1.

Python多线程和多进程

对于操作系统来说，一个任务就是一个进程，例如打开浏览器，打开计算器，打开记事本等等。同一种任务打开多个，分别属于不同进程。对于一个任务来说，它有很多子任务，例如播放器，既要解码视频、也要解码音频，所以在进程下存在多线程。在一个进程下至少存在一个线程，可以称它为主线程。操作系统创建进程时，会单独为每一个进程分配各自的资源，进程与进程之间相互隔离。而进程内的线程，则共享了当前进程内的资源。操作系统可以调度的最小系统粒度为线程，而分配资源的最小粒度是进程。我们的多任务操作系统，在单核CPU上是在各个线程上不断切换而达到目的（并发），而在多核CPU上则能同时执行多个线程任务（并行）。

并发（concurrency）和并行（parallellism）的区别：

并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔发生。

并行是在不同实体上的多个事件，并发是在同一实体上的多个事件。

在Python中，我们可以使用multiprocessing模块进行多进程开发。程序执行时候，程序自身为主进程，然后 fork出多个子进程。

在Python中，我们可以使用threading进行多线程(threading)开发。

2. Python上已经开发出了许多和数据库交互的接口DB-API.不同的数据库Python需要不同的API去与其交互。大致的流程都是引入模块，获取与数据库的链接并且返回cursor, 通过cursor输入语句（增删改查），通过cursor方法获得返回的结果。

3. Python 的telnet模块在执行每一句语句之前，都必须又要read\_until. 不管上一句话能运行多快，如果没有那么下一句话python会马上直接发送，对方的server根本没那么快反应，会没法接受。

4. API与Python包的区别

API是一个成熟的系统提供给其他系统调用的接口，表现方式可以有两种。一种是一个URL地址。例如OpenStack的API，Jira的REST API。 一种是Python包的API,里面说明了API调用方式。

Python包是一个函数集合。一般利用Python包和API合用来达到目的。

例如：

我们想使用RESTAPI创建一个JIRA ISSUE，我们可以用两种方式。

第一种：我们使用PYTHON的REQEUST包，按照文档说明的格式，向RESTAPI发送请求，发送JSON格式数据，达到目的。

第二种：我们可以直接使用Python的JIRA包，按照Python JIRA包的文档说明达到目的。（其实Python JIRA包实际上便又封装了一层，在PYTHON JIRA包里面，其实也是引入了Requests模块，然后向RESTAPI发送。但是我们使用Python的JIRA包，变得更加方便，而不需要直接操作RESTAPI地址。

5. Python的telnetlib包中的timeout定义最好在每个需要等待的操作中都指定。例如telnetlib. Telnet("regsrv-sas.in.alcatel-lucent.com", port=23, timeout=10)这句话，只是让python在建立对服务器连接的时候，timeout=10. 连接上以后tn.read\_until不会承接timeout=10的属性，所以如果在tn.read\_until的时候卡住，python一样不会超市，会一直block住。所以在接下来的语句中还需要写成tn.read\_until(…,timeout=10).

P.S Perl中在构建telnet对象的时候，设置的timeout属性会被后面的操作所继承。

6. Python中如果用with去打开文件，然后在with的上下文中进行json.dump()去写入文件的内容。那需要注意，with在上下文结束以后，还要给写入的文件加入标记结束。所以不能在with的上下文中，进行cp,因为那时候在with的上下文中，要生成的文件还没有完全生成，cp过去的文件是半成品。

7.Python中使用paramiko作为SSH的通信模块。这个模块可以使用SSH去和服务器进行交互，执行命令以及获取返回值。

对服务器执行命令以及获取返回的内容为：stdin,stdout,stderr = ssh.exec\_command(command,timeout=xx).

获取返回的内容为: stdin.read(),stdout.read(), stderr.read()

8.

在python里，string object和unicode object是两种不同的类型。  
  
string object是由characters组成的sequence，而unicode object是Unicode code units组成的sequence

9.

Python文件在引入模块的时候需要注意可能发生的循环引用问题，意思是A需要B，而B又需要A。

解决的办法有：

将引入改为单方向，进行引入整合。

将引入放入某个方法当中，当引入语句放在方法当中时，则只有执行该方法才会执行该引入语句。

10.

Python的\_\_init\_\_.py的功能

11.当建立了一个\_\_init\_\_.py文件时候, Python就会认为这个目的时Python的一个包。

12.并且我们在引入这个包的时候，在\_\_init\_\_.py中的代码会自动运行，所以如果我们在\_\_init\_\_.py中加入一些import代码等，那么这些代码便会被执行。

13.在\_\_init\_\_.py中指定\_\_all\_\_这个特殊的变量，变会在以后from xx import \*引入所有的时候，引入指定\_\_all\_\_中指定的变量。

14.

使用Python解析一个文档的时候，如果需要的匹配十分复杂，我们可以不直接使用正则表达式，而是通过文件句柄，将文件的内容都入，然后逐句进行正则分析。

15.

Pycharm是由JetBrain公司开发的用于编码Python的开发集成环境，除Pycharm以外，JetBrain公司还提供其他JAVA (IntelliJ), PHP, 数据库开发等多种继承模式。需要购买证书作为使用权限。

Pycharm的offline是指当前没有连上任何的版本控制，所以无法进行提交同步代码。但是可以进行编辑等。

16.

PEP8是一个编程规范，内容是一些关于如何让你的程序更具可读性的建议。

例如包括了缩进。4个空格的缩进（编辑器都可以完成此功能），不使用Tap，更不能混合使用Tap和空格。

每行最大长度79，换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车。

类和top-level函数定义之间空两行；类中的方法定义之间空一行；函数内逻辑无关段落之间空一行；其他地方尽量不要再空行。

17.

我们用python的json模块读文件的时候，文件中定义的数据类型将影响读入的数据类型。

例如在文件中定义：

{“A”:10} 或者 {“A”: 10} （空格不影响）

那么json.load() 以后A所对应的是数字10而不是字符串10.如果是{“A”:”10”},那么json.load()以后A所对应的值是字符串‘10’而不是数字10.

18.

Python作用域的产生

就作用域而言，Python与C有着很大的区别，在Python中并不是所有的语句块中都会产生作用域。只有当变量在Module(模块)、Class(类)、def(函数)中定义的时候，才会有作用域的概念。在if-elif-else、for-else、while、try-except\try-finally等关键字的语句块中并不会产成作用域。

使用def从而产生作用域：

#!/usr/bin/env python

def func():

variable = 100

print variable

print variable

代码的输出为：

NameError: name 'variable' is not defined

使用if-else语句不产生作用域：

if True:

variable = 100

print (variable)

print ("\*\*\*\*\*\*")

print (variable)

代码的输出为：

100

\*\*\*\*\*\*

100

19.

Python的续行，如果Python中一行太长，我们根据PEP8的原则，可以将一行分成多行来写。主要有两种写法

--第一种使用 “\” 进行换行输入：

例如2+\

3

Python将输出5.

--第二种使用括号（），方括号 []来续行。

例如

test = [1,

2,3,4,

5]

print test [3]

Python输出4

又如：

with open(  
 '/samregression/generatedToolData/Vresource/Galvatron/' + regression\_server + '/vResourceTrack\_lock\_' + regression\_server + '.json',  
 'w+') as outfile:

为合法的Python语句

20.

当我们使用Python的循环去删除文件夹的时候，比较安全的做法是将删除文件夹的语句放在判断语句内，加强删除时的条件，防止误删除。

例如：

For each\_dir in dir\_list: For each dir in dir\_list:

Os.system(“rm -rf “ + each\_dir) --🡪 if “Should\_Deleted” in each\_dir:

Os.system(“rm -rf “ + each\_dir)

21.

Python的字典是无序的，在于

（1）键值的哈希碰撞，hash(key1) == hash(key2)时，向字典里连续添加的这个两个键的顺序是不可以控制的，也是无法做到连续的，后来的键会按算法调整到其它位置。  
（2）字典空间扩容，当键的数量超过字典默认开的空间时，字典会做空间扩容，扩容后的键顺和创建顺序就会发生变化，不受人为控制。

使字典有序化的输出最好的方法是，当要输出字典的时候，先取出函数,keys(),values(),items()取出内容，然后通过排序来达到输出有序化字典的目的。

另外我们可以通过Python的有序字典[OrderedDict](http://www.cnblogs.com/mjoy/tag/python%E5%88%9B%E5%BB%BA%E6%9C%89%E5%BA%8F%E5%AD%97%E5%85%B8OrderedDict/)来让字典变得有序。它背后的实现方式是将插入的key放入一个列表中，当读取的时候从这个列表中都key,达到字典可以按key的插入顺序进行输出。

22.

re.split（“正则表达式”，内容，分割次数）， 作用是根据正则表达式进行分割，并且正则表示式作为分隔符. 如果正则表达式不被括号包围，将不会放入结果当中。如果被括号包围，则加入结果当中。

例如：

m = re.split('\d+' , '123abc321cba') 🡪 不在括号当中，匹配的结果将保留

print m

print "\n"

#保留匹配项

m = re.split('(\d+)' , '123abc321cba') 🡪 在括号当中，匹配的结果将保留

print m

23.

Python 使用open打开文件进行写操作的时候，将使用缓冲机制，缓存分三种：

全缓冲 : open函数的buffering设置大于1的整数n,n为缓冲区大小，linux默认为page的大小4096 满了n 个字节才会写入磁盘 。

例如：

f=open(“demo.txt”,’w’,buffering=1)

行缓冲 : open 函数的buffering设置为1, 碰到换行就会将缓冲区的写入磁盘。

例如：

f=open(“demo.txt”,’w’,buffering=1)

无缓冲 : open 函数的buffering设置为0 有输入就写入磁盘。

例如：

f=open(“demo.txt”,’w,’,buffering=0)

24.

[python中\_\_name\_\_的使用](https://www.cnblogs.com/1204guo/p/7966461.html)

\_\_name\_\_是一个变量。前后加了双下划线是因为是因为这是系统定义的名字。普通变量不要使用此方式命名变量。  
Python有很多模块，而这些模块是可以独立运行的！这点不像C++和C的头文件。  
而且需要注意的是import的时候是要执行所import的模块的。  
\_\_name\_\_就是标识模块的名字的一个系统变量。这里分两种情况：假如当前模块是主模块（也就是调用其他模块的模块），那么此模块名字就是\_\_main\_\_，通过if判断这样就可以执行 “\_\_main\_\_:” 后面的主函数内容；假如此模块是被import的，则此模块名字为文件名字（不加后面的.py），通过if判断这样就会跳过 “\_\_mian\_\_:” 后面的内容。

经常会出现的情况是，如果我们不加判断条件，那么在引入包的时候，如果在函数内有一些定时任务绑定等，就会被绑定多次，每一个被引入都会被绑定。

25.

Python内置函数locals () 记录的所有局部变量，并且以字典形式存储。所以要判断某个变量是否申明，使用xx in locals (). keys (). 并且locals()所返回的字典是只读字典，我们无法手动的直接对其更新键，或者键值。

Python内置函数globals () 记录的所有全局变量

26.

Python字符串的count () 内置函数，可以知道某个字符出现在该字符串中几次

例如 “ABCC”.count(“C”) # 结果为2