Linux及Python语法面试题

本文档由公众号***Python*孙行者**收集整理，个人能力有限，难免有纰漏，欢迎指正RidingRoad@163.com

- Linux及Python语法面试题
  - Linux
    - 十个常用的 Linux 命令
    - Linux 的find 和 grep
    - 执行一个定时任务
    - 线上服务可能因为种种原因导致挂掉怎么办
  - Python语法
    - 大文件的读取
    - 迭代器和生成器区别
    - 线程、进程、协程
    - 装饰器
    - 谈谈你对同步异步阻塞非阻塞理解
    - GIL 对多线程的影响
    - python 中的反射
    - 什么是线程安全
    - 什么是面向对象编程
    - 面向对象有哪些技术
    - 静态方法和类方法
    - 类属性、实例属性
    - python多进程与多线程的运行机制
    - 如何提高 Python 的运行效率
    - 介绍下“消费者”和“生产者”模型。
    - python2和python3的区别
      - 性能
      - 编码
      - 语法
      - 字符串和字节串
      - 数据类型
      - 面向对象
      - 异常

模块变动
其它
什么是 pep8
变量
函数和方法
类
模块和包
关于参数
其他

Linux

十个常用的 Linux 命令

```
ls,help,cd ,more,clear,mkdir,pwd,rm,grep,find,mv,su,date
```

Linux 的find 和 grep

（1）grep 命令是一种强大的文本搜索工具，grep 搜索内容串可以是正则表达式，允许对文本文件进行模式查找。如果找到匹配模式， grep 打印包含模式的所有行。

（2）find 通常用来在特定的目录下搜索符合条件的文件，也可以用来搜索特定用户属主的文件。

执行一个定时任务

Linux 的 Crontab 执行命令:

```
sudo crontab -e
```

例：0 */1 * * * /usr/local/etc/rc.d/lighttpd restart

每一小时重启 apache

线上服务可能因为种种原因导致挂掉怎么办

Linux 下的后台进程管理利器 supervisor

每次文件修改后在 linux 执行：service supervisord restart

Python语法

大文件的读取

1.读取大于几 G 的大文件，可以利用生成器 generator

2.对可迭代对象 file，进行迭代遍历：for line in file，会自动地使用缓冲 IO（buffered IO）以及内存管理，而不必
担心任何大文件的问题。

```
with open('filename') as file:
    for line in file:
        do_things(line)
```

迭代器和生成器区别

（1）迭代器是一个更抽象的概念，需实现nex方法和iter方法返回自己本身 。对于 strings、list、dict、tuple 等这类容器对象，使用 for 循 环遍历是很方便的。在后台 for 语句对容器象调用 iter()函数，iter()是 python 的内置函数。 iter()会返回一个定义了next()方法的迭代器对象，它在容器中逐个访问容器内元素， next()也是 python 的内置函数。在没有后续元素时， next()会 抛出一个 StopIter 异常

（2）生成器（ Generator）是创建迭代器的简单而强大工具。它们写起来就像是正规的函数，只在需要返回数据时候使用 yield 语句。每次next()被调用时，生成器会返回它脱离的位置（记忆语句最后一次执行和所有数据）

（3）区别：生成器能做到迭代器的所有事 ,而且因为自动创建了 **iter**()和 next()方法 ,生成器显得特别简洁 ,而且生成器也是高效的，使用生成器表达式可以同时节省内存。当发生器终结时 ,还会自动抛出 StopIteration 异常。Yield 的用法有点像 return,但是它返回的是一个生成器。

线程、进程、协程

（1）进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,是系统进行资源分配和调度的一个独立单位。每个进程都有自己的独立内存空间，不同进程通过进程间通信来通信。由于进程比较重量，占据独立的内存，所以上下文进程间的切换开销（栈、寄存器、虚拟内存、文件句柄等）比较大，但相对比较稳定安全。

（2）线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。线程间通信主要通过共享内存，上下文切换很快，资源开销较少，但相比进程不够稳定容易丢失数据。

（3）协程是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

装饰器

（1）装饰器经常被用于有切面需求的场景，较为经典的有插入日志、性能测试、事务处理等。装饰器是解决这类问题的绝佳设计，有了装饰器，我们就可以抽离出大量函数中与函数功能本身无关的雷同代码并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能。

（2）装饰器(decorator)里引入通用功能处理：

- 引入日志
- 函数执行时间统计
- 执行函数前预备处理
- 执行函数后清理功能
- 权限校验等场景
- 缓存

```
from time import ctime, sleep


def timefun(func):
    def wrappedfunc():
        print("%s called at %s"%(func.__name__,ctime()))
```

```
        print("%s called at %s"%(func.__name__,ctime()))
        return func()
    return wrappedfunc


@timefun
def foo():
    print("I am foo")


foo()
sleep(2)
foo()
```

谈谈你对同步异步阻塞非阻塞理解

（1）所谓同步，就是在发出一个功能调用时，在没有得到结果之前，该调用就不返回。按照这个定义，其实绝大多数函数都是同步调用（例如 sin, isdigit 等）。但是一般而言，我们在说同步、异步的时候，特指那些需要其他部件协作或者需要一定时间完成的任务。最常见的例子就是 SendMessage。该函数发送一个消息给某个窗口，在对方处理完消息之前，这个函数不返回。当对方处理完毕以后，该函数才把消息处理函数所返回的RESULT值返回给调用者。

（2）异步的概念和同步相对。当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者。以 CAsycSocket 类为例（注意，CSocket 从CAsyncSocket 派生，但是其功能已经由异步转化为同步），当一个客户端通过调用 Connect 函数发出一个连接请求后，调用者线程立刻可以往下运行。当连接真正建立起来以后，socket 底层会发送一个消息通知该对象。这里提到执行部件和调用者通过三种途径返回结果：状态、通知和回调。可以使用哪一种依赖于执行件的实现，除非执行部件提供多种选择，否则不受调用者控制。如果执行部件用状态来通知，那么调用者就需要每隔一定时间检查一次，效率就很低。如果是使用通知的方式，效率则很高，因为执行部件几乎不需要做额外的操作。至于回调函数，其实和通知没太多区别。

（3）阻塞调用是指调用结果返回之前，当前线程会被挂起。函数只有在得到结果之后才会返回。有人也许会把阻塞调用和同步调用等同起来，实际上是不同的。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回而已。例如，我们在CSocket 中调用 Receive 函数，如果缓冲区中没有数据，这个函数就会一

直等待，直到有数据才返回。而此时，当前线程还会继续处理各种各样的消息。如果主窗口和调用函数在同一个线程中，除非你在特殊的界面操作函数中调用，其实主界面还是应该可以刷新。socket 接收数据的另外一个函数 recv 则是一个阻塞调用的例子。当 socket 工作在阻塞模式的时候，如果没有数据的情况下调用该函数，则当前线程就会被挂起，直到有数据为止。

（4）非阻塞和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。
对象的阻塞模式和阻塞函数调用。对象是否处于阻塞模式和函数是不是阻塞调用有很强的相关性，但是并不是一一对应的。阻塞对象上可以有非阻塞的调用方式，我们可以通过一定的 API 去轮询状态，在适当的时候调用阻塞函数，就可以避免阻塞。而对于非阻塞对象，调用特殊的函数也可以进入阻塞调用。函数 select 就是这样的一个例子。

GIL 对多线程的影响

（1）GIL 的全称是 Global Interpreter Lock(全局解释器锁)，来源是 python 设计之初的考虑，为了数据安全所做的决定。每个 CPU 在同一时间只能执行一个线程（在单核 CPU 下的多线程其实都只是并发，不是并行，并发和并行从宏观上来讲都是同时处理多路请求的概念。

（2）但并发和并行又有区别，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内发生。）

（3）在 Python 多线程下，每个线程的执行方式：

a、获取 GIL

b、执行代码直到 sleep 或者是 python 解释器将其挂起。

c、释放 GIL

可见，某个线程想要执行，必须先拿到 GIL，我们可以把 GIL 看作是“通行证”，并且在一个 python 进程中，GIL 只有一个。拿不到通行证的线程，就不允许进入 CPU 执行。

（4）GIL 的释放逻辑是当前线程遇见 IO 操作或者ticks 计数达到 100（ticks 可以看作是 Python 自身的一个计数器，专门做用于 GIL，每次释放后归零，这个计数可以通过 sys.setcheckinterval 来调整），进行释放。

（5）而每次释放 GIL 锁，线程进行锁竞争、切换线程，会消耗资源。并且由于 GIL

锁存在，python 里一个进程永远只能同时执行一个线程(拿到 GIL 的线程才能执行)。

（6）**应用场景**：IO 密集型代码(文件处理、网络爬虫等)，多线程能够有效提升效率(单线程下有 IO 操作会进行 IO 等待，造成不必要的时间浪费，而开启多线程能在线程 A 等待时，自动切换到线程 B，可以不浪费 CPU的资源，从而能提升程序执行效率)，所以多线程对 IO 密集型代码比较友好。

python 中的反射

1）反射就是通过字符串的形式，导入模块(import)；通过字符串的形式，去模块寻找指定函数，并执行。利用字符串的形式去对象（模块）中操作（查找/获取/删除/添加）成员，一种基于字符串的事件驱动

2）反射有4个内置函数分别为:getattr、hasattr、setattr、delattr 获取成员、检查成员、设置成员、删除成员

3）应用场景：基于反射实现类Web框架的路由系统

什么是线程安全

线程安全是在多线程的环境下，能够保证多个线程同时执行时程序依旧运行正确, 而且要保证对于共享的数据可以由多个线程存取，但是同一时刻只能有一个线程进行存取。多线程环境下解决资源竞争问题的办法是加锁来保证存取操作的唯一性。

什么是面向对象编程

面向对象编程是一种解决软件复用的设计和编程方法。 这种方法把软件系统中相近相似的操作逻辑和操作 应用数据、状态,以类的型式描述出来,以对象实例的形式在软件系统中复用,以达到提高软件开发效率的作用。

面向对象有哪些技术

（1）类(Class):
用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。

（2）类变量：类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。

（3）数据成员：类变量或者实例变量用于处理类及其实例对象的相关的数据。

（4）方法重写：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。

（5）实例变量：定义在方法中的变量，只作用于当前实例的类。

（6）继承：即一个派生类（derived class）继承基类（base class）的字段和方法。

继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：

一个 Dog 类型的对象派生自 Animal 类，这是模拟"是一个（is-a）"关系（例图，Dog 是一个 Animal）。

（7）实例化：创建一个类的实例，类的具体对象。

（8）方法：类中定义的函数。

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

静态方法和类方法

（1）静态方法：

需要通过修饰器 @staticmethod 来进行修饰，静态方法一般不需要定义参数。

（2）类方法：

类方法是类对象所拥有的方法，需要用修饰器 @classmethod 来标识其为类方法，对于类方法，第一个参数必须是类对象，一般以 cls 作为第一个参数（也可以用其他名称的变量作为其第一个数），能够通过实例对象和类对象去访问。

类属性、实例属性

（1）类属性：

定义在类里面但在函数外面的变量，是静态的。 类对象 所拥有的属性，它被所有类对象的实例对象所共有，在内存中只存在一个副本。对于公有的类属性，在类外可以通过类对象和实例对象访问。

（2）实例属性：定义在__init__()方法里的变量就是实例属性，这些属性只有被创建时才会被创建。

当类属性与实例属性同名时，一个实例访问这个属性时实例属性会覆盖类属性

python多进程与多线程的运行机制

问题：有什么区别？分别在什么情况下用?

（1）运行机制：

a、进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位

b、线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

（2）区别：

a、多进程稳定性好，一个子进程崩溃了，不会影响主进程以及其余进程。但是缺点是创建进程的代价非常大，因为操作系统要给每个进程分配固定的资源，并且，操作系统对进程的总数会有一定的限制，若进程过多，操作系统调度都会存在问题，会造成假死状态。

b、多线程效率较高一些，但是致命的缺点是任何一个线程崩溃都可能造成整个进程的崩溃，因为它们共享了进程的内存资源池。

（3）应用场景：

a、如果代码是 IO 密集型的，多线程。

b、如果代码是 CPU 密集型的，多进程是更好的选择——特别是所使用的机器是多核或多 CPU 的。

如何提高 Python 的运行效率

（1）使用生成器

（2）关键代码使用外部功能包：Cython、PyInline、PyPy、Pyrex

（3）针对循环的优化——尽量避免在循环中访问变量的属性

介绍下“消费者”和“生产者”模型。

生产者-消费者模型是多线程同步的经典案例。此模型中生产者向缓冲区 push 数据，消

费者从缓冲区中 pull 数据。

a、首先，生产者只需要关心“仓库”，并不需要关心具体的消费者。

b、对于消费者而言，它不需要关心具体的生产者，它只需要关心这个“仓库”中还有没有东西存在。

c、生产者生产的时候消费者不能进行“消费”，消费者消费的时候生产者不能生产，相当于一种互斥关系，即生产者和消费者一次只能有一人能访问到“仓库”。

d、“仓库”为空时不能进行消费。

e、“仓库”满时不能进行生产。

生产者消费者模型的优点:

1、解耦

假设生产者和消费者分别是两个类。如果让生产者直接调用消费者的某个方法，那么生产者对于消费者就会产生依赖（也就是耦合）。将来如果消费者的代码发生变化，可能会影响到生产者。而如果两者都依赖于某个缓冲区，两者之间不直接依赖，耦合也就相应降低了。

2、支持并发

由于生产者与消费者是两个独立的并发体，他们之间是用缓冲区作为桥梁连接，生产者只需要往缓冲区里丢数据，就可以继续生产下一个数据，而消费者只需要从缓冲区了拿数据即可，这样就不会因为彼此的处理速度而发生阻塞。

3、支持忙闲不均

如果制造数据的速度时快时慢，缓冲区的好处就体现出来了。当数据制造快的时候，消费者来不及处理，未处理的数据可以暂时存在缓冲区中。 等生产者的制造速度慢下来，消费者再慢慢处理掉。

python2和python3的区别

性能

Py3.0 运行 pystone benchmark 的速度比 Py2.5 慢 30%。Guido 认为 Py3.0 有极大的优化
空间，在字符串和整形操作上可以取得很好的优化结果。

Py3.1 性能比 Py2.5 慢 15%，还有很大的提升空间。

编码

Py3.X 源码文件默认使用 utf-8 编码，这就使得以下代码是合法的：

```
中国 = 'china'
print(中国) # china
```

语法

1）去除了<>，全部改用!=

2）去除``，全部改用 repr()

3）关键词加入 as 和 with，还有 True,False,None

4）整型除法返回浮点数，要得到整型结果，请使用//

5）加入 nonlocal 语句。使用 noclocal x 可以直接指派外围（非全局）变量

6）去除 print 语句，加入 print()函数实现相同的功能。同样的还有 exec 语句，已经改为 exec()函数

7）改变了顺序操作符的行为，例如 x<y，当 x 和 y 类型不匹配时抛出 TypeError 而不是返回随即的 bool 值

8）输入函数改变了，删除了 raw_input，用 input 代替

```
2.X: guess = int(raw_input('Enter an integer : ')) # 读取键盘输入的方法
3.X: guess = int(input('Enter an integer : '))
```

9）去除元组参数解包。不能 def(a, (b, c)):pass 这样定义函数了

10）新式的 8 进制字面量，相应地修改了 oct()函数。

11）增加了 2 进制字面量和 bin()函数

12）扩展的可迭代解包。在 Py3.X 里，a, b, *rest* = *seq* 和 rest, a = seq 都是合法的

只要求两点：

a、rest 是 list

b、对象和 seq 是可迭代的。

13）新的 super()，可以不再给 super()传参数

14）新的 metaclass 语法：

```
class Foo(*bases, **kwds):
    pass
```

15）支持 class decorator。

字符串和字节串

1）现在字符串只有 str 一种类型，但它跟 2.x 版本的 unicode 几乎一样。

2）关于字节串，请参阅“数据类型”的第 2 条目

数据类型

1）Py3.X 去除了 long 类型，现在只有一种整型——int，但它的行为就像 2.X 版本的 long

2）新增了 bytes 类型，对应于 2.X 版本的八位串，定义一个 bytes 字面量的方法如下：

str 对象和 bytes 对象可以使用.encode() (str -> bytes) or .decode() (bytes -> str)方法相互转化。

3）dict 的.keys()、.items 和.values()方法返回迭代器，而之前的 iterkeys()等函数都被废弃。同时去掉的还有dict.has_key()，用 in 替代它吧

面向对象

1）引入抽象基类（Abstraact Base Classes，ABCs）。

2）容器类和迭代器类被 ABCs 化，所以 collections 模块里的类型比 Py2.5 多了很

多。

```
import collections

print('\n'.join(dir(collections)))

# collections 模块里的类型
Callable Container Hashable ItemsView Iterable Iterator KeysView
Mapping MappingView MutableMapping MutableSequence MutableSet Na
medTuple Sequence Set Sized ValuesView __all__ __builtins__ __do
c__ __file__ __name__ _abcoll _itemgetter _sys defaultdict deque
```

3）迭代器的 next()方法改名为__next__()，并增加内置函数 next()，用以调用迭代器的__next__()方法

4）增加了@abstractmethod 和 @abstractproperty 两个 decorator，编写抽象方法（属性）更加方便。

异常

1）所有异常都从 BaseException 继承，并删除了 StardardError

2）去除了异常类的序列行为和.message 属性

3）用 raise Exception(args)代替 raise Exception, args 语法

4）捕获异常的语法改变，引入了 as 关键字来标识异常实例

5）异常链，因为__context__在 3.0a1 版本中没有实现

模块变动

1）移除了 cPickle 模块，可以使用 pickle 模块代替。最终我们将会有一个透明高效的模块。

2）移除了 imageop 模块

3）移除了 audiodev, Bastion, bsddb185, exceptions, linuxaudiodev, md5, MimeWriter,
mimify, popen2,rexec, sets, sha, stringold, strop, sunaudiodev, timing 和 xmllib 模块

4）移除了 bsddb 模块(单独发布，可以从 http://www.jcea.es/programacion/pybsddb.htm 获取)

5）移除了 new 模块

6）os.tmpnam()和 os.tmpfile()函数被移动到 tmpfile 模块下

7）tokenize 模块现在使用 bytes 工作。主要的入口点不再是 generate_tokens，而是 tokenize.tokenize()

其它

1）xrange() 改名为 range()，要想使用 range()获得一个 list，必须显式调用：

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

2）bytes 对象不能 hash，也不支持 b.lower()、b.strip()和 b.split()方法，但对于后两者可以使用 b.strip(b' \n\t\r \f')和 b.split(b' ')来达到相同目的

3）zip()、map()和 filter()都返回迭代器。而 apply()、 callable()、coerce()、execfile()和 reload ()函数都被去除了,reduce()放在了functools模块下。现在可以使用 hasattr()来替换 callable(). hasattr()的语法如：

```
1 hasattr(string, '__name__')
```

4）string.letters 和相关的.lowercase 和.uppercase 被去除，请改用 string.ascii_letters等

5）如果 x < y 的类型不同就不能比较，否则抛出 TypeError 异常。2.x 版本是返回伪随机布尔值的

6）__getslice__系列成员被废弃。

7）file 类被废弃

什么是 pep8

变量

常量:大写加下划线 USER_CONSTANT

私有变量 : 小写和一个前导下划线 _private_value

Python 中不存在私有变量一说，若是遇到需要保护的变量，使用小写和一个前导下划线。但这只是程序员之间的一个约定，用于警告说明这是一个私有变量，外部类不要去访问它。但实际上，外部类还是可以访问到这个变量。

内置变量 : 小写，两个前导下划线和两个后置下划线 __class__两个前导下划线会导致变量在解释期间被更名。这是为了避免内置变量和其他变量产生冲突。用户定义的变量要严格避免这种风格。以免导致混乱。

函数和方法

总体而言应该使用，**小写和下划线**。但有些比较老的库使用的是混合大小写，即首单词小写，之后每个单词第一个字母大写，其余小写。但现在，小写和下划线已成为规范。

私有方法：小写和一个前导下划线

```
def _secrete(self):
    print "don't test me."
```

这里和私有变量一样，并不是真正的私有访问权限。同时也应该注意一般函数不要使用两个前导下划线(当遇到两个前导下划线时，Python 的名称改编特性将发挥作用)。

特殊方法 ： 小写和两个前导下划线，两个后置下划线

```
def __add__(self, other):
    return int.__add__(other)
```

这种风格只应用于特殊函数，比如操作符重载等。

函数参数: 小写和下划线，缺省值等号两边无空格

类

类总是使用驼峰格式命名，即所有单词首字母大写其余字母小写。类名应该简明，精确，并足以从中理解类所完成的工作。常见的一个方法是使用表示其类型或者特性的后缀，例如:

SQLEngine，MimeTypes

对于基类而言，可以使用一个 Base 或者 Abstract前缀 BaseCookie，AbstractGroup

模块和包

除特殊模块 **init** 之外，模块名称都使用不带下划线的小写字母。

若是它们实现一个协议，那么通常使用 lib 为后缀，例如:

```
import smtplib
import os
import sys
```

关于参数

1 不要用断言来实现静态类型检测。

断言可以用于检查参数，但不应进行静态类型检测。 Python 是动态类型语言，静态类型检测违背了其设计思想。断言应该用于避免函数不被毫无意义的调用。

2 不要滥用 *args 和 **kwargs。

*args 和 **kwargs 参数可能会破坏函数的健壮性。它们使签名变得模糊，而且代码常常开始在不应该的地方构建小的参数解析器。

其他

1 使用 has 或 is 前缀命名布尔元素

```
is_connect = True
has_member = False
```

2 用复数形式命名序列

```
members = ['user_1', 'user_2']
```

3 用显式名称命名字典

```
person_address = {'user_1':'10 road WD', 'user_2' : '20 street huafu'}
```

4 避免通用名称

诸如 list, dict, sequence 或者 element 这样的名称应该避免。

5 避免现有名称

诸如 os, sys 这种系统已经存在的名称应该避免。

6 一些数字

一行列数 : PEP 8 规定为 79 列。根据自己的情况，比如不要超过满屏时编辑器的显示列数。

一个函数 : 不要超过 30 行代码, 即可显示在一个屏幕类，可以不使用垂直游标即可看到整个函数。

一个类 : 不要超过 200 行代码，不要有超过 10 个方法。一个模块不要超过 500 行。

7 验证脚本

可以安装一个 pep8 脚本用于验证你的代码风格是否符合 PEP8。