证sql打包文件的MD5
    if os.path.isfile(file_name):
        f = open(file_name,'rb')
        py_ver = sys.version[:3]
        if py_ver == "2.4":
            import md5 as hashlib
        else:
            import hashlib
        md5 = hashlib.md5(f.read()).hexdigest()
        f.close()
        return md5
    else:
        return 0
def center_md5(file_name):#上传至备份中心的文件的MD5

```



```

try:
    s=paramiko.SSHClient()
    s.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    s.connect(hostname = center_hostname,port=center_port,username=center_username, password=center_password)
    conm = "/usr/bin/md5sum %s" % file_name
    stdin,stdout,stderr=s.exec_command(conm)
    result = stdout.readlines()[0].split()[0].strip()
    s.close()
    return result
except Exception,e:
    return e
def back_file(ip,tar_name,tar_md5):#上传文件到备份中心
    remote_dir='/data/sql'
    file_name=os.path.join(remote_dir,os.path.split(tar_name)[1])
    try:
        t=paramiko.Transport((center_hostname,center_port))
        t.connect(username=center_username,password=center_password)
        sftp=paramiko.SFTPClient.from_transport(t)
        sftp.put(tar_name,file_name)
        t.close()
        #print "%s back_file OK" % tar_name
        os.remove(tar_name)
        remot_md5=center_md5(file_name)
        if remot_md5 == tar_md5:
            log2db(ip,tar_name,tar_md5,0)
        else:
            log2db(ip,tar_name,tar_md5,1,'remot_md5!=tar_md5')
    except Exception,e:
        #print "connect error!"
        log2db(ip,tar_name,tar_md5,1,e)
        os.remove(tar_name)
def back_sql():#执行备份
    ip = find_ip()
    tar_name = "/tmp/%s.tar.gz" % ip
    sql_conn = "/usr/bin/find %s -type f -name '*.sql'|/usr/bin/xargs /bin/tar zcvPf %s" % (GM_path,tar_name)
    sql_tar = os.popen(sql_conn).readlines()
    tar_md5 = md5sum(tar_name)

```

```

        if tar_md5 != 0:
            back_file(ip,tar_name,tar_md5)
        else:
            error_log = "%s not find" % tar_name
            log2db(ip,tar_name,tar_md5,0,error_log)
class PFilePath(ProcessEvent):#文件变化的触发
    def process_IN_CREATE(self, event):
        if os.path.splitext(event.name)[1] == ".sql":
            text = "Create file: %s " % os.path.join(event.p
ath, event.name)
            #print text
            back_sql()
    def process_IN_MODIFY(self, event):
        if os.path.splitext(event.name)[1] == ".sql":
            text = "Modify file: %s " % os.path.join(event.p
ath, event.name)
            #print text
            back_sql()
def FSMonitor():#主监控函数
    back_sql()#运行脚本先备份sql文件
    wm = WatchManager()
    mask = IN_CREATE | IN_MODIFY
    notifier = Notifier(wm, PFilePath())
    wdd = wm.add_watch(GM_path, mask, rec=True)
    print 'now starting monitor %s' % (GM_path)
    while True:
        try :
            notifier.process_events()
            if notifier.check_events():
                notifier.read_events()
        except KeyboardInterrupt:
            notifier.stop()
            break
if __name__ == "__main__":
    FSMonitor()

```

此脚本中主要用到paramiko和pyinotify模块，关于 paramiko 的讲解可以参见：
<http://wangwei007.blog.51cto.com/68019/1058726>
 一文，pyinotify的用法可以参见官方文档：

[https://github.com/seb-m/pyinotify/wiki/Events-
types](https://github.com/seb-m/pyinotify/wiki/Events-types)

python用paramiko模块上传本地目录到远程目录

python用paramiko模块默认中只可以上传文件，不能直接上传目录。所以下例用os.walk方法和paramiko结合写了一个上传目录的方法，代码如下：

```
#!/usr/bin/env python
import paramiko,datetime,os
hostname='192.168.1.100'
username='root'
password='123456'
port=22
def upload(local_dir,remote_dir):
    try:
        t=paramiko.Transport((hostname,port))
        t.connect(username=username,password=password)
        sftp=paramiko.SFTPClient.from_transport(t)
        print 'upload file start %s ' % datetime.datetime.now()
        for root,dirs,files in os.walk(local_dir):
            for filepath in files:
                local_file = os.path.join(root,filepath)
                a = local_file.replace(local_dir,'')
                remote_file = os.path.join(remote_dir,a)
                try:
                    sftp.put(local_file,remote_file)
                except Exception,e:
                    sftp.mkdir(os.path.split(remote_file)[0])
                    sftp.put(local_file,remote_file)
                print "upload %s to remote %s" % (local_file,remote_file)
            for name in dirs:
                local_path = os.path.join(root,name)
                a = local_path.replace(local_dir,'')
                remote_path = os.path.join(remote_dir,a)
                try:
                    sftp.mkdir(remote_path)
```

```
        print "mkdir path %s" % remote_path
    except Exception,e:
        print e
    print 'upload file success %s ' % datetime.datetime.
now()
    t.close()
    except Exception,e:
        print e
if __name__=='__main__':
    local_dir='/home/soft/'
    remote_dir='/tmp/aaa/'
    upload(local_dir,remote_dir)
```

经过多次测试，完美实现上传本地目录到远程目录。

python写的分析mysql binlog日志工具

本小节写了一个用来分析bin-log的小工具，找出增删改查的表，并按照操作次数降序排列，以下是代码：

```
#for mysql5.5 binlog
import os,sys
#python binlog.py binglog-0001 '2013-07-01 00:00:00' '2013-07-02 00:00:00'
def log_w(type,text):
    logfile = "%s.txt" % (type,text)
    #now = time.strftime("%Y-%m-%d %H:%M:%S")
    tt = str(text) + "\n"
    f = open(logfile,'a+')
    f.write(tt)
    f.close()
logname = sys.argv[1]
start_time = sys.argv[2]
end_time = sys.argv[3]
comn = "/usr/bin/mysqlbinlog --start-datetime='%s' --stop-datetime='%s' %s" % (start_time,end_time,logname)
aa=os.popen(comn).readlines()
mylist=[]
for a in aa:
    if ('UPDATE' in a):
        update = ' '.join(a.split()[:2])
        mylist.append(update)
    if ('INSERT INTO' in a):
        update = ' '.join(a.split()[:3]).replace("INTO ","")
        mylist.append(update)
    if ('DELETE from' in a):
        update = ' '.join(a.split()[:3]).replace("from ","")
        mylist.append(update)
mylist.sort()
bb = list(set(mylist))
bb.sort()
cc = []
for item in bb:
```

```
        cc.append([mylist.count(item), (item)])
cc.sort()
cc.reverse()
for i in cc:
    print str(i[0])+'\t'+i[1]
```

spacer.gifspacer.gif执行结果如下:

```
root@db-node12-bj02:/home/mysql-binlog# python /root/binlog.py mysql-bin.001514 '2013-09-30 09:10:00' '2013-09-30 09:15:00'
2420  UPDATE `tuanmei_orders`
1432  UPDATE `tuanmei_user`
1386  UPDATE `tuanmei_order_items`
1385  UPDATE `inventory_warehouse_company`
```

redis多实例重启脚本

redis属于单进程的服务，它主要受内存、CPU、磁盘IO（主要是做持久化）影响，如果服务器配置比较高，多核CPU、高内存的服务器，可以考虑做redis多实例。做多实例之前，首先要考虑CPU和内存的利用，在测试的时候发现，redis在QPS为6-8W左右的时候，这个redis所在的逻辑CPU核的负载就在100%左右，所以要优化CPU使用，目前一般是做网卡软中断来实现平衡这种单进程使用CPU过高的情况，不过需要网卡支持网卡软中断，效果不错。

多实例redis的管理，涉及到监控、服务的管理等，这里只介绍redis多实例的重启。以下是多实例的重启脚本：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import redis, threading, sys, socket, time, os
from multiprocessing import Pool
def port_check(host, port):
    POOL = redis.ConnectionPool(host=host, port=port)
    my_server = redis.Redis(connection_pool=POOL)
    try:
        a = my_server.ping()
        if a:
            return 1
        else:
            return 0
    except Exception, e:
        return 0
def shutdown_server(host, port):
    flag = port_check(host, port)
    if flag == 1:
        POOL = redis.ConnectionPool(host=host, port=port)
```



```

        my_server = redis.Redis(connection_pool=POOL)
        my_server.shutdown()
        while 1:
            flag = port_check(host,port)
            if flag == 0:
                print "%s:%s is Shutdown OK" % (host,port)
                break
            else:
                time.sleep(1)
        else:
            print "%s:%s is Shutdown already" % (host,port)
def start_server(host,port):
    flag = port_check(host,port)
    if flag == 0:
        start_conm = "/usr/local/bin/redis-
server /usr/local/redis/etc/redis_%s.conf" % port
        os.popen(start_conm)
        time.sleep(3)
        i = 0
        while 1:#每5s检测一次redis ping,持续一分钟,1分钟没有起来
认为启动失败
            flag = port_check(host,port)
            if flag == 1:
                print "%s:%s is Start OK" % (host,port)
                break
            else:
                if i > 12:
                    print "%s:%s is Start Error" % (host,por
t)
                    break
                else:
                    time.sleep(5)
                    i = i + 1
        else:
            print "%s:%s is Start already" % (host,port)
def restart(host,port):
    shutdown_server(host,port)
    start_server(host,port)
def main():
    server_list = ["127.0.0.1:6379","127.0.0.1:16379"]
    pool = Pool(processes=3)
    for i in server_list:
        aa = i.split(':')
        host = aa[0]
        port = int(aa[1])

```

```
        results = pool.apply_async(restart, (host,port))
    pool.close()
    pool.join()
    if results.successful():
        print 'successful'
if __name__ == "__main__":
    main()
```

计算mysql数据库目录中表文件大小并排序

监控数据库每个表的增长量时发现，在mysql中的information_schema.TABLES表中有记录表的大小，但是不准，要计算每天每个表大小不是很准确。刚好笔者的mysql是独享表空间，所以只要计算出数据目录中的表文件大小即可实现这个目的。以下代码实现了计算在独享表空间下，计算数据库中所有表的物理大小，并计算整个mysql数据库目录的大小和数据库目录所在分区的剩余空间。以下是代码：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os,time,MySQLdb
'''CREATE TABLE DBA.datasize (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `host` varchar(20) NOT NULL COMMENT '服务器IP',
  `dataname` varchar(100) NOT NULL COMMENT '数据库名字',
  `tablename` varchar(100) NOT NULL COMMENT '表名字',
  `datasize` double NOT NULL COMMENT '表大小,单位:M',
  `uptime` datetime NOT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`,`host`,`dataname`,`tablename`,`datasize`,
`uptime`),
  KEY `index_uptime` (`uptime`),
  KEY `index_tablename` (`tablename`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8''' #表结构
def log_w(text):#写日志
    logfile = "datasize.txt"
    f = open(logfile,'a+')
    text = text+'\n'
    f.write(text)
    f.close()
def log2db(size_log):#把结果写入数据库
    log_host = '192.168.100.100'
    log_user = 'wangwei'
```

```

log_pass = 'wangwei'
try:
    conn = MySQLdb.connect(host = log_host,port = 3306,u
ser = log_user,passwd = log_pass,charset='utf8',connect_time
out=20)
    cursor = conn.cursor()
    cursor.executemany("insert into DBA.datasize (`host`
,`dataname`,`tablename`,`datasize`,`uptime`) VALUES(%s,%s,%s
,%s,%s)",tuple(size_log))
    conn.commit()
    cursor.close()
    conn.close()
except Exception,e:
    print e
def main():
    uptime = time.strftime("%Y-%m-%d %H:%M:%S")
    text = "===== %s =====
===== " % uptime
    print text
    #log_w(text)
    mysqldir = "/home/mysql/"
    tables = {}
    host = '192.168.100.10'#数据库本地IP
    conm = 'du -sh %s' % mysqldir
    datasize = os.popen(conm).readlines()[0].split('\t')
[0]
    dir_list = os.listdir(mysqldir)
    for i in dir_list:
        dirname = os.path.join(mysqldir,i)
        if os.path.isdir(dirname):
            tb_list = os.listdir(dirname)
            table_list = list(set([os.path.split
ext(ii)[0] for ii in tb_list]))
            for t_name in table_list:
                t_size = 0
                for t in tb_list:
                    if t_name in t:
                        f_size = os.
path.getsize(os.path.join(dirname,t))
                        t_size = t_s
ize + f_size
                t_size = t_size/1024/1024
                if t_size != 0:
                    tables[os.path.join(
i,t_name)]=t_size

```

```

        tables = sorted(tables.iteritems(),key = lambda asd:
asd[1],reverse = True)
        size_log = []
        for i in tables:
            text = str(i[0]).ljust(70)+str(i[1])+'M'
            aa = i[0].split("/")
            res = [host,aa[0],aa[1],i[1],uptime]
            size_log.append(res)
            #log_w(text)
            print text
        text = "All DataSize :".ljust(70)+str(datasize)
        size_log.append([host,"all","all",int(datasize.split
('G')[0])*1024,uptime])
        diskfree = os.popen("df -h|grep data").readlines()
[0].split()[3]
        size_log.append([host,"disk","free",int(diskfree.spl
it('G')[0])*1024,uptime])
        #log_w(text)
        print text
        text = "Data Disk free size:".ljust(70)+diskfree
        #log_w(text)
        print text
        log2db(size_log)
if __name__=='__main__':
    main()

```

监控redis多实例的负载情况

单个服务器上创建多实例，对其重要参数的监控是非常重要的，以下是监控服务器上多实例的负载情况。主要包含：redis实例的QPS、内存使用情况、客户端连接数，服务器的内存使用率、CPU使用率、CPU load值、网卡流量等，脚本把采集到的数据显示并且写入到数据库中，方便查看多个服务器的多实例情况以及数据的汇总等，写的有点仓促，有兴趣的同学可以整理一下脚本使其模块化、简洁化。脚本如下：

```
#!/usr/bin/env python
#-*-coding:UTF-8-*-
import os,threading,time,sys,sigar,MySQLdb
'''
安装python的sigar模块
apt-get install libtool automake gettext python-
MySQLdb screen pkg-config git
git clone git://github.com/hyperic/sigar.git sigar.git
./autogen.sh
./configure
make
make install
cd bindings/python/
python setup.py install
'''
'''
建库sql
CREATE TABLE `redis_stats` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `host_name` varchar(50) NOT NULL,
  `qps` int(11) NOT NULL,
  `clients` int(11) NOT NULL,
  `redis_mem_used` varchar(50) NOT NULL,
  `sys_mem_used_pers` float NOT NULL,
  `cpu_used` float NOT NULL,
  `cpu_load` varchar(50) NOT NULL,
  `netband` varchar(255) NOT NULL,
```

```

`uptime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UP
DATE CURRENT_TIMESTAMP,
PRIMARY KEY (`id`),
KEY `host_name` (`host_name`),
KEY `uptime` (`uptime`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
查询每个实例的最新记录
select host_name,qps,clients,redis_mem_used,concat(sys_mem_us
ed_pers,'%') as sys_mem_used,concat(cpu_used,'%') as cpu_used
,cpu_load,netband from DBA.redis_stats group by host_name des
c order by qps desc;
'''
def log2db(check_log):
    log_host = '192.168.56.101'
    log_user = 'root'
    log_pass = '1q2w3e4r'
    try:
        conn = MySQLdb.connect(host = log_host,port = 3306,us
er = log_user,passwd = log_pass,charset='utf8',connect_timeou
t=20)
        cursor = conn.cursor()
        #cursor.execute(insert_sql)
        cursor.executemany("INSERT INTO redis.redis_stats (`h
ost_name`,`qps`,`clients`,`redis_mem_used`,`sys_mem_used_pers
`,`cpu_used`,`cpu_load`,`netband`) VALUES (%s,%s,%s,%s,%s,%s,
%s,%s)",tuple(check_log))
        conn.commit()
        cursor.close()
        conn.close()
    except Exception,e:
        print e
def redis_info(host,port,res):
    var = []
    var.append(host)
    var.append(port)
    aaa = os.popen("redis-cli -h %s -p %s info|grep -
v '#'|tr -s '\r\n'" % (host,port)).readlines()
    dirc = {}
    for i in aaa:
        if i != '\r\n':
            a = i.strip()
            aa = a.split(":")
            dirc[aa[0]]=aa[1]
    var.append(dirc["connected_clients"])
    var.append(dirc["instantaneous_ops_per_sec"])

```

```

    var.append(dirc["used_memory_human"])
    res.append(var)
def main():
    netband = {}
    stime = 5
    while True:
        try:
            sg = sigar.open()
            mem = sg.mem()#内存
            mem_percent = "%.2f" % mem.used_percent()
            cpu = sg.cpu()#CPU总的使用率
            cpu_idle = "%.2f" % ((1-
float(cpu.idle())/cpu.total())*100)
            loadavg = sg.loadavg()#CPU load值
            cpu_loadavg = ','.join([str(i) for i in loadavg])
            #nets = [i for i in sg.net_interface_list() if 'd
um' not in i and i != 'lo']#网卡流量统计
            nets = [i.strip() for i in os.popen("/bin/ip a|gr
ep global|awk '{print $7}'").readlines() if i.strip() != '']
            if len(netband) != 0:
                for net in nets:
                    netband[net+'_Out'] = "%.2f" % (float((sg
.net_interface_stat(net).tx_bytes()) - int(netband[net+'_Out'
]))/stime/1024/1024)
                    netband[net+'_In'] = "%.2f" % (float((sg.
net_interface_stat(net).rx_bytes()) - int(netband[net+'_In']
))/stime/1024/1024)
                else:
                    for net in nets:
                        netband[net+'_Out'] = "%.2f" % (float(sg.
net_interface_stat(net).tx_bytes())/stime/1024/1024)
                        netband[net+'_In'] = "%.2f" % (float(sg.n
et_interface_stat(net).rx_bytes())/stime/1024/1024)
                    redis_list = ['192.168.56.101:6379','192.168.1.87
:16379']
                    text = ""*20 + " Redis Status %s " % time.str
ftime("%Y-%m-%d %H:%M:%S") + ""*20
                    print "\033[1;31;40m%s\033[0m" % text
                    threads = []
                    res = []
                    for i in redis_list:
                        aa = i.split(':')
                        host = aa[0]
                        port = aa[1]
                        t = threading.Thread(target=redis_info,args=

```



```

(host,port,res))
        threads.append(t)
        for i in range(len(threads)):
            threads[i].start()
        for i in range(len(threads)):
            threads[i].join()
        print "\033[1;35;40m%s\033[0m" % ("Redis_host:port".ljust(23)+"Redis:QPS".ljust(10)+"Redis:Clients".ljust(15)+
"Redis:Mem_used".ljust(15)+"Mem_percent".ljust(12)+"Cpu_used"
.ljust(10)+"Cpu_loadavg".ljust(17))
        All_qps = 0
        All_clients = 0
        res.sort()
        check_log = []
        for i in res:
            log = [i[0]+':'+i[1],int(i[3]),int(i[2]),i[4]
,float(mem_percent),float(cpu_idle),cpu_loadavg,str(netband)]
            check_log.append(log)
            print (i[0]+':'+i[1]).ljust(23)+i[3].ljust(10)
)+i[2].ljust(15)+i[4].ljust(15)+mem_percent.ljust(12)+cpu_idl
e.ljust(10)+cpu_loadavg.ljust(17)+str(netband)
            All_qps = All_qps + int(i[3])
            All_clients = All_clients + int(i[2])
            log2db(check_log)
            print "\033[1;35;40m%s\033[0m" % ("Summary All ho
st:".ljust(23)+str(All_qps).ljust(10)+str(All_clients).ljust(
10))

            netband = {}
            for net in nets:
                netband[net+'_Out'] = sg.net_interface_stat(n
et).tx_bytes()
                netband[net+'_In'] = sg.net_interface_stat(ne
t).rx_bytes()
            time.sleep(stime)
        except KeyboardInterrupt :
            sys.exit(0)
            print
            break
if __name__ == "__main__":
    main()

```

效果如图:

```

root@debian:~/redis_status# python redis_status.py
***** Redis Status 2014-01-16 10:32:52 *****
Redis_host:port    Redis:QPS Redis:Clients Redis:Mem_used Mem_percent Cpu_used Cpu_loadavg
192.168.1.87:16379 0         2         28.10M      41.89      17.02      0.24,0.49,0.6 ('eth0_In': '0.44', 'eth0_Out': '0.01', 'eth1_In': '0.07', 'eth1_Out': '0.19')
192.168.56.101:6379 2         1         528.10M     41.89      17.02      0.24,0.49,0.6 ('eth0_In': '0.44', 'eth0_Out': '0.01', 'eth1_In': '0.07', 'eth1_Out': '0.19')
Summary All host: 2         3
***** Redis Status 2014-01-16 10:32:57 *****
Redis_host:port    Redis:QPS Redis:Clients Redis:Mem_used Mem_percent Cpu_used Cpu_loadavg
192.168.1.87:16379 0         2         28.10M      42.05      17.00      0.22,0.48,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
192.168.56.101:6379 0         1         528.10M     42.05      17.00      0.22,0.48,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
Summary All host: 0         3
***** Redis Status 2014-01-16 10:33:03 *****
Redis_host:port    Redis:QPS Redis:Clients Redis:Mem_used Mem_percent Cpu_used Cpu_loadavg
192.168.1.87:16379 0         2         28.10M      42.11      16.98      0.2,0.47,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
192.168.56.101:6379 0         1         528.10M     42.11      16.98      0.2,0.47,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
Summary All host: 0         3
***** Redis Status 2014-01-16 10:33:08 *****
Redis_host:port    Redis:QPS Redis:Clients Redis:Mem_used Mem_percent Cpu_used Cpu_loadavg
192.168.1.87:16379 0         2         28.10M      42.09      16.97      0.19,0.46,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
192.168.56.101:6379 0         1         528.10M     42.09      16.97      0.19,0.46,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
Summary All host: 0         3
***** Redis Status 2014-01-16 10:33:13 *****
Redis_host:port    Redis:QPS Redis:Clients Redis:Mem_used Mem_percent Cpu_used Cpu_loadavg
192.168.1.87:16379 0         2         28.10M      42.11      16.96      0.25,0.47,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
192.168.56.101:6379 0         1         528.10M     42.11      16.96      0.25,0.47,0.59 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
Summary All host: 0         3
***** Redis Status 2014-01-16 10:33:18 *****
Redis_host:port    Redis:QPS Redis:Clients Redis:Mem_used Mem_percent Cpu_used Cpu_loadavg
192.168.1.87:16379 0         2         28.10M      42.23      16.96      0.39,0.5,0.6 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
192.168.56.101:6379 1         1         528.10M     42.23      16.96      0.39,0.5,0.6 ('eth0_In': '0.00', 'eth0_Out': '0.00', 'eth1_In': '0.00', 'eth1_Out': '0.00')
Summary All host: 1         3

```