目录

Python 模拟面试技术面试题答案

一、python 语法

1. 请说一下你对迭代器和生成器的区别？

答：（1）迭代器是一个更抽象的概念，任何对象，如果它的类有 next 方法和 iter 方法返回自己本身。对于 string、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 语句对容器对象调用 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 StopIteration 异常

（2）生成器（Generator）是创建迭代器的简单而强大的工具。它们写起来就像是正规的函数，只是在需要返回数据的时候使用 yield 语句。每次 next()被调用时，生成器会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）

区别：生成器能做到迭代器能做的所有事,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,使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当发生器终结时,还会自动抛出 StopIteration 异常

2. 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。

3.你所遵循的代码规范是什么？请举例说明其要求？

PEP8

1 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。

2 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

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

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

3 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

4 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
```

5 关于参数

5.1 不要用断言来实现静态类型检测。断言可以用于检查参数，但不应仅仅是进行静态类型检测。 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。

5.2 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。

6 其他

6.1 使用 has 或 is 前缀命名布尔元素

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

6.2 用复数形式命名序列

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

6.3 用显式名称命名字典

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

6.4 避免通用名称

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

6.5 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。

7 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。

8 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。

4. Python 中数组有哪些类型？字典可以是有序的吗？？

```
List Tuple  Dictionary
from  collections  import  OrderedDict
dic=OrderedDict()#声明有序字典
```

5. python 中 yield 的用法？

答： yield 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 回时在函数体中的位置。对生成器第 二次（或 n 次）调用跳转至该函 次）调用跳转至该函 数。

6. Python 中 pass 语句的作用是什么？

pass 语句什么也不做，一般作为占位符或者创建占位程序，pass 语句不会执行任何操作。

7. python2 和 python3 的区别?

1.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。

2.编码

Py3.X 源码文件默认使用 utf-8 编码

3. 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 进制字变量，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：rest 是 list
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。用法与函数 decorator 一样：

4. 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

5.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

6.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。

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。

7.异常

1）所以异常都从 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

8.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()

9.其它

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

>>> 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()、reduce()和 reload ()函数都被去除了现在可以使用 hasattr()来替换 callable(). hasattr()的语法如：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 的不能比较，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。a[i:j]根据上下文转换为 a.__getitem__(slice(I, j))或 __setitem__和 __delitem__调用

7）file 类被废弃

8.谈谈你对 GIL 锁对 python 多线程的影响？

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。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。）

在 Python 多线程下，每个线程的执行方式：

1、获取 GIL

2、执行代码直到 sleep 或者是 python 虚拟机将其挂起。

3、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。

在 Python2.x 里,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。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)。

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。

9.python 是如何进行内存管理的？

一、垃圾回收：python 不像 C++，Java 等语言一样，他们可以不用事先声明变量类型而直接对变量进行赋值。对 Python 语言来讲，对象的类型和内存都是在运行时确定的。这也是为什么我们称 Python 语言为动态类型的原因（这里我们把动态类型可以简单的归结为对变量内存地址的分配是在运行时自动判断变量类型并对变量进行赋值）。

二、引用计数：Python 采用了类似 Windows 内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对对象的引用的计数。当变量被绑定在一个对象上的时候，该变量的引用计数就是 1，(还有另外一些情况也会导致变量引用计数的增加),系统会自动维护这些标签，并定时扫描，当某标签的引用计数变为 0 的时候，该对就会被回收。

三、内存池机制 Python 的内存机制以金字塔行，-1，-2 层主要有操作系统进行操作，

第 0 层是 C 中的 malloc，free 等内存分配和释放函数进行操作；

第 1 层和第 2 层是内存池，有 Python 的接口函数 PyMem_Malloc 函数实现，当对象小于 256K 时有该层直接分配内存；

第 3 层是最上层，也就是我们对 Python 对象的直接操作；

在 C 中如果频繁的调用 malloc 与 free 时,是会产生性能问题的.再加上频繁的分配与释放小块的内存会产生内存碎片. Python 在这里主要干的工作有:

如果请求分配的内存在 1~256 字节之间就使用自己的内存管理系统,否则直接使用 malloc.

这里还是会调用 malloc 分配内存,但每次会分配一块大小为 256k 的大块内存.

经由内存池登记的内存到最后还是会回收到内存池,并不会调用 C 的 free 释放掉.以便下次使用.对于简单的 Python 对象，例如数值、字符串，元组（tuple 不允许被更改)采用的是复制的方式(深拷贝?)，也就是说当将另一个变量 B 赋值给变量 A 时，虽然 A 和 B 的内存空间仍然相同，但当 A 的值发生变化时，会重新给 A 分配空间，A 和 B 的地址变得不再相同

10. Python 里面如何拷贝一个对象？（赋值，浅拷贝，深拷贝的区别）

答：赋值（=），就是创建了对象的一个新的引用，修改其中任意一个变量都会影响到另一个。

浅拷贝：创建一个新的对象，但它包含的是对原始对象中包含项的引用（如果用引用的方式修改其中一个对象,另外一个也会修改改变){1,完全切片方法;2,工厂函数,如 list();3，copy 模块的 copy()函数}

深拷贝：创建一个新的对象，并且递归的复制它所包含的对象（修改其中一个，另外一个不会改变）{copy 模块的 deep.deepcopy()函数}

11. 如何用 Python 来进行查询和替换一个文本字符串？

可以使用 re 模块中的 sub()函数或者 subn()函数来进行查询和替换，

格式：sub(replacement, string[,count=0])（replacement 是被替换成的文本，string 是需要被替换的文本，count 是一个可选参数，指最大被替换的数量）

12. 递归函数停止的条件

递归的终止条件一般定义在递归函数内部，在递归调用前要做一个条件判断，根据判断的结果选择是继续调用自身，还是 return;返回终止递归。

终止的条件：1.判断递归的次数是否达到某一限定值

2. 判断运算的结果是否达到某个范围等，根据设计的目的来选择

13. Python 常用的设计模式有哪些？

1 创建型模式

前面讲过，社会化的分工越来越细，自然在软件设计方面也是如此，因此对象的创建和对象的使用分开也就成为了必然趋势。因为对象的创建会消耗掉系统的很多资源，所以单独对对象的创建进行研究，从而能够高效地创建对象就是创建型模式要探讨的问题。这里有 6 个具体的创建型模式可供研究，它们分别是：

简单工厂模式（Simple Factory）；

工厂方法模式（Factory Method）；

抽象工厂模式（Abstract Factory）；

创建者模式（Builder）；

原型模式（Prototype）；

单例模式（Singleton）。

说明：严格来说，简单工厂模式不是 GoF 总结出来的 23 种设计模式之一。

2 结构型模式

在解决了对象的创建问题之后，对象的组成以及对象之间的依赖关系就成了开发人员关注的焦点，因为如何设计对象的结构、继承和依赖关系会影响到后续程序的维护性、代码的健壮性、耦合性等。对象结构的设计很容易体现出设计人员水平的高低，这里有 7 个具体的结构型模式可供研究，它们分别是：

外观模式（Facade）；

适配器模式（Adapter）；

代理模式（Proxy）；

装饰模式（Decorator）；

桥模式（Bridge）；

组合模式（Composite）；

享元模式（Flyweight）

3 行为型模式

在对象的结构和对象的创建问题都解决了之后，就剩下对象的行为问题了，如果对象的行为设计的好，那么对象的行为就会更清晰，它们之间的协作效率就会提高，这里有 11 个具体的行为型模式可供研究，它们分别是：

模板方法模式（Template Method）；

观察者模式（Observer）；

状态模式（State）；

策略模式（Strategy）；

职责链模式（Chain of Responsibility）；

命令模式（Command）；

访问者模式（Visitor）；

调停者模式（Mediator）；

备忘录模式（Memento）；

迭代器模式（Iterator）；

解释器模式（Interpreter）。

14. 单例的应用场景有哪些？

单例模式应用的场景一般发现在以下条件下：

（1）资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如日志文件，应用配置。

（2）控制资源的情况下，方便资源之间的互相通信。如线程池等。

1.网站的计数器

2.应用配置

3.多线程池

4.数据库配置，数据库连接池

5.应用程序的日志应用....

15. 解释一下什么是闭包

内部函数可以使用外部函数变量的行为，就叫闭包

二、Linux 基础和数据结构与算法

1. 10 个常用的 Linux 命令？

答案：略

2. find 和 grep 的区别？

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。

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。

3. 什么是阻塞？什么是非阻塞？

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，实际上他是不同的。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。

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。

4. linux 重定向命令有哪些？有什么区别？？

重定向>

Linux 允许将命令执行结果重定向到一个文件，本应显示在终端上的内容保存到指定文件中。如：ls > test.txt （ test.txt 如果不存在，则创建，存在则覆盖其内容 ）

重定向>>

>>这个是将输出内容追加到目标文件中。如果文件不存在，就创建文件；如果文件存在，

则将新的内容追加到那个文件的末尾，该文件中的原有内容不受影响。

5.软硬链接区别?

软连接类似 windows 的快捷方式，当删除源文件，那么软链接失效。硬链接可以理解为源文件的一个别名。多个别名所代表的是同一个文件。当 rm 一个文件的时候，那么此文件的硬链接数减 1，当硬链接数为 0 的时候，文件删除。

6.linux 关机命令有哪些?

命令	含义
reboot	重新启动操作系统
shutdown –r now	重新启动操作系统，shutdown 会给别的用户提示
shutdown -h now	立刻关机，其中 now 相当于时间为 0 的状态
shutdown -h 20:25	系统在今天的 20:25 会关机
shutdown -h +10	系统再过十分钟后自动关机
init 0	关机
init 6	重启

7.linux 下管道的作用?

一个命令的输出可以通过管道做为另一个命令的输入

三、Web 框架

1.django 中当一个用户登录 A 应用服务器（进入登录状态），然后下次请求被 nginx 代理到 B 应用服务器会出现什么影响?

如果用户在 A 应用服务器登陆的 session 数据没有共享到 B 应用服务器,纳米之前的登录状态就没有了。

2.跨域请求问题 django 怎么解决的（原理）

启用中间件

post 请求

验证码

表单中添加{%csrf_token%}标签

3.请解释或描述一下 Django 的架构

对于 Django 框架遵循 MVC 设计，并且有一个专有名词：MVT

M 全拼为 Model，与 MVC 中的 M 功能相同，负责数据处理，内嵌了 ORM 框架

V 全拼为 View,与 MVC 中的 C 功能相同,接收 HttpRequest,业务处理,返回 HttpResponse

T 全拼为 Template，与 MVC 中的 V 功能相同，负责封装构造要返回的 html，内嵌了模板引擎

4.django 对数据查询结果排序怎么做，降序怎么做，查询大于某个字段怎么做

排序使用 order_by()

降序需要在排序字段名前加-

查询字段大于某个值：使用 filter(字段名_gt=值)

5.说一下 Django，MIDDLEWARES 中间件的作用？

答：中间件是介于 request 与 response 处理之间的一道处理过程，相对比较轻量级，并且在全局上改变 django 的输入与输出。

6.你对 Django 的认识？

Django 是走大而全的方向，它最出名的是其全自动化的管理后台：只需要使用起 ORM，做简单的对象定义，它就能自动生成数据库结构、以及全功能的管理后台。

Django 内置的 ORM 跟框架内的其他模块耦合程度高。

应用程序必须使用 Django 内置的 ORM,否则就不能享受到框架内提供的种种基于其 ORM 的便利；理论上可以切换掉其 ORM 模块，但这就相当于要把装修完毕的房子拆除重新装修，倒不如一开始就去毛胚房做全新的装修。

Django 的卖点是超高的开发效率，其性能扩展有限；采用 Django 的项目，在流量达到一定规模后，都需要对其进行重构，才能满足性能的要求。

Django 适用的是中小型的网站，或者是作为大型网站快速实现产品雏形的工具。

Django 模板的设计哲学是彻底的将代码、样式分离； Django 从根本上杜绝在模板中进行编码、处理数据的可能。

7. Django 重定向你是如何实现的？用的什么状态码？

使用 HttpResponseRedirect

redirect 和 reverse

状态码：302,301

8.ngnix 的正向代理与反向代理？

答：正向代理 是一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端必须要进行一些特别的设置才能使用正向代理。

反向代理正好相反，对于客户端而言它就像是原始服务器，并且客户端不需要进行任何特别的设置。客户端向反向代理的命名空间中的内容发送普通请求，接着反向代理将判断向何处(原始服务器)转交请求，并将获得的内容返回给客户端，就像这些内容原本就是它自己的一样。

9. Tornado 的核是什么？

Tornado 的核心是 ioloop 和 iostream 这两个模块，前者提供了一个高效的 I/O 事件循环，后者则封装了 一个无阻塞的 socket 。通过向 ioloop 中添加网络 I/O 事件，利用无阻塞的 socket ，再搭配相应的回调 函数，便可达到梦寐以求的高效异步执行。

10.Django 本身提供了 runserver，为什么不能用来部署？

runserver 方法是调试 Django 时经常用到的运行方式，它使用 Django 自带的

WSGI Server 运行，主要在测试和开发中使用，并且 runserver 开启的方式也是单进程 。

uWSGI 是一个 Web 服务器，它实现了 WSGI 协议、uwsgi、http 等协议。注意 uwsgi 是一种通信协议，而 uWSGI 是实现 uwsgi 协议和 WSGI 协议的 Web 服务器。uWSGI 具有超快的性能、低内存占用和多 app 管理等优点，并且搭配着 Nginx

就是一个生产环境了，能够将用户访问请求与应用 app 隔离开，实现真正的部署 。相比来讲，支持的并发量更高，方便管理多进程，发挥多核的优势，提升性能。

11.创建项目，创建应用的命令？

django-admin startproject **

python manage.py startapp **

12. 生成迁移文件？执行迁移？

python manage.py makemigrations

python manage.py migrate

13. 关系型数据库的关系包括那些类型？

ForeignKey：一对多，将字段定义在多的一端中

ManyToManyField：多对多，将字段定义在两端中

OneToOneField：一对一，将字段定义在任意一端中

14. 查询集返回列表的过滤器有哪些？

all()：返回所有数据

filter()：返回满足条件的数据

exclude()：返回满足条件之外的数据，相当于 sql 语句中 where 部分的 not 关键字

order_by()：排序

15. 判断查询集中是否有数据？

exists()：判断查询集中是否有数据，如果有则返回 True，没有则返回 False

16.查询集两大特性？惰性执行？

惰性执行、缓存 。

创建查询集不会访问数据库，直到调用数据时，才会访问数据库，调用数据的情况包括迭代、序列化、与 if 合用

四、网络编程和前端

1.AJAX 是什么，如何使用 AJAX？

ajax(异步的 javascript 和 xml) 能够刷新局部网页数据而不是重新加载整个网页。

第一步，创建 xmlhttprequest 对象，var xmlhttp =new XMLHttpRequest ();XMLHttpRequest 对象用来和服务器交换数据。

第二步，使用 xmlhttprequest 对象的 open（）和 send（）方法发送资源请求给服务器。

第三步，使用 xmlhttprequest 对象的 responseText 或 responseXML 属性获得服务器的响应。

第四步，onreadystatechange 函数，当发送请求到服务器，我们想要服务器响应执行一些功能就需要使用 onreadystatechange 函数，每次 xmlhttprequest 对象的 readyState 发生改变都会触发 onreadystatechange 函数。

2. 常见的 HTTP 状态码有哪些？

200 OK

301 Moved Permanently

302 Found

304 Not Modified

307 Temporary Redirect

400 Bad Request

401 Unauthorized

403 Forbidden

404 Not Found

410 Gone

500 Internal Server Error

501 Not Implemented

3. Post 和 get 区别？

1、GET 请求，请求的数据会附加在 URL 之后，以?分割 URL 和传输数据，多个参数用&连接。URL 的编码格式采用的是 ASCII 编码，而不是 uniclde，即是说所有的非 ASCII 字符都要编码之后再传输。

POST 请求：POST 请求会把请求的数据放置在 HTTP 请求包的包体中。上面的 item=bandsaw 就是实际的传输数据。

因此，GET 请求的数据会暴露在地址栏中，而 POST 请求则不会。

2、传输数据的大小

在 HTTP 规范中，没有对 URL 的长度和传输的数据大小进行限制。但是在实际开发过程中，对于 GET，特定的浏览器和服务器对 URL 的长度有限制。因此，在使用 GET 请求时，传输数据会受到 URL 长度的限制。

对于 POST，由于不是 URL 传值，理论上是不会受限制的，但是实际上各个服务器会规定对 POST 提交数据大小进行限制，Apache、IIS 都有各自的配置。

3、安全性

POST 的安全性比 GET 的高。这里的安全是指真正的安全，而不同于上面 GET 提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过 GET 请求，用户名和密码都会暴露再 URL 上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET 请求提交的数据还可能会造成 Cross-site request frogery 攻击。

4.cookie 和 session 的区别？

1、cookie 数据存放在客户的浏览器上，session 数据放在服务器上。

2、cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗考虑到安全应当使用 session。

3、session 会在一定时间内保存在服务器上。当访问增多，会比较占用服务器的性能考虑到减轻服务器性能方面，应当使用 COOKIE。

4、单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

5、建议：

将登陆信息等重要信息存放为 SESSION

其他信息如果需要保留，可以放在 COOKIE 中

5.创建一个简单 tcp 服务器需要的流程

1.socket 创建一个套接字

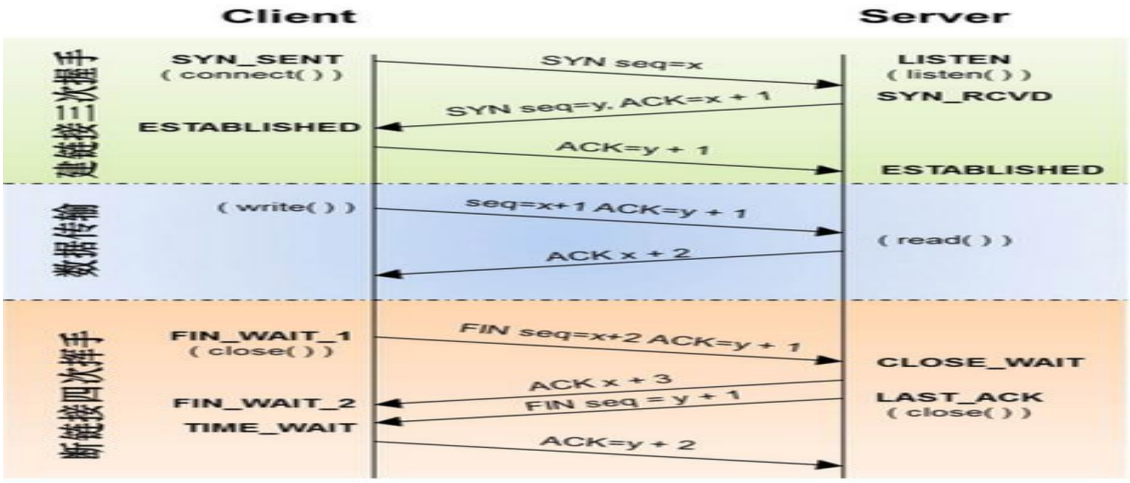
2.bind 绑定 ip 和 port

3.listen 使套接字变为可以被动链接

4.accept 等待客户端的链接

5.recv/send 接收发送数据

6.请简单说一下三次握手和四次挥手？什么是 2msl？为什么要这样做？



2MSL 即两倍的 MSL，TCP 的 TIME_WAIT 状态也称为 2MSL 等待状态，

当 TCP 的一端发起主动关闭，在发出最后一个 ACK 包后，

即第 3 次握 手完成后发送了第四次握手的 ACK 包后就进入了 TIME_WAIT 状态，

必须在此状态上停留两倍的 MSL 时间，

等待 2MSL 时间主要目的是怕最后一个 ACK 包对方没收到，

那么对方在超时后将重发第三次握手的 FIN 包，

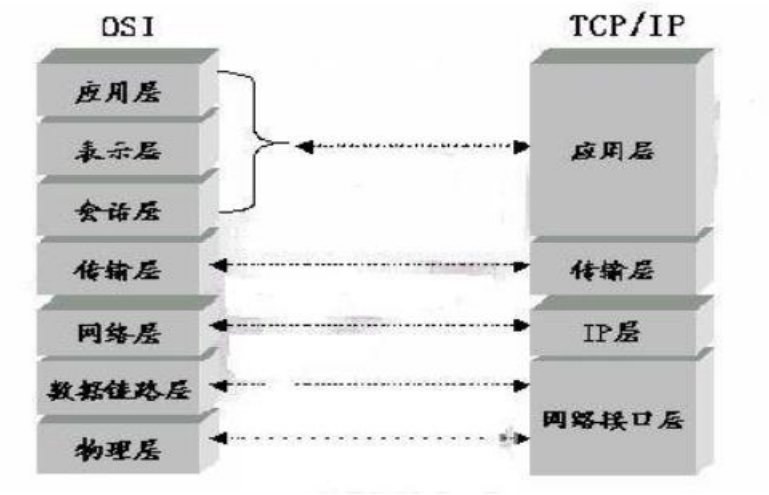
主动关闭端接到重发的 FIN 包后可以再发一个 ACK 应答包。

在 TIME_WAIT 状态 时两端的端口不能使用，要等到 2MSL 时间结束才可继续使用。

当连接处于 2MSL 等待阶段时任何迟到的报文段都将被丢弃。

不过在实际应用中可以通过设置 SO_REUSEADDR 选项达到不必等待 2MSL 时间结束再使用此端口。

7.七层模型？IP ，TCP/UDP， HTTP、RTSP、FTP 分别在哪层？



IP 网络层

TCP/UDP 传输层

HTTP、RTSP、FTP 应用层协议(

五、爬虫和数据库

1.scrapy 和 scrapy-redis 有什么区别？为什么选择 redis 数据库？

1) scrapy 是一个 Python 爬虫框架，爬取效率极高，具有高度定制性，但是不支持分布式。而 scrapy-redis 一套基于 redis 数据库、运行在 scrapy 框架之上的组件，可以让 scrapy 支持分布式策略，Slaver 端共享 Master 端 redis 数据库里的 item 队列、请求队列和请求指纹集合。

2) 为什么选择 redis 数据库，因为 redis 支持主从同步，而且数据都是缓存在内存中的，所以基于 redis 的分布式爬虫，对请求和数据的高频读取效率非常高。

2. 你用过的爬虫框架或者模块有哪些？谈谈他们的区别或者优缺点？

Python 自带：urllib，urllib2

第 三 方：requests

框　　架：Scrapy

urllib 和 urllib2 模块都做与请求 URL 相关的操作，但他们提供不同的功能。

urllib2.：urllib2.urlopen 可以接受一个 Request 对象或者 url，（在接受 Request 对象时候，并以此可以来设置一个 URL 的 headers），urllib.urlopen 只接收一个 url

urllib 有 urlencode,urllib2 没有，因此总是 urllib，urllib2 常会一起使用的原因

scrapy 是封装起来的框架，他包含了下载器，解析器，日志及异常处理，基于多线程，twisted 的方式处理，对于固定单个网站的爬取开发，有优势，但是对于多网站爬取 100 个网站，并发及分布式处理方面，不够灵活，不便调整与括展。

request 是一个 HTTP 库， 它只是用来，进行请求，对于 HTTP 请求，他是一个强大的库，下载，解析全部自己处理，灵活性更高，高并发与分布式部署也非常灵活，对于功能可以更好实现.

Scrapy 优缺点：

优点：scrapy 是异步的

采取可读性更强的 xpath 代替正则

强大的统计和 log 系统

同时在不同的 url 上爬行

支持 shell 方式，方便独立调试

写 middleware,方便写一些统一的过滤器

通过管道的方式存入数据库

缺点：基于 python 的爬虫框架，扩展性比较差

基于 twisted 框架，运行中的 exception 是不会干掉 reactor，并且异步框架出错后是不会停掉其他任务的，数据出错后难以察觉。

3.你常用的 mysql 引擎有哪些？各引擎间有什么区别？

主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：

一、InnoDB 支持事务，MyISAM 不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而 MyISAM 就不可以了；

二、MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用；

三、InnoDB 支持外键，MyISAM 不支持；

四、MyISAM 是默认引擎，InnoDB 需要指定；

五、InnoDB 不支持 FULLTEXT 类型的索引；

六、InnoDB 中不保存表的行数，如 select count(*) from table 时，InnoDB；需要扫描一遍整个表来计算有多少行，但是 MyISAM 只要简单的读出保存好的行数即可。注意的是，当 count(*)语句包含 where 条件时 MyISAM 也需要扫描整个表；

七、对于自增长的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM 表中可以和其他字段一起建立联合索引；

八、清空整个表时，InnoDB 是一行一行的删除，效率非常慢。MyISAM 则会重

建表；

九、InnoDB 支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like '%lee%'

4.描述下 scrapy 框架运行的机制？

答：从 start_urls 里获取第一批 url 并发送请求，请求由引擎交给调度器入请求队列，获取完毕后，调度器将请求队列里的请求交给下载器去获取请求对应的响应资源，并将响应交给自己编写的解析方法做提取处理：1. 如果提取出需要的数据，则交给管道文件处理；2. 如果提取出 url，则继续执行之前的步骤（发送 url 请求，并由引擎将请求交给调度器入队列...），直到请求队列里没有请求，程序结束。

5.什么是关联查询，有哪些？

答：将多个表联合起来进行查询，主要有内连接、左连接、右连接、全连接（外连接）

6.写爬虫是用多进程好？还是多线程好？ 为什么？

答：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)。在实际的数据采集过程中，既考虑网速和响应的问题，也需要考虑自身机器的硬件情况，来设置多进程或多线程

7.数据库的优化？

1. 优化索引、SQL 语句、分析慢查询；
2. 设计表的时候严格根据数据库的设计范式来设计数据库；
3. 使用缓存，把经常访问到的数据而且不需要经常变化的数据放在缓存中，能节约磁盘 IO；
4. 优化硬件；采用 SSD，使用磁盘队列技术(RAID0,RAID1,RDID5)等；
5. 采用 MySQL 内部自带的表分区技术，把数据分层不同的文件，能够提高磁盘的读取效率；
6. 垂直分表；把一些不经常读的数据放在一张表里，节约磁盘 I/O；
7. 主从分离读写；采用主从复制把数据库的读操作和写入操作分离开来；
8. 分库分表分机器（数据量特别大），主要的的原理就是数据路由；
9. 选择合适的表引擎，参数上的优化；
10. 进行架构级别的缓存，静态化和分布式；
11. 不采用全文索引；
12. 采用更快的存储方式，例如 NoSQL 存储经常访问的数据

8.常见的反爬虫和应对方法？

1）.通过 Headers 反爬虫

从用户请求的 Headers 反爬虫是最常见的反爬虫策略。很多网站都会对 Headers 的 User-Agent 进行检测，还有一部分网站会对 Referer 进行检测（一些资源网站的防盗链就是检测 Referer）。如果遇到了这类反爬虫机制，可以直接在爬虫中添加 Headers，将浏览器的 User-Agent 复制到爬虫的 Headers 中；或者将 Referer 值修改为目标网站域名。对于检测 Headers 的反爬虫，在爬虫中修改或者添加 Headers 就能很好的绕过。

2）.基于用户行为反爬虫

还有一部分网站是通过检测用户行为，例如同一 IP 短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。

大多数网站都是前一种情况，对于这种情况，使用 IP 代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理 ip，检测后全部保存起来。这样的代理 ip 爬虫经常会用到，最好自己准备一个。有了大量代理 ip 后可以每请求几次更换一个 ip，这在 requests 或者 urllib2 中很容易做到，这样就能很容易的绕过第一种反爬虫。

对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

3）.动态页面的反爬虫

上述的几种情况大多都是出现在静态页面，还有一部分网站，我们需要爬取的数据是通过 ajax 请求得到，或者通过 JavaScript 生成的。首先用 Fiddler 对网络请求进行分析。如果能够找到 ajax 请求，也能分析出具体的参数和响应的具体含义，我们就能采用上面的方法，直接利用 requests 或者 urllib2 模拟 ajax 请求，对响应的 json 进行分析得到需要的数据。

能够直接模拟 ajax 请求获取数据固然是极好的，但是有些网站把 ajax 请求的所有参数全部加密了。我们根本没办法构造自己所需要的数据的请求。这种情况下就用 selenium+phantomJS，调用浏览器内核，并利用 phantomJS 执行 js 来模拟人为操作以及触发页面中的 js 脚本。从填写表单到点击按钮再到滚动页面，全部都可以模拟，不考虑具体的请求和响应过程，只是完完整整的把人浏览页面获取数据的过程模拟一遍。

用这套框架几乎能绕过大多数的反爬虫，因为它不是在伪装成浏览器来获取数据（上述的通过添加 Headers 一定程度上就是为了伪装成浏览器），它本身就是浏览器，phantomJS 就是一个没有界面的浏览器，只是操控这个浏览器的不是人。利 selenium+phantomJS 能干很多事情，例如识别点触式（12306）或者滑动式的验证码，对页面表单进行暴力破解等。

9.分布式爬虫主要解决什么问题？

1)ip

2)带宽

3）cpu

4）io

10.爬虫过程中验证码怎么处理？

1.scrapy 自带（但是成功率不高）

2.付费接口（若快 http://www.ruokuai.com/home/pricetype）

11.redis 数据结构有哪些？

String——字符串

Hash——字典

List——列表

Set——集合

Sorted Set——有序集合

12. redis 中 list 底层实现有哪几种？有什么区别？

列表对象的编码可以是 ziplist 或者 linkedlist。

ziplist 是一种压缩链表，它的好处是更能节省内存空间，因为它所存储的内容都是在连续的内存区域当中的。当列表对象元素不大，每个元素也不大的时候，就采用 ziplist 存储。但当数据量过大时就 ziplist 就不是那么好用了。因为为了保证他存储内容在内存中的连续性，插入的复杂度是 O(N)，即每次插入都会重新进行 realloc。如下图所示，对象结构中 ptr 所指向的就是一个 ziplist。整个 ziplist 只需要 malloc 一次，它们在内存中是一块连续的区域。

13. QPS？

每秒查询率

QPS（TPS）：每秒钟 request/事务　数量

六、数据结构与算法

1.基础的数据结构有哪些？

集合、线性结构、树形结构、图状结构

集合结构:除了同属于一种类型外，别无其它关系

线性结构:元素之间存在一对一关系常见类型有: 数组,链表,队列,栈,它们之间在操作上有所区别.例如:链表可在任意位置插入或删除元素,而队列在队尾插入元素,队头删除元素,栈只能在栈顶进行插

入,删除操作.

树形结构:元素之间存在一对多关系,常见类型有:树(有许多特例:二叉树、平衡二叉树、查找树等)

图形结构:元素之间存在多对多关系,图形结构中每个结点的前驱结点数和后续结点多个数可以任意

2.怎么评价一个算法的好坏？

①时间复杂度：同样的输入规模（问题规模）花费多少时间

②空间复杂度：同样的输入规模花费多少空间（主要是内存）

以上两点越小越好

③稳定性：不会因为输入的不同而导致不稳定的情况发生

④算法思路是否简单：越简单越容易实现越好

3.算法的特性？

数据结构算法具有五个基本特征：输入、输出、有穷性、确定性和可行性

4.有没有了解过桶排序？

工作的原理是将数组分到有限数量的桶子里。每个桶子再个别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排序）

1,桶排序是稳定的

2,桶排序是常见排序里最快的一种,比快排还要快…大多数情况下

3,桶排序非常快,但是同时也非常耗空间,基本上是最耗空间的一种排序算法

5.快排/冒泡的思想？

冒泡思想：通过无序区中相邻记录的关键字间的比较和位置的交换，使关键字最小的记录像气泡一样逐渐向上漂至水面。整个算法是从最下面的记录开始，对每两个相邻的关键字进行比较，把关键字较小的记录放到关键字较大的记录的上面，经过一趟排序后，关键字最小的记录到达最上面，接着再在剩下的记录中找关键字次小的记录，把它放在第二个位置上，依次类推，一直到所有记录有序为止

复杂度：时间复杂度为0（n2）,空间复杂度为0(1)

快排思想：

复杂度：快速排序是不稳定的排序算法，最坏的时间复杂度是0（n2），

最好的时间复杂度是(nlogn),空间复杂度为0(logn)

快排的基本思想：通过一躺排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一不部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

6.你知道几种排序,讲一讲你最熟悉的一种?

排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
希尔排序	$O(n\log n)$~ $O(n^2)$	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
堆排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	O(1)	不稳定
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	稳定
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$~ $O(n)$	不稳定