前　言

下列题目要求每位同学在总结后能够用自己的语言清晰地把答案表达出来，所有答案仅做参考使用，不要求教科书式死记硬背。如果在面试中遇到不会的题目，能够说出思路也是可以的。希望每位同学在面试前能够有针对性的进行重点复习，面试后能够善于总结。前人栽树，后人乘凉。现在的题目都是前辈们心血所在，希望大家放开心胸，能够主动分享自己在笔试面试过程中遇到的各种题目，集思广益。最后，祝大家都能早日找到心仪的工作。

待添加.....

第一章、Python 基础及综合类

1. 现有字典 d={‘a’:24,’ g’:52,’ l’:12,’ k’:33}请按字典中的 value 值进行排序？

```
sorted(d.items(),key = lambda x:x[1])
```

2. map 函数和 reduce 函数？

①从参数方面来讲：

map()包含两个参数，第一个是参数是一个函数，第二个是序列（列表或元组）。其中，函数（即 map 的第一个参数位置的函数）可以接收一个或多个参数。

reduce() 第一个参数是函数，第二个是 序列（列表或元组）。但是，其函数必须接收两个参数。

②从对传进去的数值作用来讲：

map()是将传入的函数依次作用到序列的每个元素，每个元素都是独自被函数“作用”一次；（请看下面的例子）

reduce()是将传人的函数作用在序列的第一个元素得到结果后，把这个结果继续与下一个元素作用（累积计算）。

3. python 的魔法方法

魔法方法就是可以给你的类增加魔力的特殊方法，如果你的对象实现（重载）了这些方法中的某一个，那么这个方法就会在特殊的情况下被 Python 所调用，你可以定义自己想要的行为，而这一切都是自动发生的。它们经常是两个下划线包围来命名的（比如 __init__，__lt__），Python 的魔法方法是非常强大的，所以了解其使用方法也变得尤为重要！

__init__ 构造器，当一个实例被创建的时候初始化的方法。但是它并不是实例化调用的第一个方法。

__new__才是实例化对象调用的第一个方法，它只取下 cls 参数，并把其他参数传给 __init__。 __new__很少使用，但是也有它适合的场景，尤其是当类继承自一个像元组或者字符串这样不经常改变的类型的时候。

__call__ 允许一个类的实例像函数一样被调用

__getitem__ 定义获取容器中指定元素的行为，相当于 self[key]

__getattr__ 定义当用户试图访问一个不存在属性的时候的行为

__setattr__ 定义当一个属性被设置的时候的行为

__getattribute__ 定义当一个属性被访问的时候的行为

4. 求输出结果

```
def num():
   return [lambda x:i*x for i in range(4)]
```

```
print [m(2) for m in num()]
[6, 6, 6, 6]
```

5. 对缺省参数的理解？

缺省参数指在调用函数的时候没有传入参数的情况下,调用默认的参数,在调用函数的同时赋值时,所传入的参数会替代默认参数

6. 对装饰器的理解，并写出一个计时器记录方法执行性能的装饰器。

装饰器本质上是一个 Python 函数,它可以让其他函数在不需要做任何代码变动的前提下增加额外功能，装饰器的返回值也是一个函数对象

```
import time
def timeit(func):
    def wrapper():
        start = time.clock()
        func() end =time.clock()
        print 'used:', end - start
        return wrapper
@timeit
def foo():
print 'in foo()'foo()
```

7. 什么是线程安全，什么是互斥锁？

每个对象都对应于一个可称为" 互斥锁" 的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。

同一个进程中的多线程之间是共享系统资源的,多个线程同时对一个对象进行操作,一个线程操作尚未结束,另一个线程已经对其进行操作,导致最终结果出现错误,此时需要对被操作对象添加互斥锁,保证每个线程对该对象的操作都得到正确的结果

8. 什么单例模式，其应用场景都有哪些？

确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，单例模式是一种对象创建型模式。

Windows 的 Task Manager（任务管理器）、Recycle Bin（回收站）、网站计数器

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。如上述中的日志文件，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。

9. 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

10. python 是强语言还是弱语言。

Python 是强类型的动态脚本语言 。

强类型：不允许不同类型相加

动态：不使用显示数据类型声明，且确定一个变量的类型是在第一次给它赋值的时候

脚本语言：一般也是解释型语言，运行代码只需要一个解释器，不需要编译

11. python 的内存管理机制及调优手段？

内存管理机制：引用计数、垃圾回收、内存池

引用计数

引用计数是一种非常高效的内存管理手段，当一个 Python 对象被引 用时其引用计数增加 1，当其不再被一个变量引用时则计数减 1. 当引用计数等于 0 时对象被删除.

垃圾回收

1. 引用计数

引用计数也是一种垃圾收集机制，而且也是一种最直观，最简单的垃圾收集技术。当 Python 的某个对象的引用计数降为 0 时，说明没有任何引用指向该对象，该对象就成为要被回收的垃圾了。比如某个新建对象，它被分配给某个引用，对象的引用计数变为 1。如果引用被删除，对象的引用计数为 0，那么该对象就可以被垃圾回收。

不过如果出现循环引用的话，引用计数机制就不再起有效的作用了

2. 标记清除

如果两个对象的引用计数都为 1，但是仅仅存在他们之间的循环引用，那么这两个对象都是需要被回收的，也就是说，它们的引用计数虽然表现为非 0，但实际上有效的引用计数为 0。所以先将循环引用摘掉，就会得出这两个对象的有效计数。

3. 分代回收

从前面“标记-清除”这样的垃圾收集机制来看，这种垃圾收集机制所带来的额外操作实际上与系统中总的内存块的数量是相关的，当需要回收的内存块越多时，垃圾检测带来的额外操作就越多，而垃圾回收带来的额外操作就越少；反之，当需回收的内存块越少时，垃圾检测就将比垃圾回收带来更少的额外操作。

举个例子：

当某些内存块 M 经过了 3 次垃圾收集的清洗之后还存活时，我们就将内存块 M 划到一个集合 A 中去，而新分配的内存都划分到集合 B 中去。当垃圾收集开始工作时，大多数情况都只对集合 B 进行垃圾回收，而对集合 A 进行垃圾回收要隔相当长一段时间后才进行，这就使得垃圾收集机制需要处理的内存少了，效率自然就提高了。在这个过程中，集合 B 中的某些内存块由于存活时间长而会被转移到集合 A 中，当然，集合 A 中实际上也存在一些垃圾，这些垃圾的回收会因为这种分代的机制而被延迟。

内存池

1.Python 的内存机制呈现金字塔形状，-1，-2 层主要有操作系统进行操作；

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

3.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；

4.第 3 层是最上层，也就是我们对 Python 对象的直接操作；

Python 在运行期间会大量地执行 malloc 和 free 的操作，频繁地在用户态和核心态之间进行切换，这将严重影响 Python 的执行效率。为了加速 Python 的执行效率，Python 引入了一个内存池机制，用于管理对小块内存的申请和释放。

Python 内部默认的小块内存与大块内存的分界点定在 256 个字节，当申请的内存小于 256 字节时，PyObject_Malloc 会在内存池中申请内存；当申请的内存大于 256 字节时，PyObject_Malloc 的行为将蜕化为 malloc 的行为。当然，通过修改 Python 源代码，我们可以改变这个默认值，从而改变 Python 的默认内存管理行为。

调优手段（了解）

1.手动垃圾回收

2.调高垃圾回收阈值

3.避免循环引用（手动解循环引用和使用弱引用）

12. 生成器、迭代器的区别？

在 Python 中，一边循环一边计算的机制，称为生成器：generator，生成器是可以迭代对象，但是生成器可以通过 send 传值返回到前面；

迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退

迭代器是一个可以记住遍历的位置的对象

13. 对不定长参数的理解？

不定长参数有两种：*args 和**kwargs;

*args：是不定长参数，用来将参数打包成 tuple 给函数体调用

**kwargs:是关键字参数，打包关键字参数成 dict 给函数体调用在定义函数的时候不确定要传入的参数个数会有多少个的时候就可以使用不定长参数作为形参

14. 什么是 lambda 函数？它有什么好处？写一个匿名函数求两个

数的和。

lambda 函数是匿名函数；使用 lambda 函数能创建小型匿名函数。这种函数得名于省略了用 def 声明函数的标准步骤；

```
f = lambda x,y:x+y
print(f(2017,2018))
```

15. 写一个列表生成式，例:产生一个公差为 1 的等差数列

```
print([x for x in range(10)])
```

16. 解释一下什么是闭包

在函数内部再定义一个函数，并且这个函数用到了外边函数的变量，那么将这个函数以及用到的一些变量称之为闭包

17. 解释一下什么是锁，有哪几种锁?

锁(Lock)是 python 提供的对线程控制的对象
有互斥锁、可重入锁、死锁

18. 什么是死锁以及怎么解决?

死锁：在线程间共享多个资源的时候，如果两个线程分别占有一部分资源并且同时等待对方的资源，就会造成死锁
给互斥锁添加超时时间
程序设计时要尽量避免（银行家算法）

19. 什么是僵尸进程和孤儿进程,怎么避免僵尸进程?

孤儿进程：父进程退出，子进程还在运行的这些子进程都是孤儿进程，孤儿进程将被 init 进程(进程号为 1)所收养，并由 init 进程对它们完成状态收集工作
僵尸进程：进程使用 fork 创建子进程，如果子进程退出，而父进程并没有调用 wait 或 waitpid 获取子进程的状态信息,那么子进程的进程描述符仍然保存在系统中的这些进程是僵尸进程
避免僵尸进程的方法：1.fork 两次用孙子进程去完成子进程的任务
2.用 wait()函数使父进程阻塞
3.使用信号量，在 signal handler 中调用 waitpid，这样父进程不用阻塞

20. 谈一下什么是解释性语言,什么是编译性语言?

计算机不能直接理解高级语言，只能直接理解机器语言，所以必须要把

高级语言翻译成机器语言，计算机才能值型高级语言编写的程序

解释性语言在运行程序的时候才翻译，解释性语言在运行程序的时候才翻译

编译型语言写的程序执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件

21. 捕获异常机制，在 except 中 return 后还会不会执行 finally 中的代码？怎么抛出自己处理不了的异常？

会继续处理 finally 中的代码；
用 raise 方法可以手动抛出异常

22. Python 中有日志吗?怎么使用?

有日志；

python 自带 logging 模块，调用 logging.basicConfig()方法，配置需要的日志等级和相应的参数，python 解释器会按照配置的参数生成相应的日志

23. 代码中要修改不可变数据会出现什么问题?抛出什么异常?

代码不会正常运行，抛出异常 TypeError 异常

24. python 中的进程与线程的使用场景?

多进程适合在 CPU 密集型操作(cpu 操作指令比较多，如位数多的浮点运算)

多线程适合在 IO 密集型操作(读写数据操作较多的，比如爬虫)

25. 为什么函数名字可以当做参数用？

python 中一切皆对象，函数名是函数在内存中的空间，也是一个对象

26. linux 的基本命令（怎么区分一个文件还是文件夹）

ls -F 在显示名称的时候会在文件夹后添加“/”，在文件后面加“*”

27. 日志以什么格式，存放在哪里？

日志以文本可以存储在“/var/log/”目录下后缀名为.log

28. 回调函数，如何通信的?

回调函数是把函数的指针(地址)作为参数传递给另一个函数，将整个函数当作一个对象，赋值给调用的函数

29. 浮点数如何进行比较大小?

浮点数的表示是不精确的，不能直接比较两个数是否完全相等，一般都是在允许的某个范围内认为像个浮点数相等，如有两个浮点数 a,b，允许的误差范 围为 1e-6，则 abs(a-b)<=1e-6，即可认为 a 和 b 相等

python3.5 的 math 模块新增一个 isclose 函数用来判断两个浮点数的值是否接近或相等

```
import math
print(math.isclose(1.25,1.25))# 返回一个布尔值 True
print(math.isclose(2.1,2.2,rel_tol=0.1))
```

参数：

a,b:两个需要比较的浮点数；

rel_tol: 相对于输入值的大小，被认为是“接近”的最大差异;

abs_tol: 无论输入值的大小，被认为“接近”的最大差异

30. 线程是并发还是并行，进程是并发还是并行？

线程是并发，进程是并行；进程之间相互独立，是系统分配资源的最小单位，同一个线程中的所有线程共享资源

31. 什么是可变、不可变？元组里添加字典，会改变 id 吗？

可变不可变指的是内存中的值是否可以被改变，不可变类型指的是对象所在内存块里面的值不可以改变，有数值、字符串、元组；可变类型则是可以改变，主要有列表、字典

元组的顶层元素中包含可变类型,在可变类型中修改或添加字典 id 不会改变

32. init 和 news 的区别？

init 在对象创建后，对对象进行初始化

new 是在对象创建之前创建一个对象，并将该对象返回给 init

33. 存入字典里的数据有没有先后排序？

存入的数据不会自动排序，可以使用 sort 函数对字典进行排序

34. 解释一下 Python 中的 and-or 语法？

逻辑运算 and-or，在计算机运算中的短路规则（以尽量少运算，得出正确结果）可以提高计算效率，

在 result = a and b 运算中：

当 a 为假时，无论 b 为真或假，结果都为假，此时 b 的运算就不会进行，结果直接为 a 即可；

当 a 为真时，结果还得看 b，b 为真则真，b 为假则假，此时结果即为 b；

在 result = a or b 运算中：

如果 a 为真则无论 b 为什么结果都会是真，结果即为 b

如果 a 为假则看 b 的情况了，b 为真则结果为真，b 为假则结果为假，即结果为 b

35. Python 是如何进行类型转换的？

内建函数封装了各种转换函数，可以使用目标类型关键字强制类型转换

36. 请写出一段 Python 代码实现删除一个 list 里面的重复元素。

```
lista = [1,2,2,3,3,3,4]
用 set()方法
new_list = list(set(lista))
自定义函数
def distinct(lista):
    new_list = []
    if not lista:
        return
    else:
        for i in lista:
            if i not in new_list:
                new_list.append(i)
            else:
                continue
    return new_list

lista = [1,2,2,3,3,3,4,4,4,4]
print(distinct(lista))
```

37. Python 中类方法、类实例方法、静态方法有何区别？

类方法：是类对象的方法，在定义时需要在上方使用“@classmethod”进行装饰，形参为 cls，表示类对象，类对象和实例对象都可调用；

类实例方法：是类实例化对象的方法，只有实例对象可以调用，形参为 self，指代对象本身；

静态方法：是一个任意函数，在其上方使用“@staticmethod”进行装饰，可以用对象直接调用，静态方法实际上跟该类没有太大关系

38. Python 中 pass 语句的作用是什么

在编写代码时只写框架思路，具体实现还未编写就可以用 pass 进行占位，使程序不报错，不会进行任何操作

39. 谈谈你对同步、异步、阻塞、非阻塞的理解

在计算机领域，同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程就会一直等待下去，直到收到返回信息才继续执行下去，异步是值进程不需要一直等下去，而继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执的效率。举个例子，打电话时就是同步通信，发短息时就是异步通信。

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。

推荐：https://www.cnblogs.com/Anker/p/5965654.html

40. 介绍一下 Python 下 range（）和 xrange（）函数的用法？

range(start,stop,step)函数按照从 start 到 stop 每个 step 生成一个数值，生成的是列表对象，一次性将所有数据都返回；

xrange(start,stop,step)函数按照从 start 到 stop 每个 step 生成一个数值，返回的是可迭代对象，每次调用返回其中的一个值

41. 用 Python 匹配 HTML tag 的时候，<.*>和<.*?>有什么区别？

<.*>是贪婪匹配，会从第一个“<”开始匹配，直到最后一个“>”中间所有的字符都会匹配到，中间可能会包含“<>”

<.*?>是非贪婪匹配，从第一个“<”开始往后，遇到第一个“>”结束匹配，这中间的字符串都会匹配到，但是不会有“<>”

42. Python 里面如何拷贝一个对象？

python 中的拷贝分为浅拷贝和深拷贝，若不特殊说明拷贝一般是浅拷贝

浅拷贝是将对象顶层拷贝，拷贝了引用，并没有拷贝内容，原对象改变，新对象也跟着改变

深拷贝是对一个对象的所有层次的拷贝（递归），但是修改原来的值，新对象不受影响

浅拷贝对于可变类型和不可变类型是不同的，对于可变类型只拷贝顶层，不可变类型依然是原来的对象

43. 如何用 Python 来进行查询和替换一个文本字符串？

使用正则表达式

```
re.findall(r’目的字符串’,’原有字符串’) #查询
re.findall(r'cast','itcast.cn')[0]
re.sub(r‘要替换原字符’,’要替换新字符’,’原始字符串’)
re.sub(r'cast','heima','itcast.cn')
```

44. 如何理解 Python 中字符串中的\字符。

转义字符

路径名中用来连接路径名

编写太长代码手动软换行

45. 说明一下 os.path 和 sys.path 分别代表什么？

os.path 主要是用于用户对系统路径文件的操作

sys.path 主要用户对 python 解释器的系统环境参数的操作

46. 考虑一下 Python 代码，如果运行结束，命令行中的运行结果是什么？

```
If __name__ == ’__main__’:
    l = []
    for i inxrange(10):
        l.append({ ‘num’ :i})
    print l
```

在考虑一下代码，运行结束后的结果是什么？

```
if __name__ == ‘__main__’:
l = []
a = { ‘num’ :0}
for i in xrange(10):
    a[ ‘num’ ] = i
    l.append(a)
print l
```

以上两端代码的运行结果是否相同，如果不相同，原因是什么？

上方代码的结果：

```
[{‘num’:0},{‘num’:1},{‘num’:2},{‘num’:3},{‘num’:4},{‘num’:5},{
```

‘num’ :6},{ ‘num’ :7},{ ‘num’ :8},{ ‘num’ :9}]

下方代码结果：

[{‘num’:9},{‘num’:9},{‘num’:9},{‘num’:9},{‘num’:9},{‘num’:9},{‘num’ :9},{ ‘num’ :9},{ ‘num’ :9},{ ‘num’ :9}]

原因是：字典是可变对象，在下方的 l.append(a)的操作中是把字典 a 的引用传到列表 l 中，当后续操作修改 a[‘num’]的值的时候，l 中的值也会跟着改变，相当于浅拷贝

47. 请尝试用“一行代码”实现将 1-N 的整数列表以 3 为单位分组，比如 1-100 分组后为?

```
print([[x for x in range(1,100)][i:i+3] for i in range(0,len(list_a),3)])
```

48. 现在考虑有一个 jsonline 格式的文件 file.txt 大小约为 10K，JAMY 之前处理文件的代码如下所示：

```
def get_lines():
    l = []
    with open( ‘file.txt’ ,’ rb’ ) as f:
        for eachline in f:
            l.append(eachline)
    return l

if __name__ == ‘__main__’ :
    for e in get_lines():
        process(e)   #处理每一行数据
```

现在 JAMY 被交付一个大小为 10G 的文件，但是内存只有 4G，如果在只修改 get_lines 函数而其他代码保持不变的情况下,应该如何实现。需要考虑的问题都有哪些。

```
def get_lines():
    l = []
    with open( ‘file.txt’ ,’ rb’ ) as f:
        data = f.readlines(60000)
        l.append(data)
        yield l
```

要考虑到的问题有：

内存只有 4G 无法一次性读入 10G 的文件，需要分批读入；

分批读入数据要纪录每次读入数据的位置；

分批读入每次读入数据的大小，太小就会在读取操作上话费过多时间

49. 已知路径 path，写一个函数并以前向遍历的方式打印目录中的所有文件

（可使用 os.listdir，os.path.isdir，os.path.join 和递归的方式来实现），比如如下的一个文件树结构：

```
root
 |——————a.file
 |——————b.dir
 |      |———ba.file
 |      └———c.dir
 |        └———d.file
 |——————e.file
 └——————f.dir
    |———————g.file
└———h.dir
   └———i.file
```

将 root 的绝对路径传递给函数 lookup 后，打印的内容如下：

```
/root/testWork/test_dir/a.file
/root/testWork/test_dir/b.dir/ba.file
/root/testWork/test_dir/b.dir/c.dir/d.file
/root/testWork/test_dir/e.file
/root/testWork/test_dir/f.dir/g.file
/root/testWork/test_dir/f.dir/h.dir/i.file
```

请实现此函数。

```
def lookup(path):
# -*- coding:utf-8 -*-
# 要求：升序遍历指定目录中的所有文件名
import os
full_names = []
def lookup(path=None):
    global full_names
    if not path:
        dirs = os.getcwd()
    else:
        if  os.path.isdir(path):
            dirs = path
        else:
            print(path)
            return
    # 遍历路径下的所有目录，并按顺序排序
    os.chdir(dirs)
    dirs = os.getcwd()
    dir_files = os.listdir(dirs)
    dir_files.sort()
```

```
        for dir_file in dir_files:
            if os.path.isfile(dir_file):
                full_name = os.path.join(dirs,dir_file)
                full_names.append(full_name)
            elif os.path.isdir(dir_file):
                lookup(dir_file)
        os.chdir('../')

    if __name__ == '__main__':
        dir = input('请输入一个要遍历的目录：')
        lookup(dir)
        for full_name in full_names:
            print(full_name)
```

50. 并行（parallel）和并发（concurrency）？

并行：同一时刻多个任务同时在运行
并发：在同一时间间隔内多个任务都在运行，但是并不会在同一时刻同时运行，存在交替执行的情况
实现并行的库有：multiprocessing
实现并发的库有：threading
程序需要执行较多的读写、请求和回复任务的需要大量的 IO 操作，IO 密集型操作使用并发更好
CPU 运算量大的程序程序，使用并行会更好

51. Python 主要的内置数据类型都有哪些？print dir（‘a’）的输出？

内建类型：布尔类型、数字、字符串、列表、元组、字典、集合；
输出字符串‘a’的内建方法

52. 给定两个 listA、B,请用 Python 找出 A、B 中相同的元素，A、B 中不同的元素

A、B 中相同元素：print(set(A)&set(B))
A、B 中不同元素：print(set(A)^set(B))

53. 请反转字符串’aStr’

```
print(‘aStr’[::-1])
```

54. a=1,b=2,不用中间变量交换 a 和 b 的值？

```
a=a+b
b=a-b
a=a-b
```

55. 给定一个 int list a，满足 a[i + 1]>=a[i],给定 int key

找出 list a 中第一个大于等于 key 的元素的 index，无满足要求的元素则返回-1

```
def findindex(int_list,int_key):
    int_list = sorted(int_list)
    for i in range(len(int_list)):
        if int_list[i] == int_key:
            return i
    return -1

if __name__ == "__main__":
    lista = [12,3,4,5,8,6]
    index = findindex(lista,5)
    print(index)
```

56. 正则表达式

正则匹配中贪婪模式与非贪婪模式的区别？match、search 函数的使用及区别？请写出以字母或下划线开始，以数字结束的正则表达式？

①贪婪模式：正则会尽量多的匹配字符串，被匹配到的字符串里面可能包含多个可以终止匹配的字符，但是会匹配到最后一个字符才会停止；非贪婪模式：正则会保证符合规则的情况下尽量少匹配字符，不会出现匹配到的字符中包含可以终止匹配的字符的情况

②match 是从字符串的第一个字符开始依次往后匹配，中途若出现不匹配的情况则立即终止匹配，返回寄过为空，匹配到第一个满足要求的字符串也停止匹配并返回匹配到的字符串；search 也是有第一个字符开始，若第一个字符不匹配则从下一个字符开始匹配（即可以从中间开始匹配），知道匹配到第一个符合条件字符串，或者匹配到最后一个字符终止匹配

③r’^[A-Za-z]|[_]\.*\d$’

57. 描述 Python GIL 的概念，以及它对 Python 多线程的影响？

GIL：全局解释器锁。每个线程在执行的过程都需要先获取 GIL，保证同一时刻只有一个线程可以执行字节码。

线程释放 GIL 锁的情况：

在 IO 操作等可能会引起阻塞的 system call 之前,可以暂时释放 GIL,但在执行完毕后,必须重新获取 GIL

Python 3.x 使用计时器（执行时间达到阈值后，当前线程释放 GIL）或 Python 2.x，tickets 计数达到 100

Python 使用多进程是可以利用多核的 CPU 资源的。
多线程爬取比单线程性能有提升，因为遇到 IO 阻塞会自动释放 GIL 锁

58. Python 中如何动态获取和设置对象的属性；

```
if hasattr(Parent,'x'):
    print(getattr(Parent,'x'))
    setattr(Parent,'x',3)
    print(getattr(Parent,'x'))
```

59. git merge 和 git rebase 有什么区别？git reset 和 git revert 有什么区别？

merge 操作会生成一个新的节点，之前的提交分开显示。而 rebase 操作不会生成新的节点，是将两个分支融合成一个线性的提交

git revert 是生成一个新的提交来撤销某次提交，此次提交之前的 commit 都会被保留

git reset 是回到某次提交，提交及之前的 commit 都会被保留，但是此次之后的修改都会被退回到暂存区

60. Print 调用 python 中底层的什么方法？

print 方法默认调用 sys.stdout.write 方法，即往控制台打印字符串

61. dict 的 items()和 iteritems()方法的不同？

items()返回的是列表对象，而 iteritems()返回的是 iterator 对象

62. 函数调用一个变量的顺序？

Python 变量访问时有个 LEGB 原则，也就是说，变量访问时搜索顺序为 Local ==> Enclosing ==> Global ==> Builtin

简单地说，访问变量时，先在当前作用域找，如果找到了就使用，如果没找到就继续到外层作用域看看有没有，找到了就使用，如果还是没找到就继续到更外层作用域找，如果已经到了最外层作用域了还是实在找不到就看看是不是内置对象，如果也不是，抛出异常。

63. 面向对象中怎么实现只读属性？

将对象私有化,通过共有方法提供一个读取数据的接口

64. git 合并文件有冲突，如何处理？

1、git merge 冲突了，根据提示找到冲突的文件，解决冲突如果文件有冲突，那么会有类似的标记

2、修改完之后，执行 git add 冲突文件名

3、git commit 注意:没有-m 选项 进去类似于 vim 的操作界面，把 conflict 相关的行删除掉

4、直接 push 就可以了，因为刚刚已经执行过相关 merge 操作了，

第二章、网络编程及前端

65. 谈一下 http 协议以及协议头部中表示数据类型的字段？

HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

HTTP 协议工作于客户端-服务端架构为上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

表示数据类型字段：Content-Type

66. 服务器处理响应的顺序有哪些方式？ select 和 epoll 区别？

Select/epoll/poll

区别：

1.select 的句柄数目受限，在 linux/posix_types.h 头文件有这样的声明：#define __FD_SETSIZE 1024 表示 select 最多同时监听 1024 个 fd。而 epoll 没有，它的限制是最大的打开文件句柄数目。

2.epoll 的最大好处是不会随着 FD 的数目增长而降低效率，在 selec 中采用轮询处理，其中的数据结构类似一个数组的数据结构，而 epoll 是维护一个队列，直接看队列是不是空就可以了。epoll 只会对"活跃"的 socket 进行操作---这是因为在内核实现中 epoll 是根据每个 fd 上面的 callback 函数实现的。那么，只有"活跃"的 socket 才会主动的去调用 callback 函数(把这个句柄加入队列)，其他 idle 状态句柄则不会，在这点上，epoll 实现了一个"伪"AIO。但是如果绝大部分的 I/O 都是“活跃的”，每个 I/O 端口使用率很高的话，epoll 效率不一定比 select 高（可能是要维护队列复杂）。

3.使用 mmap 加速内核与用户空间的消息传递。无论是 select,poll 还是 epoll 都需要内核把 FD 消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll 是通过内核于用户空间 mmap 同一块内存实现的。

67. Ip 包中有哪些字段，以及 ip 包的大小是多少？

字段：版本号/ IP 包头长度/服务类型/ IP 包总长/标识符/标记/片偏移/生存时间/协议/头部校验/起源和目标地址/可选项/填充

大小：最大长度 65535 字节

68. Tcp 协议和 udp 协议有什么区别？

1、TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接

2、TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保　　证可靠交付

3、TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流;UDP 是面向报文的

UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）

4、每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信

5、TCP 首部开销 20 字节;UDP 的首部开销小，只有 8 个字节

6、TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

69. 知道 arp 报文吗？全称叫什么以及有什么作用？

ARP (Address Resolution Protocol) 是个地址解析协议。最直白的说法是：在 IP 以太网中，当一个上层协议要发包时，有了该节点的 IP 地址，ARP 就能提供该节点的 MAC 地址。　报文在传输的过程中在路由器中的处理过程

70. 长链接与短链接的区别？

短连接

连接->传输数据->关闭连接

HTTP 是无状态的，浏览器和服务器每进行一次 HTTP 操作，就建立一次连接，但任务结束后就中断连接。短连接是指 SOCKET 连接后发送后接收完数据后马上断开连接。

长连接

连接->传输数据->保持连接->传输数据->....->关闭连接

长连接指建立 SOCKET 连接后不管是否使用都保持连接，但安全性较差。

71. HTTP 协议状态码有什么用，列出你知道的 HTTP 协议的状态码，然后讲出他们都表示什么意思。

作用：告诉用户当前操作所处的状态

200 OK 服务器成功处理了请求（这个是我们见到最多的）

301/302 Moved Permanently（重定向）请求的 URL 已移走。Response 中应该包含一个 Location URL, 说明资源现在所处的位置

304 Not Modified（未修改）客户的缓存资源是最新的， 要客户端使用缓

存

404 Not Found 未找到资源

501 Internal Server Error 服务器遇到一个错误，使其无法对请求提供服务

72. 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。

73. 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：

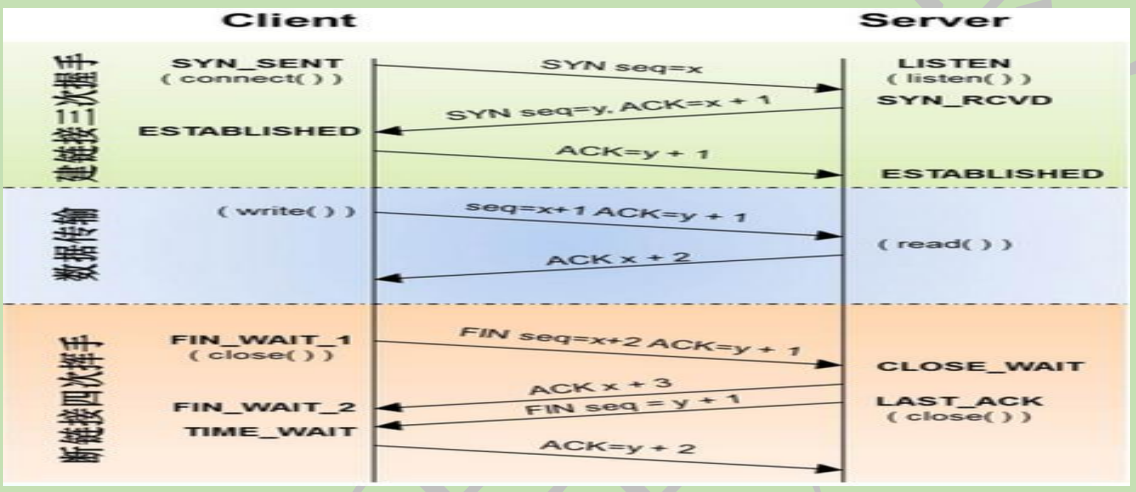
将登陆信息等重要信息存放为 SESSION

其他信息如果需要保留，可以放在 COOKIE 中

74. 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

75. 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。

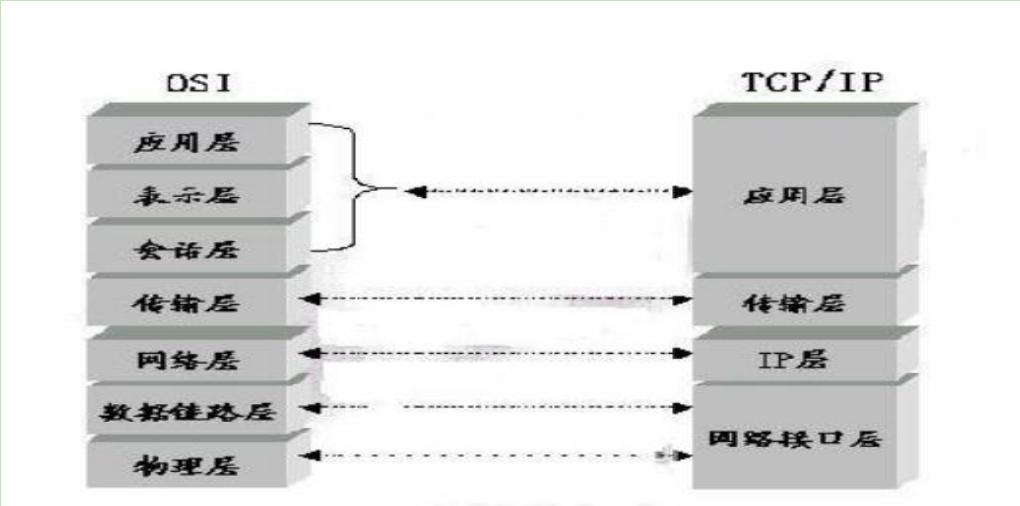
在 TIME_WAIT 状态 时两端的端口不能使用，要等到 2MSL 时间结束才可继续使用。

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。

76. 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



77. 从输入 http://www.baidu.com/ 到页面返回，中间都是发生了什么？

（浏览器发送一个请求到返回一个页面的具体过程）

第一步，解析域名，找到 ip

浏览器会缓存 DNS 一段时间，一般 2-30 分钟不等，如果有缓存，直接返回 ip，否则下一步。

缓存中无法找到 ip，浏览器会进行一个系统调用，查询 hosts 文件。如果找到，直接返回 ip，否则下一步。

进行 1 和 2 本地查询无果，只能借助于网络，路由器一般都会有自己的 DNS 缓存，ISP 服务商 DNS 缓存，这时一般都能够得到相应的 ip，如果还是无果，只能借助于 DNS 递归解析了。

这时 ISP 的 DNS 服务器就会开始从根域名服务器开始递归搜索，从.com 顶级域名服务器，到 baidu 的域名服务器。

到这里，浏览器就获得网络 ip，在 DNS 解析过程中，常常解析出不同的 IP。

第二步，浏览器于网站建立 TCP 连接

浏览器利用 ip 直接网站主机通信，浏览器发出 TCP 连接请求，主机返回 TCP 应答报文，浏览器收到应答报文发现 ACK 标志位为 1，表示连接请求确认，浏览器返回 TCP（）确认报文，主机收到确认报文，三次握手，TCP 连接建立完成。

第三步， 浏览器发起默认的 GET 请求

浏览器向主机发起一个 HTTP-GET 方法报文请求，请求中包含访问的 URL，也就是 http://www.baidu.com/还有 User-Agent 用户浏览器操作系统信息，编码等，值得一提的是 Accep-Encoding 和 Cookies 项。Accept-Encoding 一般采用 gzip，压缩之后传输 html 文件，Cookies 如果是首次访问，会提示服务器简历用户缓存信息，如果不是，可以利用 Cookies 对应键值，找到相应缓存，缓存里面存放着用户名，密码和一些用户设置项

第四步，显示页面或返回其他

返回状态码 200 OK，表示服务器可以响应请求，返回报文，由于在报头中 Content-type 为“text/html”,浏览器以 HTML 形式呈现，而不是下载文件。但是对于大型网站存在多个主机站点,往往不会直接返回请求页面,而是重定向。返回的状态码就不是 200 OK， 而是 301,302 以 3 开头的重定向吗。浏览器在获取了重定向响应后，在响应报文中 Location 项找到重定向地址，浏览器重新第一步访问即可。

78. 对 react 和 vue 的了解？

React 是一个用于构建用户界面的 js 库 React 主要用于构建 UI， 是 MVC 中的 V（视图）拥有较高的性能，代码逻辑非常简单

Vue 是一套用于构建用户界面的渐进式框架 Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。

第三章、数据库

79. 什么是 SQL 注入，如何杜绝？

通过把 SQL 命令插入到 Web 表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。

1.对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.不要使用动态拼装 SQL，可以使用参数化的 SQL 或者直接使用存储过程进行数据查询存取。

3.不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者 hash 掉密码和敏感的信息。

5.应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装，把异常信息存放在独立的表中。

80. 如何查找 MySQL 中查询慢的 SQL 语句

1，slow_query_log

这个参数设置为 ON，可以捕获执行时间超过一定数值的 SQL 语句。

2，long_query_time

当 SQL 语句执行时间超过此数值时，就会被记录到日志中，建议设置为 1 或者更短。

81. Mysql 数据库中的事务？

1、事务的原子性：一组事务，要么成功；要么撤回。
2、稳定性 ：有非法数据（外键约束之类），事务撤回。
3、隔离性：事务独立运行。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的 100%隔离，需要牺牲速度。
4、可靠性：软、硬件崩溃后，InnoDB 数据表驱动会利用日志文件重构修改。可靠性和高速度不可兼得， innodb_flush_log_at_trx_commit 选项 决定什么时候把事务保存到日志里。
MYSQL 事务处理主要有两种方法：
1、用 BEGIN, ROLLBACK, COMMIT 来实现
BEGIN 开始一个事务
ROLLBACK 事务回滚
COMMIT 事务确认
2、直接用 SET 来改变 MySQL 的自动提交模式:
SET AUTOCOMMIT=0 禁止自动提交
SET AUTOCOMMIT=1 开启自动提交

82. 数据库备份怎么做，服务器挂了怎么办？

备份数据库

```
shell> mysqldump -h host -u root -p dbname >dbname_backup.sql
```

恢复数据库

```
shell> mysqladmin -h myhost -u root -p create dbname
shell> mysqldump -h host -u root -p dbname < dbname_backup.sql
```

83. 优化数据库？提高数据库的性能？

1、对语句的优化
①用程序中，保证在实现功能的基础上，尽量减少对数据库的访问次数；通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担；
②能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用 SQL

时，尽量把使用的索引放在选择的首列；算法的结构尽量简单；

③在查询时，不要过多地使用通配符如 SELECT * FROM T1 语句，要用到几列就选择几列如：SELECT COL1,COL2 FROM T1；

④在可能的情况下尽量限制尽量结果集行数如：SELECT TOP 300 COL1,COL2,COL3 FROM T1,因为某些情况下用户是不需要那么多的数据的。

⑤不要在应用中使用数据库游标，游标是非常有用的工具，但比使用常规的、面向集的 SQL 语句需要更大的开销；按照特定顺序提取数据的查找。

2、 避免使用不兼容的数据类型。

数据类型的不兼容可能使优化器无法执行一些本来可以进行的优化操作。

例如： SELECT name FROM employee WHERE salary ＞ 60000

应当在编程时将整型转化成为钱币型,而不要等到运行时转化。

若在查询时强制转换，查询速度会明显减慢。

3、 尽量避免在 WHERE 子句中对字段进行函数或表达式操作。

若进行函数或表达式操作，将导致引擎放弃使用索引而进行全表扫描。

4、 避免使用!=或＜＞、IS NULL 或 IS NOT NULL、IN ，NOT IN 等这样的操作符

5、 尽量使用数字型字段

6、 合理使用 EXISTS,NOT EXISTS 子句。

7、 尽量避免在索引过的字符数据中，使用非打头字母搜索。

8、 分利用连接条件

9、 消除对大型表行数据的顺序存取

10、 避免困难的正规表达式

11、 使用视图加速查询

12、 能够用 BETWEEN 的就不要用 IN

13、 DISTINCT 的就不用 GROUP BY

14、 部分利用索引

15、 能用 UNION ALL 就不要用 UNION

16、 不要写一些不做任何事的查询

17、 尽量不要用 SELECT INTO 语句

18、 必要时强制查询优化器使用某个索引

19、 虽然 UPDATE、DELETE 语句的写法基本固定，但是还是对 UPDATE 语句给点建议：

a) 尽量不要修改主键字段。

b) 当修改 VARCHAR 型字段时，尽量使用相同长度内容的值代替。

c）尽量最小化对于含有 UPDATE 触发器的表的 UPDATE 操作。

d）避免 UPDATE 将要复制到其他数据库的列。

e）避免 UPDATE 建有很多索引的列。

f）避免 UPDATE 在 WHERE 子句条件中的列

84. 索引怎么建立？原理是什么？

在表格上面创建某个一个唯一的索引。唯一的索引意味着两个行不能拥有相同的索引值。

CREATE UNIQUE INDEX 索引名称

ON 表名称 (列名称)

"列名称" 规定你需要索引的列

原理：索引问题就是一个查找问题。

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B 树及其变种 B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

85. 与、或、亦或的优先级（and、or、xor）

not（非） 和 xor （异或）优先级高,因为他们是单运算符.

and 和 or 优先级低,因为他们是双运算符,就是说需要两边都有数据才算.

86. 什么是死锁？

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。由于资源占用是互斥的，当某个进程提出申请资源后，使得有关进程在无外力协助下，永远分配不到必需的资源而无法继续运行，这就产生了一种特殊现象死锁。

87. redis 数据库，内容是以何种结构存放在 ridis 中的？

String(字符串),Hash(哈希),List(列表),Set(集合)及 zset(sortedset:有序集合

88. innerdb 和 MyIASM 的区别以及适用场景？

1. InnoDB 支持事务，MyISAM 不支持，对于 InnoDB 每一条 SQL 语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条 SQL 语言放在 begin 和 commit 之间，组成一个事务；

2. InnoDB 支持外键，而 MyISAM 不支持。对一个包含外键的 InnoDB 表转为 MYISAM 会失败；

3. InnoDB 是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。而 MyISAM 是非聚集索引，数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。

4. InnoDB 不保存表的具体行数，执行 select count(*) from table 时需要全表扫描。而 MyISAM 用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

5. Innodb 不支持全文索引，而 MyISAM 支持全文索引，查询效率上 MyISAM 要高；

如何选择：

1. 是否要支持事务，如果要请选择 innodb，如果不需要可以考虑 MyISAM；

2. 如果表中绝大多数都只是读查询，可以考虑 MyISAM，如果既有读写也挺频繁，请使用 InnoDB。

3. 系统奔溃后，MyISAM 恢复起来更困难，能否接受；

4. MySQL5.5 版本开始 Innodb 已经成为 Mysql 的默认引擎(之前是 MyISAM)，说明其优势是有目共睹的，如果你不知道用什么，那就用 InnoDB，至少不会差。

89. 如果每天有 5T 的用户数据量需要存入 mysql 中，怎么优化数据库？

1、尽量使数据库一次性写入 Data File

2、减少数据库的 checkpoint 操作

3、程序上尽量缓冲数据，进行批量式插入与提交

4、减少系统的 IO 冲突

90. 在关系型数据库中，索引（index）存在的意义是什么？BTree

索引和 Hash 索引的优缺点各是什么？

1.通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

2.可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

3.可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

4.在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

5.通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

BTree 索引

BTree（多路搜索树，并不是二叉的）是一种常见的数据结构。使用 BTree 结构可以显著减少定位记录时所经历的中间过程，从而加快存取速度。这个数据结构一般用于数据库的索引，综合效率较高。

不适合：

单列索引的列不能包含 null 的记录，复合索引的各个列不能包含同时为 null 的记录，否则会全表扫描；

不适合键值较少的列（重复数据较多的列）；

前导模糊查询不能利用索引(like '%XX'或者 like '%XX%')

Hash 散列索引

Hash 散列索引是根据 HASH 算法来构建的索引。虽然 Hash 索引效率高，但是 Hash 索引本身由于其特殊性也带来了很多限制和弊端，主要有以下这些。

适合：

精确查找非常快（包括= <> 和 in），其检索效率非常高，索引的检索可以一次定位，不像 BTree 索引需要从根节点到枝节点，所以 Hash 索引的查询效率要远高于 B-Tree 索引。

不适合：

不适合模糊查询和范围查询（包括 like，>，<，between……and 等），由于 Hash 索引比较的是进行 Hash 运算之后的 Hash 值，所以它只能用于等值的过滤，不能用于基于范围的过滤，因为经过相应的 Hash 算法处理之后的 Hash 值的大小关系，并不能保证和 Hash 运算前完全一样；

不适合排序，数据库无法利用索引的数据来提升排序性能，同样是因为 Hash 值的大小不确定；

复合索引不能利用部分索引字段查询，Hash 索引在计算 Hash 值的时候是组合索引键合并后再一起计算 Hash 值，而不是单独计算 Hash 值，所以通过组合索引的前面一个或几个索引键进行查询的时候，Hash 索引也无法被利用。

同样不适合键值较少的列（重复值较多的列）；

91. nosql 与 sql 区别，以及使用场景？

1、存储方式

SQL 数据存在特定结构的表中；而 NoSQL 则更加灵活和可扩展，存储方式可以省是 JSON 文档、哈希表或者其他方式。

2、表/数据集合的数据的关系

在 SQL 中，必须定义好表和字段结构后才能添加数据，例如定义表的主键(primary key),索引(index),触发器(trigger),存储过程(stored procedure)等。表结构可以在被定义之后更新，但是如果有比较大的结构变更的话就会变得比较复杂。在 NoSQL 中，数据可以在任何时候任何地方添加，不需要先定义表。NoSQL 也可以在数据集中建立索引。

3、外部数据存储

SQL 中如何需要增加外部关联数据的话，规范化做法是在原表中增加一个外键，关联外部数据表。在 NoSQL 中除了这种规范化的外部数据表做法以外，我们还能用如下的非规范化方式把外部数据直接放到原数据集中，以提高查询效率。缺点也比较明显，更新审核人数据的时候将会比较麻烦。

4、SQL 中的 JOIN 查询

SQL 中可以使用 JOIN 表链接方式将多个关系数据表中的数据用一条简单的查询语句查询出来。NoSQL 暂未提供类似 JOIN 的查询方式对多个数据集中的数据做查询。所以大部分 NoSQL 使用非规范化的数据存储方式存储数据。

5、数据耦合性

SQL 中不允许删除已经被使用的外部数据以保证数据完整性。而 NoSQL 中则没有这种强耦合的概念，可以随时删除任何数据。

6、事务

SQL 中如果多张表数据需要同批次被更新，即如果其中一张表更新失败的话其他表也不能更新成功。这种场景可以通过事务来控制，可以在所有命令完成后再统一提交事务。而 NoSQL 中没有事务这个概念，每一个数据集的操作都是原子级的。

7、查询性能

在相同水平的系统设计的前提下，因为 NoSQL 中省略了 JOIN 查询的消耗，故理论上性能上是优于 SQL 的。

92. Redis 的并发竞争问题怎么解决？

方案一：可以使用独占锁的方式，类似操作系统的 mutex 机制。（网上有例子，http://blog.csdn.net/black_ox/article/details/48972085 不过实现相对复杂，成本较高）

方案二：使用乐观锁的方式进行解决（成本较低，非阻塞，性能较高）

93. MongoDB

MongoDB 是一个面向文档的数据库系统。使用 C++编写，不支持 SQL，但有自己功能强大的查询语法。

MongoDB 使用 BSON 作为数据存储和传输的格式。BSON 是一种类似 JSON 的二进制序列化文档，支持嵌套对象和数组。

MongoDB 很像 MySQL，document 对应 MySQL 的 row，collection 对应 MySQL 的 table

应用场景：

1.网站数据：mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

2.缓存：由于性能很高，mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由 mongo 搭建的持久化缓存可以避免下层的数据源过载。

3.大尺寸、低价值的数据：使用传统的关系数据库存储一些数据时可能会比较贵，在此之前，很多程序员往往会选择传统的文件进行存储。

4.高伸缩性的场景：mongo 非常适合由数十或者数百台服务器组成的数据库。

5.用于对象及 JSON 数据的存储：mongo 的 BSON 数据格式非常适合文档格式化的存储及查询。

6.重要数据：mysql，一般数据：mongodb，临时数据：memcache

7.对于关系数据表而言，mongodb 是提供了一个更快速的视图 view；而对于 PHP 程序而言，mongodb 可以作为一个持久化的数组来使用，并且这个持久化的数组还可以支持排序、条件、限制等功能。

8.将 mongodb 代替 mysql 的部分功能，主要一个思考点就是：把 mongodb 当作 mysql 的一个 view（视图），view 是将表数据整合成业务数据的关键。比如说对原始数据进行报表，那么就要先把原始数据统计后生成 view，在对 view 进行查询和报表。

不适合的场景：

a.高度事物性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。

b.传统的商业智能应用：针对特定问题的 BI 数据库会对产生高度优化的

查询方式。对于此类应用，数据仓库可能是更合适的选择。

c.需要 SQL 的问题

d.重要数据，关系数据

优点:弱一致性（最终一致），更能保证用户的访问速度

文档结构的存储方式，能够更便捷的获取数

内置 GridFS，高效存储二进制大对象（比如照片和视频）

支持复制集、主备、互为主备、自动分片等特性

动态查询

全索引支持,扩展到内部对象和内嵌数组

缺点：不支持事务

MongoDB 占用空间过大

维护工具不够成熟

94. Redis、mongodb 优缺点

MongoDB 和 Redis 都是 NoSQL，采用结构型数据存储。二者在使用场景中，存在一定的区别，这也主要由于二者在内存映射的处理过程，持久化的处理方法不同。MongoDB 建议集群部署，更多的考虑到集群方案，Redis 更偏重于进程顺序写入，虽然支持集群，也仅限于主-从模式.

Redis 优点：

1 读写性能优异

2 支持数据持久化，支持 AOF 和 RDB 两种持久化方式

3 支持主从复制，主机会自动将数据同步到从机，可以进行读写分离。

4 数据结构丰富：除了支持 string 类型的 value 外还支持 string、hash、set、sortedset、list 等数据结构。

缺点：

1 Redis 不具备自动容错和恢复功能，主机从机的宕机都会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的 IP 才能恢复。

2 主机宕机，宕机前有部分数据未能及时同步到从机，切换 IP 后还会引入数据不一致的问题，降低了系统的可用性。

3 Redis 的主从复制采用全量复制，复制过程中主机会 fork 出一个子进程对内存做一份快照，并将子进程的内存快照保存为文件发送给从机，这一过程需要确保主机有足够多的空余内存。若快照文件较大，对集群的服务能力会产生较大的影响，而且复制过程是在从机新加入集群或者从机和主机网络断开重连时都会进行，也就是网络波动都会造成主机和从机间的一次全量的数据复制，这对实际的系统运营造成了不小的麻烦。

4 Redis 较难支持在线扩容，在集群容量达到上限时在线扩容会变得很复杂。为避免这一问题，运维人员在系统上线时必须确保有足够的空间，这对资源造成了很大的浪费。

优点:弱一致性（最终一致），更能保证用户的访问速度

文档结构的存储方式，能够更便捷的获取数

内置 GridFS，高效存储二进制大对象（比如照片和视频）

支持复制集、主备、互为主备、自动分片等特性

动态查询

全索引支持,扩展到内部对象和内嵌数组

缺点：不支持事务

MongoDB 占用空间过大

维护工具不够成熟

95. 数据库负载均衡

负载均衡集群是由一组相互独立的计算机系统构成，通过常规网络或专用网络进行连接，由路由器衔接在一起，各节点相互协作、共同负载、均衡压力，对客户端来说，整个群集可以视为一台具有超高性能的独立服务器。

1、实现原理

实现数据库的负载均衡技术，首先要有一个可以控制连接数据库的控制端。在这里，它截断了数据库和程序的直接连接，由所有的程序来访问这个中间层，然后再由中间层来访问数据库。这样，我们就可以具体控制访问某个数据库了，然后还可以根据数据库的当前负载采取有效的均衡策略，来调整每次连接到哪个数据库。

2、实现多据库数据同步

对于负载均衡，最重要的就是所有服务器的数据都是实时同步的。

数据条数很少，数据内容也不大，则直接同步数据

数据条数很少，但是里面包含大数据类型，比如文本，二进制数据等，则先对数据进行压缩然后再同步，从而减少网络带宽的占用和传输所用的时间。

数据条数很多，此时中间件会拿到造成数据变化的 SQL 语句， 然后对 SQL 语句进行解析，分析其执行计划和执行成本，并选择是同步数据还是同步 SQL 语句到其他的数据库中。此种情况应用在对表结构进行调整或者批量更改数据的时候非常有用。

3、优缺点

优点：

(1) 扩展性强：当系统要更高数据库处理速度时，只要简单地增加数据库服务器就 可以得到扩展。

(2) 可维护性：当某节点发生故障时，系统会自动检测故障并转移故障节点的应用，保证数据库的持续工作。

(3) 安全性：因为数据会同步的多台服务器上，可以实现数据集的冗余，通过多份数据来保证安全性。另外它成功地将数据库放到了内网之中，更好地保护了数据库的安全性。

(4) 易用性：对应用来说完全透明，集群暴露出来的就是一个 IP

缺点：

(1) 不能够按照 Web 服务器的处理能力分配负载。

(2) 负载均衡器(控制端)故障，会导致整个数据库系统瘫痪。

96. 怎样解决海量数据的存储和访问造成系统设计瓶颈的问题？

水平切分数据库：可以降低单台机器的负载，同时最大限度的降低了宕机造成的损失；分库降低了单点机器的负载；分表，提高了数据操作的效率，

负载均衡策略：可以降低单台机器的访问负载，降低宕机的可能性；

集群方案：解决了数据库宕机带来的单点数据库不能访问的问题；

读写分离策略：最大限度了提高了应用中读取数据的速度和并发量；

97. 怎样解决数据库高并发的问题？

解决数据库高并发：

分表分库

数据库索引

redis 缓存数据库

读写分离

负载均衡集群：将大量的并发请求分担到多个处理节点。由于单个处理节点的故障不影响整个服务，负载均衡集群同时也实现了高可用性。

第四章、Web 后端

98. Django 创建项目后，项目文件夹下的组成部分（对 mvt 的理解）？

项目文件夹下的组成部分：

manage.py 是项目运行的入口，指定配置文件路径。

与项目同名的目录，包含项目的配置文件

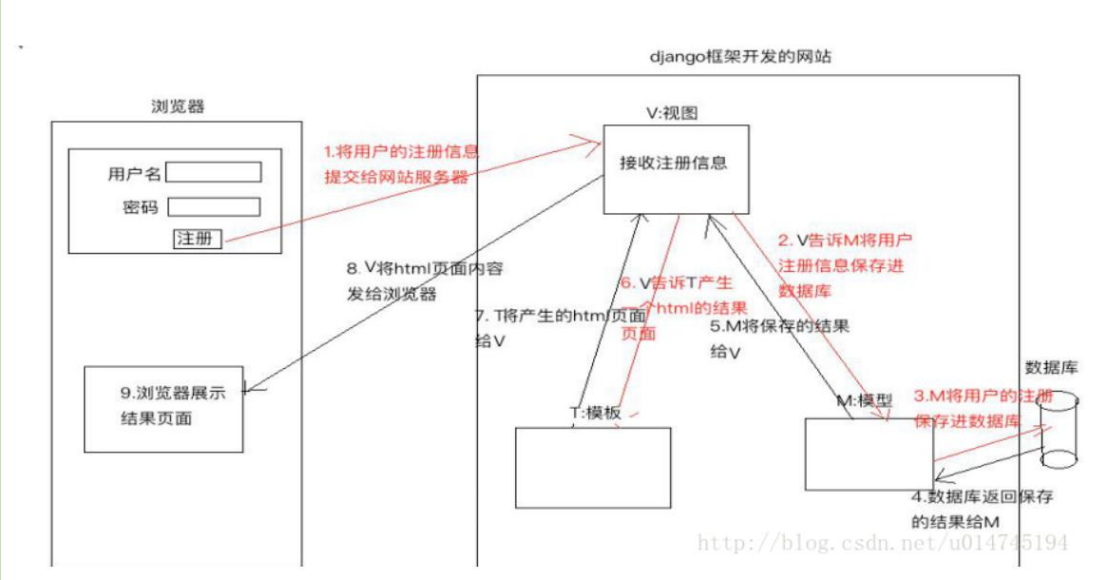
___init.py 是一个空文件，作用是这个目录可以被当作包使用。

settings.py 是项目的整体配置文件。

urls.py 是项目的 URL 配置文件。

wsgi.py 是项目与 WSGI 兼容的 Web 服务器入口

对 MVT 的理解



M 全拼为 Model，与 MVC 中的 M 功能相同，负责和数据库交互，进行数据处理。

V 全拼为 View，与 MVC 中的 C 功能相同，接收请求，进行业务处理，返回应答。

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。

99. 对 uWSGI 的理解？

uWSGI 是一个 Web 服务器，它实现了 WSGI 协议、uwsgi 协议、http 等协议。

WSGI 是一种通信协议，它用于定义传输信息的类型，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。

nginx 具备优秀的静态内容处理能力,然后将动态内容转发给 uWSGI 服务器，这样可以达到很好的客户端响应。

uwsgi 的参数：

-M 开启 Master 进程

-p 4 开启 4 个进程

-s 使用的端口或者 socket 地址

-d 使用 daemon 的方式运行，注意，使用-d 后，需要加上 log 文件地址，比如-d /var/log/uwsgi.log

-R 10000 开启 10000 个进程后，自动 respawn（复位）下

-t 30 设置 30s 的超时时间，超时后，自动放弃该链接

- limit-as 32 将进程的总内存量控制在 32M

-x 使用配置文件模式

100. 怎样测试 Django 框架中的代码？

在单元测试方面，Django 继承 python 的 unittest.TestCase 实现了自己的 django.test.TestCase，编写测试用 例通常从这里开始。测试代码通常位于 app 的 tests.py 文件中(也可以在 models.py 中编写，一般不建议）。在 Django 生成的 depotapp 中，已经包含了这个文件，并且其中包含了一个测试用例的样例：

python manage.py test：执行所有的测试用例

python manage.py test app_name，执行该 app 的所有测试用例

python manage.py test app_name.case_name：执行指定的测试用例

一些测试工具：unittest 或者 pytest

101. 简单说一下，用户访问一个 url 的过程（从浏览器输入 URL 到显示界面，中间发生了什么？）

第一步，解析域名，找到 IP

（1）浏览器会缓存 DNS 一段时间，一般 2-30 分钟不等。如果有缓存，直接返回 IP，否则下一步。

（2）缓存中无法找到 IP，浏览器会进行一个系统调用，查询 hosts 文件。如果找到，直接返回 IP，否则下一步。（在计算机本地目录 etc 下有一个 hosts 文件，hosts 文件中保存有域名与 IP 的对应解析，通常也可以修改 hosts 科学上网或破解软件。）

（3）进行了（1）（2）本地查询无果，只能借助于网络。路由器一般都会有自己的 DNS 缓存，ISP 服务商 DNS 缓存，这时一般都能够得到相应的 IP。如果还是无果，只能借助于 DNS 递归解析了。

（4)这时,ISP 的 DNS 服务器就会开始从根域名服务器开始递归搜索,从.com 顶级域名服务器，到 baidu 的域名服务器。

到这里，浏览器就获得了 IP。在 DNS 解析过程中，常常会解析出不同的 IP。比如，电信的是一个 IP，网通的是另一个 IP。这是采取了智能 DNS 的结果， 降低运营商间访问延时，在多个运营商设置主机房，就近访问主机。电信用户返回电信主机 IP，网通用户返回网通主机 IP。当然，劫持 DNS，也可以屏蔽掉一 部分网点的访问，某防火长城也加入了这一特性。

第二步，浏览器与网站建立 TCP 连接

浏览器利用 IP 直接与网站主机通信。浏览器发出 TCP（SYN 标志位为 1）连接请求，主机返回 TCP（SYN，ACK 标志位均为 1）应答报文，浏览器收到 应答报文发现 ACK 标志位为 1，表示连接请求确认。浏览器返回 TCP（）确认报文，主机收到确认报文，三次握手，TCP 链接建立完成。

第三步，浏览器发起默认的 GET 请求

浏览器向主机发起一个 HTTP-GET 方法报文请求。请求中包含访问的 URL，也就是 http://www.baidu.com/ ，还有 User-Agent 用户浏览器操作系统信息，编码等。值得一提的是 Accep-Encoding 和 Cookies 项。Accept- Encoding 一般采用 gzip，压缩之后传输 html 文件。Cookies 如果是首次访问，会提示服务器建立用户缓存信息，如果不是，可以利用 Cookies 对应键值，找到相应缓存，缓存里面存放着用户名，密码和一些用户设置项。

第四步，显示页面或返回其他

返回状态码 200 OK，表示服务器可以相应请求，返回报文，由于在报头中 Content-type 为“text/html”，浏览器以 HTML 形式呈现，而不是下载文件。

但是，对于大型网站存在多个主机站点，往往不会直接返回请求页面，而是重定向。返回的状态码就不是 200 OK，而是 301,302 以 3 开头的重定向码，浏览器在获取了重定向响应后，在响应报文中 Location 项找到重定向地址，浏览器重新第一步访问即可。

补充一点的就是，重定向是为了负载均衡或者导入流量，提高 SEO 排名。利用一个前端服务器接受请求，然后负载到不同的主机上，可以大大提高站点的业务并发 处理能力；重定向也可将多个域名的访问，集中到一个站点；由于 baidu.com，www.baidu.com 会被搜索引擎认为是两个网站，照成每个的链 接数都会减少从而降低排名，永久重定向会将两个地址关联起来，搜索引擎会认为是同一个网站，从而提高排名。

102. 有过部署经验？用的什么技术？可以满足多少压力？

1.有部署经验，在阿里云服务器上部署的
2.技术有：nginx + uwsgi 的方式来部署 Django 项目
3.无标准答案（例：压力测试一两千）

103. 项目中使用了什么调试?

一般用 IDE 进行调试：例如：Pycharm 做断点调试
or
1.在对应位置使用 print 和 logging
2. sentry 第三方调试库，主要应用在 django 项目
3. ipdb（pdb）可以设置断点、单步调试、进入函数调试、查看当前代码、查看栈片段、动态改变变量的值等
4，直接让你想要调试的位置让它先跑个异常

104. 第三方支付是如何实现的？

聚合支付：ping++，付钱拉，Paymax...
直接对接：支付宝，微信，qq 钱包，银联，百度钱包...
建议：去该类官网了解接口相关知识

1、简单加密

目的是为了保证上传的参数信息没有被篡改，主要分成三部分
接口参数 ： 需要和第三方对接的参数
加密类型 ： 使用什么类型加密，一般为 MD5
加密密文 ： 使用接口参数和第三方生成的 Code 值(固定 salt),进行 MD5 加密成密文
Md5 作为数字签名 H(A)=P 已知 A P，在特殊情况下可以伪造 A1 满足 H(A1)=p

2、生成加密的密钥 Key

随机生成 16 位的加密密钥 Key，用于对上一步的内容进行对称加密

3、使用证书加密密钥 Key

对随机生成的密钥 key 进行加密。防止在传输过程中被截获破解。

使用了三种加密算法：
摘要算法：验证原文是否被篡改
对称加密算法：使用密钥对原文进行加密(AES)
非对称加密算法：对密钥进行分发

发送方：

入参 + H(入参+分配的 code)= P 得到键值对 ParamMap
AES（ParamMap + 随机生成的 Key ）进行对称加密得到 DecodeString
随机生成的 Key + 私钥证书加密， 得到 EnCodeKey
Http 发送请求，将 DecodeString 和 EnCodeKey 发送给第三方支付公司

接收方：

证书加密的随机数，使用私钥解密得到 Akey
AES(AKey) = sourceCode("入参","P")
验证：H(sourceCode + code) = P

105. django 关闭浏览器，怎样清除 cookies 和 session？

删除 cookie:

```
Response.delete_cookie("username")
```

删除 session:

```
del req.session['username']
```

Session 依赖于 Cookie,如果浏览器不能保存 cooki 那么 session 就失效了。因为它需要浏览器的 cooki 值去 session 里做对比。session 就是用来在服务器端保存用户的会话状态。

cookie 可以有过期时间，这样浏览器就知道什么时候可以删除 cookie 了。如果 cookie 没有设置过期时间，当用户关闭浏览器的时候，cookie 就自动过期了。 你可以改变 SESSION_EXPIRE_AT_BROWSER_CLOSE 的设置来控制 session 框架的这一行为。缺省情况下， SESSION_EXPIRE_AT_BROWSER_CLOSE 设置为 False ， 这 样 ， 会 话 cookie 可 以 在 用 户 浏 览 器 中 保 持 有 效 达 SESSION_COOKIE_AGE 秒（缺省设置是两周，即 1,209,600 秒）。 如果你不想用户每次打开浏览器都必须重新登陆的话，用这个参数来帮你。如果 SESSION_EXPIRE_AT_BROWSER_CLOSE 设置为 True ，当浏览器关闭时，Django 会使 cookie 失效。

SESSION_COOKIE_AGE：设置 cookie 在浏览器中存活的时间

106. 代码优化从哪些方面考虑？有什么想法？

1、优化算法时间

算法的时间复杂度对程序的执行效率影响最大，在 Python 中可以通过选择合适的数据结构来优化时间复杂度，如 list 和 set 查找某一个元素的时间复杂度分别是 O(n)和 O(1)。不同的场景有不同的优化方式，总得来说，一般有分治，分支界限，贪心，动态规划等思想。

2、循环优化

每种编程语言都会强调需要优化循环。当使用 Python 的时候，你可以

依靠大量的技巧使得循环运行得更快。然而，开发者经常漏掉的一个方法是：避免在一个循环中使用点操作。例如，考虑下面的代码：

每一次你调用方法 str.upper，Python 都会求该方法的值。然而，如果你用一个变量代替求得的值，值就变成了已知的，Python 就可以更快地执行任务。优化循环的关键，是要减少 Python 在循环内部执行的工作量，因为 Python 原生的解释器在那种情况下，真的会减缓执行的速度。

（注意：优化循环的方法有很多，这只是其中的一个。例如，许多程序员都会说，列表推导是在循环中提高执行速度的最好方式。这里的关键是，优化循环是程序取得更高的执行速度的更好方式之一。）

3、函数选择

在循环的时候使用 xrange 而不是 range；使用 xrange 可以节省大量的系统内存，因为 xrange()在序列中每次调用只产生一个整数元素。而 range()将直接返回完整的元素列表，用于循环时会有不必要的开销。在 python3 中 xrange 不再存在，里面 range 提供一个可以遍历任意长度的范围的 iterator。

4、并行编程

因为 GIL 的存在，Python 很难充分利用多核 CPU 的优势。但是，可以通过内置的模块 multiprocessing 实现下面几种并行模式：

多进程：对于 CPU 密集型的程序，可以使用 multiprocessing 的 Process,Pool 等封装好的类，通过多进程的方式实现并行计算。但是因为进程中的通信成本比较大，对于进程之间需要大量数据交互的程序效率未必有大的提高。

多线程：对于 IO 密集型的程序，multiprocessing.dummy 模块使用 multiprocessing 的接口封装 threading，使得多线程编程也变得非常轻松(比如可以使用 Pool 的 map 接口，简洁高效)。

分布式：multiprocessing 中的 Managers 类提供了可以在不同进程之共享数据的方式，可以在此基础上开发出分布式的程序。

不同的业务场景可以选择其中的一种或几种的组合实现程序性能的优化。

5、使用性能分析工具

除了上面在 ipython 使用到的 timeit 模块，还有 cProfile。cProfile 的使用方式也非常简单：python-mcProfilefilename.py，filename.py 是要运行程序的文件名，可以在标准输出中看到每一个函数被调用的次数和运行的时间，从而找到程序的性能瓶颈，然后可以有针对性地优化。

6、set 的用法

set 的 union，intersection，difference 操作要比 list 的迭代要快。因此如果涉及到求 list 交集，并集或者差的问题可以转换为 set 来操作。

7、PyPy

PyPy 是用 RPython(CPython 的子集)实现的 Python，根据官网的基准测试数据，它比 CPython 实现的 Python 要快 6 倍以上。快的原因是使用了 Just-in-Time(JIT)编译器，即动态编译器，与静态编译器(如 gcc,javac 等)不同，它是利用程序运行的过程的数据进行优化。由于历史原因，目前 pypy 中还保留着 GIL，不过正在进行的 STM 项目试图将 PyPy 变成没有 GIL 的 Python。

如果 python 程序中含有 C 扩展(非 cffi 的方式)，JIT 的优化效果会大打折扣，甚至比 CPython 慢（比 Numpy）。所以在 PyPy 中最好用纯 Python 或使用 cffi 扩展。

107. 有用过 docker 吗？docker 的机制、原理(外壳和镜像的区别)，说说 docker 与 VMware 虚拟化实现的不同？？

1. docker 概念

Docker 是一个开源的引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。开发者在笔记本上编译测试通过的容器可以批量地在生产环境中部署，包括 VMs（虚拟机）、bare metal、OpenStack 集群和其他的基础应用平台。

Docker 通常用于如下场景：

web 应用的自动化打包和发布；

自动化测试和持续集成、发布；

在服务型环境中部署和调整数据库或其他的后台应用；

从头编译或者扩展现有的 OpenShift 或 Cloud Foundry 平台来搭建自己的 PaaS 环境。

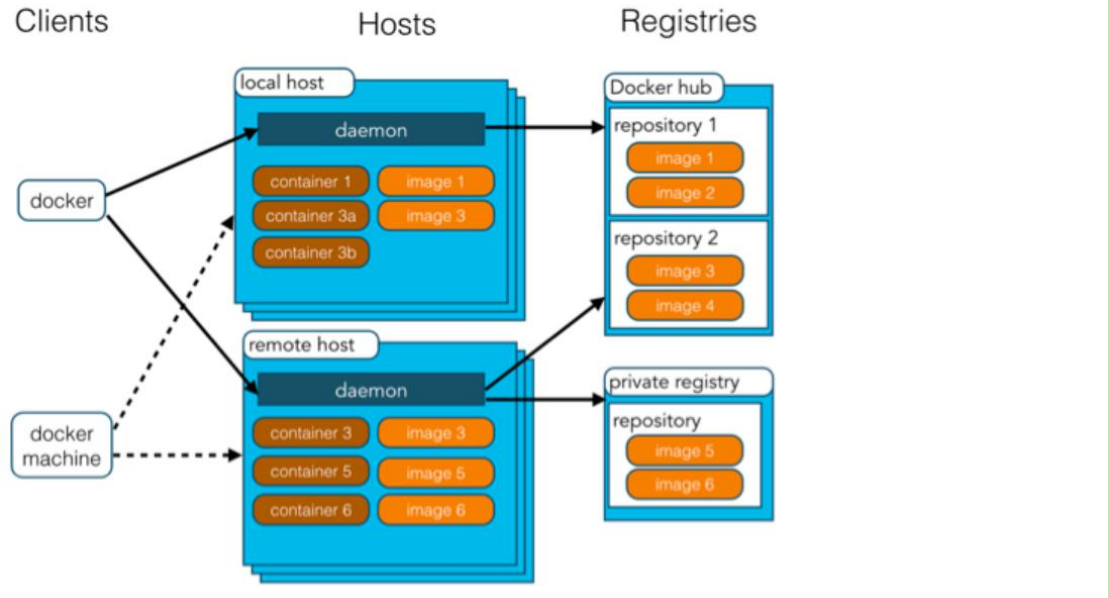
docker 的机制和原理

Docker 使用客户端-服务器 (C/S) 架构模式，使用远程 API 来管理和创建 Docker 容器。

Docker 容器通过 Docker 镜像来创建。

容器与镜像的关系类似于面向对象编程中的对象与类。

Docker	面向对象
容器	对象
镜像	类



Docker 镜像 (Images)	Docker 镜像是用于创建 Docker 容器的模板。
Docker 容器 (Container)	容器是独立运行的一个或一组应用。
Docker 客户端 (Client)	Docker 客户端通过命令行或者其他工具使用 Docker API (https://docs.docker.com/reference/api/docker_remote_api) 与 Docker 的守护进程通信。
Docker 主机 (Host)	一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。
Docker 仓库 (Registry)	Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。 Docker Hub(https://hub.docker.com) 提供了庞大的镜像集合供使用。
Docker Machine	Docker Machine是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、Microsoft Azure。

docker 和 VMware 的区别？

各种虚拟机技术开启了云计算时代；而 Docker，作为下一代虚拟化技术，正在改变我们开发、测试、部署应用的方式

Docker 守护进程可以直接与主操作系统进行通信，为各个 Docker 容器分配资源；它还可以将容器与主操作系统隔离，并将各个容器互相隔离。

虚拟机启动需要数分钟，而 Docker 容器可以在数毫秒内启动。由于没有臃肿的从操作系统，Docker 可以节省大量的磁盘空间以及其他系统资源。

说了这么多 Docker 的优势，大家也没有必要完全否定虚拟机技术，因为两者有不同的使用场景。虚拟机更擅长于彻底隔离整个运行环境。例如，云服务提供商通常采用虚拟机技术隔离不同的用户。而 Docker 通常用于隔离不同的应用，例如前端，后端以及数据库。

108. api 和 sdk 的区别？

API: Application Programming Interface

编写应用程序的接口，抽象概念。

SDK: Software Development Kit

使用某种语言来暴露 API，提供头文件、库文件、工具（通常不是 IDE）和范例。

API 是接口，SDK 是开发包。就是钢笔和书包的差别，SDK 里面可以有很多 API，也可以说是一系列 API 的幸福全家桶，

109. 你的这些项目中印象最深的技术难点？

答案：略...（根据自身实际情况回答）

110. 什么是 QPS?

每秒查询率

QPS（TPS）：每秒钟 request/事务　数量

111. Django 里 QuerySet 的 get 和 filter 方法的区别？

【输入参数】

get 的参数只能是 model 中定义的那些字段，只支持严格匹配

filter 的参数可以是字段，也可以是扩展的 where 查询关键字，如 in，like 等

【返回值】

get 返回值是一个定义的 model 对象

filter 返回值是一个新的 QuerySet 对象，然后可以对 QuerySet 在进行查询返回新的 QuerySet 对象，支持链式操作，QuerySet 一个集合对象，可使用迭代或者遍历，切片等，但是不等于 list 类型(使用一定要注意)

【异常】

get 只有一条记录返回的时候才正常，也就说明 get 的查询字段必须是主键或者唯一约束的字段。当返回多条记录或者是没有找到记录的时候都会抛出异常

filter 有没有匹配的记录都可以

112. 简述 Django 对 HTTP 请求的执行流程。

Django 接收到用户请求的 URL 后，去所有 URL 集合的 urls.py 文件中按照正则匹配目的 url，若匹配到就调用该 URL 的视图函数，执行代码完成相应的功能（对数据库增删改查），返回处理结果的数据，调用相应的模板或构建新页面，在前端页面通过模板语言接收该结果，并将数据展现填充到相应位置，并将构建好的页面直接返回给浏览器，客户端浏览器将页面代码渲染输出为一个完整页面显示给用户

113. 简述 Django 下的（内建的）缓存机制。

Django 自带了一个健壮的缓存系统来保存动态页面这样避免对于每次请求都重新计算。方便起见，Django 提供了不同级别的缓存粒度：可以缓存特定视图的输出、可以仅仅缓存那些很难生产出来的部分、或者可以缓存整个网站

Django 也能很好的配合那些“下游”缓存，比如 Squid 和基于浏览器的缓存。这里有一些缓存不必要直接去控制但是可以提供线索，（via HTTP headers）关于网站哪些部分需要缓存和如何缓存

114. Django 中 Model 的 SlugField 类型字段有什么用途？

SlugField 字段用于生成一个有意义的 URL，一般是将标题转换成 URL，在后台添加时直接使用

115. 过 http 服务器日志如何干掉恶意爬虫；

```
cat uwsgi.log | grep spider -c  #查看有爬虫标志的访问次数
cat uwsgi.log | wc  #查看总页面访问次数
cat uwsgi |grep spider|awk '{print $1}'|sort -n|uniq -c|sort -nr #
```

查看爬虫来源 IP

cat uwsgi.log |awk '{print $1 " " substr($4,14,5)}'|sort -n|uniq -c|sort -nr|head -20 #列出在一分钟内访问网站次数最多的前 20 位 ip 地址

/etc/hosts.deny 中列出的都是要禁止的 IP 地址

116. 对 cookie 与 session 的了解？他们能单独用吗？

Session 采用的是在服务器端保持状态的方案，而 Cookie 采用的是在客户端保持状态的方案。但是禁用 Cookie 就不能得到 Session。

因为 Session 是用 Session ID 来确定当前对话所对应的服务器 Session，而 Session ID 是通过 Cookie 来传递的，禁用 Cookie 相当于失去了 Session ID，也就得不到 Session。

117. ngnix 的正向代理与反向代理

web 开发中，部署方式大致类似。简单来说，使用 Nginx 主要是为了实现分流、转发、负载均衡，以及分担服务器的压力。Nginx 部署简单，内存消耗少，成本低。Nginx 既可以做正向代理，也可以做反向代理。

正向代理：请求经过代理服务器从局域网发出，然后到达互联网上的服务器。

特点：服务端并不知道真正的客户端是谁。

反向代理：请求从互联网发出，先进入代理服务器，再转发给局域网内的服务器。

特点：客户端并不知道真正的服务端是谁。

区别：正向代理的对象是客户端。反向代理的对象是服务端。

118. Jieba 分词

Jieba 分词支持三种分词模式：

精确模式，试图将句子最精确地切开，适合文本分析；

全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；

搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词

功能：

分词，添加自定义词典，关键词提取，词性标注，并行分词，Tokenize：返回词语在原文的起始位置，ChineseAnalyzer for Whoosh 搜索引擎

119. 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。

120. Flask 中正则 URL 的实现？

@app.route('<URL>')中 URL 显式支持 string、int、float、path 4 种类型，隐式支持正则

第一步：写正则类，继承 BaseConverter，将匹配到的值设置为 regex 的值

```
class RegexUrl(BaseConverter):
    def __init__(self, url_map, *args):
        super(RegexUrl, self).__init__(url_map)
        self.regex = args[0]
```

第二步：把正则类赋值给我们定义的正则规则

```
app.url_map.converters['re'] = RegexUrl
```

第三步：在 URL 中使用正则

```
@app.route('/regex/<re("[a-z]{3}"):id>')
def regex111(id):
return 'id:%s'%id
```

121. Flask 中请求上下文和应用上下文的区别和作用？

current_app、g 就是应用上下文

requests、session 就是请求上下文

手动创建上下文的两种方法：

```
with app.app_context()
app = current_app._get_current_object()
```

122. 一个变量后写多个过滤器是如何执行？

{{ expression | filter1 | filter2 | ... }} 即 表达式(expression)使用 filter1 过滤后再使用 filter2 过滤...

123. Flask 中数据库 app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN']设置的作用？

作用：可以配置请求执行完逻辑之后自动提交，而不用我们每次都手动调用 session.commit()

补充：

监听数据库中的数据，当发生改变，就会显示一些内容

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=True
```

显示打印的数据以及 sql 语句，建议不设置，默认为 False

```
app.config['SQLALCHEMY_ECHO'] = True
```

124. 模糊查询和精确查询的区别？

filter()模糊查询:把过滤器添加到原查询上，返回一个新查询

filter_by()精确查询:把等值过滤器添加到原查询上，返回一个新查询

125. 对 flask 蓝图(Blueprint)的理解？

1.蓝图定义：

蓝图 /Blueprint 是 Flask 应用程序 组件化 的方法，可以在一个应用内或跨越多个项目共用蓝图。使用蓝图可以极大地简化大型应用的开发难度，也为 Flask 扩展 提供了一种在应用中注册服务的集中式机制。

2.蓝图的应用场景

1，把一个应用分解为一个蓝图的集合。这对大型应用是理想的。一个项目可以实例化一个应用对象，初始化几个扩展，并注册一集合的蓝图。

2，以 URL 前缀和/或子域名，在应用上注册一个蓝图。 URL 前缀/子域名中的参数即成为这个蓝图下的所有视图函数的共同的视图参数（默认情况下）。

3，在一个应用中用不同的 URL 规则多次注册一个蓝图。

4，通过蓝图提供模板过滤器、静态文件、模板和其它功能。一个蓝图不一定要实现应用或者视图函数。

5，初始化一个 Flask 扩展时，在这些情况中注册一个蓝图。

3.蓝图的缺点

不能在应用创建后撤销注册一个蓝图而不销毁整个应用对象。

4.使用蓝图的三个步骤

1.创建 一个蓝图对象

```
blue = Blueprint("blue",__name__)
```

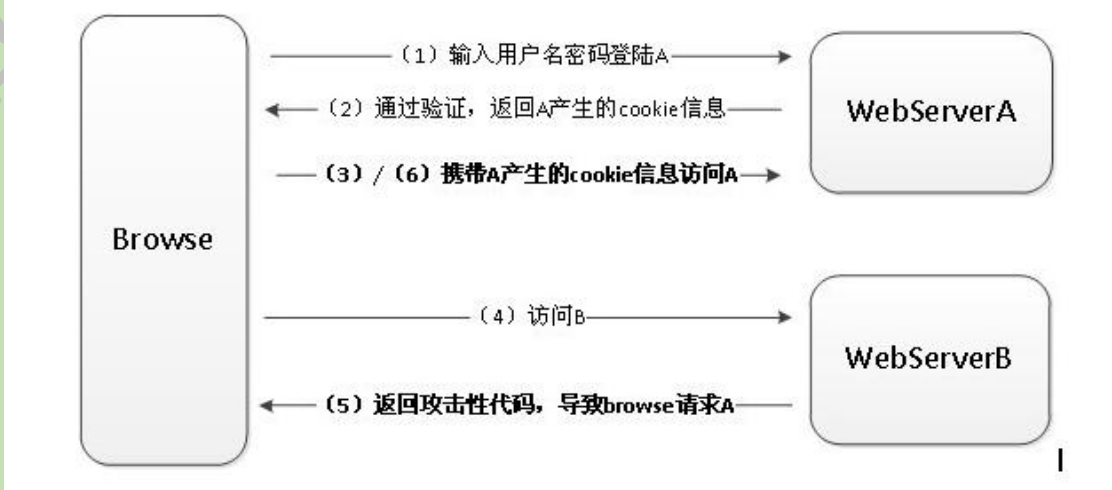
2.在这个蓝图对象上进行 操作 ，例如注册路由、指定静态文件夹、注册模板过滤器...

```
@blue.route('/')
def blue_index():
    return 'Welcome to my blueprint'
```

3.在应用对象上 注册 这个蓝图对象

```
app.register_blueprint(blue,url_prefix='/blue')
```

126. 跨站请求伪造和跨站请求保护的实现？



图中 Browse 是浏览器，WebServerA 是受信任网站/被攻击网站 A，WebServerB 是恶意网站/攻击网站 B。
（1）一开始用户打开浏览器，访问受信任网站 A，输入用户名和密码登陆请求登陆网站 A。
（2）网站 A 验证用户信息，用户信息通过验证后，网站 A 产生 Cookie 信息并返回给浏览器。
（3）用户登陆网站 A 成功后，可以正常请求网站 A。
（4）用户未退出网站 A 之前，在同一浏览器中，打开一个 TAB 访问网站 B。
（5）网站 B 看到有人方式后，他会返回一些攻击性代码。
（6）浏览器在接受到这些攻击性代码后，促使用户不知情的情况下浏览器携带 Cookie（包括 sessionId）信息，请求网站 A。这种请求有可能更新密码，添加用户什么的操作。

从上面 CSRF 攻击原理可以看出，要完成一次 CSRF 攻击，需要被攻击者完成两个步骤：

1.登陆受信任网站 A，并在本地生成 COOKIE。

2.在不登出 A 的情况下，访问危险网站 B。

如果不满足以上两个条件中的一个，就不会受到 CSRF 的攻击，以下情况可能会导致 CSRF：

1.登录了一个网站后，打开一个 tab 页面并访问另外的网站。

2.关闭浏览器了后，本地的 Cookie 尚未过期，你上次的会话还没有已经结束。（事实上，关闭浏览器不能结束一个会话，但大多数人都会错误的认为关闭浏览器就等于退出登录/结束会话了……）

解决办法：就是在表单中添加 from.csrf_token

127. Flask 项目中如何实现 session 信息的写入？

flask 中有三个 session：

第一个：数据库中的 session，例如:db.session.add()

第二个：在 flask_session 扩展中的 session，使用：from flask_session import Session，使用第三方扩展的 session 可以把信息存储在服务器中，客户端浏览器中只存储 sessionid

第三个：flask 自带的 session，是一个请求上下文， 使用：from flask import session。自带的 session 把信息加密后都存储在客户端的浏览器 cookie 中

128. 在虚拟机中部署项目，在 windows 下如何访问？

以 Django 项目为例，启动测试服务器时输入 python manage.py runserver 0.0.0.0:8000 或者 python manage.py runserver ip:port (ip 为虚拟机的 ip 地址，端口号自己定义)，然后在主机浏览器地址栏中输入 ip:port(ip 为虚拟机的 ip 地址，端口号为前文定义的端口号)，此时就能看到 Django 的 Demo 程序"It 's works!"了

129. Flask(__name__)中的__name__可以传入哪些值？

可以传入的参数：

1，字符串：‘hello’，
但是‘abc’,不行，因为 abc 是 python 内置的模块
2，__name__，约定俗成

不可以插入的参数

1，python 内置的模块，re,urllib,abc 等
2，数字

130. Flask 中请求钩子的理解和应用

请求钩子是通过装饰器的形式实现的，支持以下四种：
1，before_first_request 在处理第一个请求前运行
2，before_request:在每次请求前运行
3，after_request:如果没有未处理的异常抛出，在每次请求后运行
4，teardown_request:即使有未处理的异常抛出，在每次请求后运行

应用：

```
# 请求钩子
@api.after_request
def after_request(response):
    """设置默认的响应报文格式为 application/json"""
    # 如果响应报文 response 的 Content-Type 是以 text 开头，则将其改为默认的 json 类型
    if response.headers.get("Content-Type").startswith("text"):
        response.headers["Content-Type"] = "application/json"
return response
```

131. 自定义过滤器的步骤？

第一步：先定义自定义过滤器函数

```
def count_substring(string, substring):
    return string.count(substring)
```

第二步：注册自己定义的过滤器

```
app.jinja_env.filters['count_substring'] = count_substring
```

第三步：最后在模板文件 html 中直接使用注册时的键名

```
{#前面的作为原字符串 string,传入的作为子字符串 substring#}
{{ 'A long long long long long long longabc string ' | count_substring('long') }}<br/>
```

132. 如何把整个数据库导出来，再导入指定数据库中

导出：

mysqldump [-h 主机] -u 用户名 -p 数据库名 > 导出的数据库名.sql

导入指定的数据库中:

第一种方法：

mysqldump [-h 主机] -u 用户名 -p 数据库名 < 导出的数据库名.sql

第二种方法：

先创建好数据库,因为导出的文件里没有创建数据库的语句,如果数据库已经建好,则不用再创建。

create database example charset=utf8;（数据库名可以不一样）

切换数据库：

use example;

导入指定 sql 文件：

mysql>source /path/example.sql;

第五章、Python 爬虫

133. Scrapy 怎么设置深度爬取?

通过在 settings.py 中设置 DEPTH_LIMIT 的值可以限制爬取深度，这个深度是与 start_urls 中定义 url 的相对值。也就是相对 url 的深度。若定义 url 为 http://www.domz.com/game/,DEPTH_LIMIT=1 那么限制爬取的只能是此 url 下一级的网页。深度大于设置值的将被忽视。

134. 代理 IP 里的“透明”“匿名”“高匿”分别是指?

透明代理的意思是客户端根本不需要知道有代理服务器的存在，但是它传送的仍然是真实的 IP。你要想隐藏的话，不要用这个。

普通匿名代理能隐藏客户机的真实 IP，但会改变我们的请求信息，服务器端有可能会认为我们使用了代理。不过使用此种代理时，虽然被访问的网站不能知道你的 ip 地址，但仍然可以知道你在使用代理，当然某些能够侦测 ip 的网页仍然可以查到你的 ip。

高匿名代理不改变客户机的请求，这样在服务器看来就像有个真正的客户浏览器在访问它，这时客户的真实 IP 是隐藏的，服务器端不会认为我们使用了代理。

135. 字符集和字符编码

字符是各种文字和符号的总称，包括各个国家文字、标点符号、图形符号、数字等。字符集是多个字符的集合，字符集种类较多，每个字符集包含的字符个数不同，常见字符集有：ASCII 字符集、ISO 8859 字符集、GB2312 字符集、BIG5 字符集、GB18030 字符集、Unicode 字符集等。

字符编码就是以二进制的数字来对应字符集的字符。

常见的编码字符集（简称字符集）

Unicode：也叫统一字符集，它包含了几乎世界上所有的已经发现且需要使用的字符（如中文、日文、英文、德文等）。

ASCII：ASCII 既是编码字符集，又是字符编码。早期的计算机系统只能处理英文，所以 ASCII 也就成为了计算机的缺省字符集，包含了英文所需要的所有字符。

GB2312：中文字符集，包含 ASCII 字符集。ASCII 部分用单字节表示，剩余部分用双字节表示。

GBK：GB2312 的扩展，但完整包含了 GB2312 的所有内容。

GB18030：GBK 字符集的超集，常叫大汉字字符集，也叫 CJK（Chinese，Japanese，Korea）字符集，包含了中、日、韩三国语。

注意：Unicode 字符集有多种编码方式，如 UTF-8、UTF-16 等；ASCII 只有一种；大多数 MBCS（包括 GB2312）也只有一种。

136. 写一个邮箱地址的正则表达式？

[A-Za-z0-9\u4e00-\u9fa5]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$

137. 编写过哪些爬虫中间件？

user-agent、代理池等

138. 用什么方法提取数据？

答：可以使用正则、xpath、json、pyquery、bs4 等(任选一二即可)。

139. 平常怎么使用代理的 ？

1. 自己维护代理池
2. 付费购买（目前市场上有很多 ip 代理商，可自行百度了解，建议看看他们的接口文档（API&SDK））

140. 怎么获取加密的数据？

1. Web 端加密可尝试移动端（app）
2. 解析加密，看能否破解
3. 反爬手段层出不穷，js 加密较多，只能具体问题具体分析

141. 什么是分布式存储？

传统定义：分布式存储系统是大量 PC 服务器通过 Internet 互联，对外提供一个整体的服务。

分布式存储系统具有以下的几个特性：

可扩展：分布式存储系统可以扩展到几百台甚至几千台这样的一个集群规模，系统的整体性能线性增长。

低成本：分布式存储系统的自动容错、自动负载均衡的特性，允许分布式存储系统可以构建在低成本的服务器上。另外，线性的扩展能力也使得增加、减少服务器的成本低，实现分布式存储系统的自动运维。

高性能：无论是针对单台服务器，还是针对整个分布式的存储集群，都要求分布式存储系统具备高性能。

易用：分布式存储系统需要对外提供方便易用的接口，另外，也需要具备完善的监控、运维工具，并且可以方便的与其他的系统进行集成。

分布式存储系统的挑战主要在于数据和状态信息的持久化，要求在自动迁移、自动容错和并发读写的过程中，保证数据的一致性。分布式存储所涉及到的技术主要来自于两个领域：分布式系统以及数据库，如下所示：

数据分布：如何将数据均匀的分布到整个分布式存储集群中的各台服务器？如何从分布式存储集群中读取数据？

一致性：如何将数据的多个副本复制到多台服务器，即使在异常情况下，也能保证不同副本之间的数据一致性。

容错：如何可以快速检测到服务器故障，并自动的将在故障服务器上的数据进行迁移

负载均衡：新增的服务器如何在集群中保障负载均衡？数据迁移过程中如何保障不影响现有的服务。

事务与并发控制：如何实现分布式事务。

易用性：如何设计对外接口，使得设计的系统易于使用

压缩/加压缩：如何根据数据的特点设计合理的压缩/解压缩算法？如何平衡压缩/解压缩算法带来的空间和 CPU 计算资源？

142. 你所知道的分布式爬虫方案有哪些？

三种分布式爬虫策略：

1.Slaver 端从 Master 端拿任务（Request/url/ID）进行数据抓取，在抓取数据的同时也生成新任务，并将任务抛给 Master。Master 端只有一个 Redis 数据库，负责对 Slaver 提交的任务进行去重、加入待爬队列。

优点： scrapy-redis 默认使用的就是这种策略，我们实现起来很简单，因为任务调度等工作 scrapy-redis 都已经帮我们做好了，我们只需要继承 RedisSpider、指定 redis_key 就行了。

缺点： scrapy-redis 调度的任务是 Request 对象，里面信息量比较大（不仅包含 url，还有 callback 函数、headers 等信息），导致的结果就是会降低爬虫速度、而且会占用 Redis 大量的存储空间。当然我们可以重写方法实现调度 url 或者用户 ID。

2.Master 端跑一个程序去生成任务（Request/url/ID）。Master 端负责的是生产任务，并把任务去重、加入到待爬队列。Slaver 只管从 Master 端拿任务去爬。

优点： 将生成任务和抓取数据分开，分工明确，减少了 Master 和 Slaver 之间的数据交流；Master 端生成任务还有一个好处就是：可以很方便地重写判重策略（当数据量大时优化判重的性能和速度还是很重要的）。

缺点： 像 QQ 或者新浪微博这种网站，发送一个请求，返回的内容里面可能包含几十个待爬的用户 ID，即几十个新爬虫任务。但有些网站一个请求只能得到一两个新任务，并且返回的内容里也包含爬虫要抓取的目标信息，如果将生成任务和抓取任务分开反而会降低爬虫抓取效率。毕竟带宽也是爬虫的一个瓶颈问题，我们要秉着发送尽量少的请求为原则，同时也是为了减轻网站服务器的压力，要做一只有道德的 Crawler。所以，视情况而定。

3.Master 中只有一个集合，它只有查询的作用。Slaver 在遇到新任务时询问 Master 此任务是否已爬，如果未爬则加入 Slaver 自己的待爬队列中，Master 把此任务记为已爬。它和策略一比较像，但明显比策略一简单。策略一的简单是因为有 scrapy-redis 实现了 scheduler 中间件，它并不适用于非 scrapy 框架的爬虫。

优点： 实现简单，非 scrapy 框架的爬虫也适用。Master 端压力比较小，Master 与 Slaver 的数据交流也不大。

缺点： “健壮性”不够，需要另外定时保存待爬队列以实现“断点续爬”功能。各 Slaver 的待爬任务不通用。

如果把 Slaver 比作工人，把 Master 比作工头。策略一就是工人遇到新任务都上报给工头，需要干活的时候就去工头那里领任务；策略二就是工头去找新任务，工人只管从工头那里领任务干活；策略三就是工人遇到新任务时询问工头此任务是否有人做了，没有的话工人就将此任务加到自己的“行程表”。

143. 除了 scrapy-redis，有做过其他的分布式爬虫吗？

Celery、gearman 等，参考其他分布式爬虫策略

144. IP 存放在哪里？怎么维护 IP？对于封了多个 ip 的，怎么判定 IP 没被封？

存放在数据库(redis、mysql 等)
维护多个代理网站
一般代理的存活时间往往在十几分钟左右，定时任务，加上代理 IP 去访问网页，验证其是否可用，如果返回状态为 200，表示这个代理是可以使用的。

145. 假如每天爬取量在 5、6 万条数据，一般开几个线程，每个线程 ip 需要加锁限定吗？

1.5、6 万条数据相对来说数据量比较小，线程数量不做强制要求(做除法得一个合理值即可）
2.多线程使用代理，应保证不在同时一刻使用一个代理 IP

146. 怎么样让 scrapy 框架发送一个 post 请求（具体写出来）

使用 FormRequest

```
class mySpider(scrapy.Spider):
    # start_urls = ["http://www.taobao.com/"]

    def start_requests(self):
        url = 'http://http://www.taobao.com//login'

        # FormRequest  是 Scrapy 发送 POST 请求的方法
        yield scrapy.FormRequest(
            url = url,
            formdata = {"email" : "xxx", "password" : "xxxxx"},
            callback = self.parse_page
        )
    def parse_page(self, response):
        # do something
```

147. 怎么监控爬虫的状态

1. 使用 python 的 STMP 包将爬虫的状态信心发送到指定的邮箱
2. Scrapyd、pyspider

148. 怎么判断网站是否更新？

使用 MD5 数字签名

每次下载网页时，把服务器返回的数据流 ResponseStream 先放在内存缓冲区，然后对 ResponseStream 生成 MD5 数字签名 S1，下次下载同样生成签名 S2,比较 S2 和 S1，如果相同，则页面没有跟新，否则网页就有跟新。

149. 你爬出来的数据量大概有多大？大概多长时间爬一次？

无标准答案，根据自己爬取网站回答即可

150. 用什么数据库存爬下来的数据？部署是你做的吗？怎么部署？

1. 常用 MongoDB、mysql、redis 等
2. 是
3. 例：本地 pull，服务器 push 然后启动

关于部署建议了解以下技术栈：corntab、supervisor...

151. 分布式爬虫主要解决什么问题？

分布式只是提高爬虫功能和效率的一个环节而已，它从来不是爬虫的本质东西。

爬虫的本质是网络请求和数据处理，如何稳定地访问网页拿到数据，如何精准地提取出高质量的数据才是核心问题。

分布式爬虫只有当爬虫任务量很大的时候才会凸显优势，一般情况下也确实不必动用这个大杀器，所以要明确你的目标是什么

1)ip
2)带宽
3）cpu
4）io
5）大量任务

152. 谈一谈你对 Selenium 和 PhantomJS 了解？

Selenium 是一个 Web 的自动化测试工具，可以根据我们的指令，让浏览器自动加载页面，获取需要的数据，甚至页面截屏，或者判断网站上某些动作是否发生。Selenium 自己不带浏览器，不支持浏览器的功能，它需要与第三方浏览器结合在一起才能使用。但是我们有时候需要让它内嵌在代码中运行，所以我们可以用一个叫 PhantomJS 的工具代替真实的浏览器。Selenium 库里有个叫 WebDriver 的 API。WebDriver 有点儿像可以加载网站的浏

览器，但是它也可以像 BeautifulSoup 或者其他 Selector 对象一样用来查找页面元素，与页面上的元素进行交互 (发送文本、点击等)，以及执行其他动作来运行网络爬虫。

PhantomJS 是一个基于 Webkit 的“无界面”(headless)浏览器，它会把网站加载到内存并执行页面上的 JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器要高效。

如果我们把 Selenium 和 PhantomJS 结合在一起，就可以运行一个非常强大的网络爬虫了，这个爬虫可以处理 JavaScrip、Cookie、headers，以及任何我们真实用户需要做的事情。

153. Python 中有哪些模块可以发送请求？

socket，request，urlib...

154. “极验”滑动验证码如何破解？

1.selenium 控制鼠标实现，速度太机械化，成功率比较低

2.计算缺口的偏移量（推荐博客：http://blog.csdn.net/paololiu/article/details/52514504?%3E）

3.“极验”滑动验证码需要具体网站具体分析，一般牵扯算法乃至深度学习相关知识

155. 增量爬取？

增量爬取即保存上一次状态，本次抓取时与上次比对，如果不在上次的状态中，便视为增量，保存下来。对于 scrapy 来说，上一次的状态是抓取的特征数据和上次爬取的 request 队列（url 列表），request 队列可以通过 request 队列可以通过 scrapy.core.scheduler 的 pending_requests 成员得到，在爬虫启动时导入上次爬取的特征数据，并且用上次 request 队列的数据作为 start url 进行爬取，不在上一次状态中的数据便保存。

选用 BloomFilter 原因：对爬虫爬取数据的保存有多种形式，可以是数据库，可以是磁盘文件等，不管是数据库，还是磁盘文件，进行扫描和存储都有很大的时间和空间上的开销，为了从时间和空间上提升性能，故选用 BloomFilter 作为上一次爬取数据的保存。保存的特征数据可以是数据的某几项，即监控这几项数据，一旦这几项数据有变化，便视为增量持久化下来，根据增量的规则可以对保存的状态数据进行约束。比如：可以选网页更新的时间，索引次数或是网页的实际内容，cookie 的更新等...

第六章、数据分析

156. 1G 大小的文件，每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M，返回频数最高的 100 个词。

使用生成器读取文件。每次读取 65536 行，一共进行 1500 次，当读取不到内容时关闭文件。每次读取，最终要得到 100 个频数最高的词。每 500 次，进行一次合并和统计，得到最多 50000 个词，对这 50000 个词提取其中频数最高的 100 个词。最终对最多 300 个筛选出来的词，进行合并和统计，提取最终频数最高的 100 个词。

筛选出 100 个高频词的步骤：

1.统计每个词出现的次数。维护一个 Key 为词，Value 为该词出现次数的 hash 表。每次读取一个词，如果该字串不在 hash 表中，那么加入该词，并且将 Value 值设为 1；如果该字串在 hash 表中，那么将该字串的计数加一即可。

2.根据每个词的引用次数进行排序。冒泡、快排等等都可以

157. 一个大约有一万行的文本文件，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想和时间复杂度分析。

用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)(le 表示单词的平准长度)。然后是找出出现最频繁的前 10 个词，可以用堆来实现，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一个。

http://www.mamicode.com/info-detail-1037262.html

158. 怎么在海量数据中找出重复次数最多的一个？

算法思想：

先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。

然后找出上一步求出的数据中重复次数最多的一个就是所求

https://www.cnblogs.com/lianghe01/p/4391804.html

http://blog.csdn.net/u010601183/article/details/56481868

159. 在 1 亿个整数中找出不重复的整数

思路和上题一样。不同之处在于，统计完所有整数的引用之后，找出所有引用为 1 的整数。

160. 给 2 亿个不重复的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 2 亿个数当中？

unsigned int 的取值范围是 0 到 2^32-1。我们可以申请连续的 2^32/8=512M 的内存，用每一个 bit 对应一个 unsigned int 数字。首先将 512M 内存都初始化为 0，然后每处理一个数字就将其对应的 bit 设置为 1。当需要查询时，直接找到对应 bit，看其值是 0 还是 1 即可

161. 逻辑斯蒂回归和线性回归的区别

线性回归使用的是线性方程。通过样本 X 获得预测值 Y，然后通过最小化所有样本预测值 Y 和真实值 Y`的误差，获得模型参数。

逻辑斯蒂回归使用的是 $Y=\frac{1}{1+e^{-W^TX}}$。逻辑斯蒂回归用于二分类问题，所以

它不是通过样本值得到预测值，而是计算两个结果各自的概率
区别：1.使用的数学模型不同；2.应用场景不同

162. 聚类分析？聚类算法有哪几种？请选择一种详细描述其计算原理和步骤。

聚类分析：将大量的对象，根据某些特征进行分类的行为
常见聚类算法：k-means，层次聚类算法，SOM 聚类算法，FCM 聚类算法
k-means 聚类算法：算法的核心-“距离”
步骤：1.随机设置 k 个特征空间内的点，作为聚类中心
2.对其他每个点，分别计算到各个聚类中心的距离。根据距离，选择最近的聚类中心作为这个未知点的标记类别。
3.每次将一个未知点标记类别后，重新计算所属类别的聚类中心。
4.重复以上过程，直到准则函数收敛。准则函数计算的是 空间内所有点到各聚类中心的距离之和。

163. 如何利用 numpy 对数列的前 n 项进行排序？

数列本身就是有序的。
numpy 对数组的前 n 项进行排序：
1.ndarray.sort(axis=, kind=, order=)
参数说明：axis：沿着什么方向进行排序（0：按列，1：按行）
kind：使用什么排序方式（可选的有 quicksort,mergesort,heapsort）
2.numpy.sort(a, axis=, kind=, order=)
第一个参数是要排序的数组，其他参数和函数 1 相同
3.numpy.argsort(a, axis=, kind=, order)
参数和函数 2 完全相同。
区别在于：函数 2 直接修改原始数组

164. 描述 numpy array 比 python list 的优势？

numpy.array 支持更多的索引方式
numpy.array 类型支持[n,m]形式的检索方式。
例如 a 为 array([1,2,3],
[4,5,6],
[7,8,9])。a[:, 1]输出为[2,5,8]。

165. 是否了解做数据分析的一些框架？例如 zipline？

Zipline 是一个交易算法库，该系统是对现场交易系统如何运转的一个近似，可以对历史数据进行投资算法的回溯检验。Zipline 目前作为 Quantopian 的回溯检验引擎。

166. 有用过 Elasticsearch 吗？

ElasticSearch 是一个基于 Lucene 的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口，中文文档：

https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/

167. 有使用过 hive 吗？

hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的 sql 查询功能，可以将 sql 语句转换为 MapReduce 任务进行运行。

官方文档：https://hive.apache.org/

中文文档：

http://blog.csdn.net/u011812294/article/details/53610210?utm_source=itdadao&utm_medium=referral

168. 现有如下数据；

A=[{“dt”:”2016-06-06”,”a”:100},{“dt”:”2016-06-07”,”b”:200}]

B=[{“dt”:”2016-06-06”,”b”:200},{“dt”:”2016-06-07”,”a”:400}]

请使用 pandas 进行数据处理并得到如下结果：

C=[{“dt”:”2016-06-06”,”a”:100,”b”:200},

{“dt”:”2016-06-07”,”a”:400,”b”:200}

```
import pandas as pd

A = [{'dt': '2016-06-06', 'a': 100}, {'dt': '2016-06-07', 'b': 200}]
B = [{"dt": '2016-06-06', 'b': 200}, {'dt': '2016-06-07', 'a': 400}]

dfa = pd.DataFrame(A)
dfb = pd.DataFrame(B)
dfa.update(dfb)
dfa.to_dict(orient='index')
res = list()
for i in dfa.to_dict(orient='index').values():
    res.append(i)
print res
```

第七章、数据结构与算法

169. 用两个队列如何实现一个栈，用两个栈如何实现一个队列？

两个栈实现一个队列

栈的特性是先进后出（FILO）,队列的特性是先进先出（FIFO）,在实现 delete 时，将一个栈中的数据依次拿出来压入到另一个为空的栈，另一个栈中数据的顺序恰好是先压入栈 1 的元素此时在栈 2 的上面，为了实现效率的提升，在 delete 时，判断栈 2 是否有数据，如果有的话，直接删除栈顶元素，在栈 2 为空时才将栈 1 的数据压入到栈 2 中，从而提高程序的运行效率，实现过程可以分为下面几个步骤：

1、push 操作时，一直将数据压入到栈 2 中

2、delete 操作时，首先判断栈 2 是否为空，不为空的情况直接删除栈 2 栈顶元素，为空的话将栈 1 的数据压入到栈 2 中，再将栈 2 栈顶元素删除。

两个队列实现一个栈

因为队列是先进先出，所以要拿到队列中最后压入的数据，只能每次将队列中数据 pop 到只剩一个，此时这个数据为最后压入队列的数据，在每次 pop 时，将数据压入到另一个队列中。每次执行 delete 操作时，循环往复。（感觉效率低）每次删除时间复杂度 O(N)

170. 基础的数据结构有哪些？

数据结构是以某种形式将数据组织在一起的集合,不仅存储数据，还支持访问和处理数据的操作.

基础的数据结构有:线性表(数组,链表),栈与队列,树与二叉树,图等

171. 基本的算法有哪些，怎么评价一个算法的好坏？

基本的算法有：排序算法(简单排序，快速排序，归并排序)，搜索(二分搜索)，对于其他基本数据结构，栈，队列,树,都有一些基本的操作.

评价算法的好坏一般有两种：时间复杂度和空间复杂度

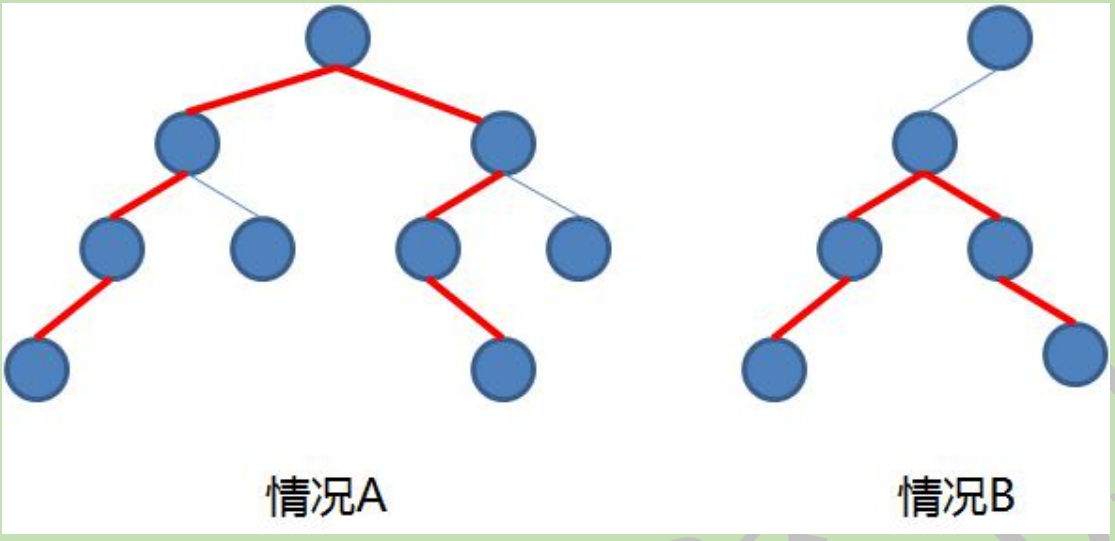
172. 找出二叉树中最远结点的距离？

计算一个二叉树的最大距离有两个情况:

情况 A: 路径经过左子树的最深节点，通过根节点，再到右子树的最深节点。

情况 B: 路径不穿过根节点，而是左子树或右子树的最大距离路径，取其大者。

只需要计算这两个情况的路径距离，并取其大者，就是该二叉树的最大距离。



173. 那种数据结构可以实现递归（栈）？

栈，递归需要保存正在计算的上下文，等待当前计算完成后弹出,再继续计算，只有栈先进后出的的特性才能实现

174. 你知道哪些排序算法（一般是通过问题考算法）

冒泡，选择，快排，归并

175. 写一个二叉树

```
class TreeNode(object):
    def __init__(self, left=None, right=None, data=None):
        self.data = data
        self.left = left
        self.right = right

def preorder(root):       #前序遍历
    if root is None:
        return
    else:
        print root.data
        preorder(root.left)
        preorder(root.right)

def inorder(root):      #中序遍历
    if root is None:
        return
```

```
    else:
        inorder(root.left)
        print root.data
        inorder(root.right)

def postorder(root):   # 后序遍历
    if root is None:
        return
    else :
        postorder(root.left)
        postorder(root.right)
        print root.data
```

176. 写一个霍夫曼数

```
class TreeNode(object):
    def __init__(self, left=None, right=None, data=None):
        self.data = data
        self.left = left
        self.right = right
# 根据数组 a 中 n 个权值建立一棵哈夫曼树，返回树根指针
def CreateHuffman(a):
    a.sort()  # 对列表进行排序(从小到大)
    n = len(a)
    b = [TreeNode(data=a[i]) for i in range(n)]
    for i in range(n):  # 进行 n-1 次循环建立哈夫曼树
        [print(temp, end=' ') for temp in b ]
        #k1 表示森林中具有最小权值的树根结点的下标，k2 为次最小的下标
        k1 = -1
        k2 = 0
        for j in range(n):  # 让 k1 初始指向森林中第一棵树，k2 指向第二
棵
            if b[j]:
                if k1 == -1:
                    k1 = j
                    continue
                k2 = j
                break
        print('k1', k1, 'k2', k2)
        # print(b[k1].data, b[k2].data)
        for j in range(k2, n):  # 从当前森林中求出最小权值树和次最小
            if b[j]:
                if b[j].data < b[k1].data:
                    k2 = k1
```

```
                k1 = j
            elif b[k2]:
                if b[j].data < b[k2].data:
                    k2 = j

        # 由最小权值树和次最小权值树建立一棵新树，q 指向树根结点
        if b[k2]:
            q = TreeNode()
            q.data = b[k1].data + b[k2].data
            q.left = b[k1]
            q.right = b[k2]

            b[k1] = q  # 将指向新树的指针赋给 b 指针数组中 k1 位置
            b[k2] = None  # k2 位置为空
    return q  # 返回整个哈夫曼树的树根指针
```

177. 写一个二分查找

```
def binary_search(a, n, key):
    m = 0; l = 0; r = n - 1#闭区间[0, n - 1]
    while (l < r):
        m = l + ((r - l) >> 1)#向下取整
        if (a[m] < key): l = m + 1;
        else: r = m;

    if (a[r] == key): return r;
    return -1
```

178. set 用 in 时间复杂度是多少，为什么？

O(1)，因为 set 是键值相同的一个数据结构，键做了 hash 处理

179. 深度优先遍历和广度优先遍历的区别？

1） 二叉树的深度优先遍历的非递归的通用做法是采用栈，广度优先遍历的非递归的通用做法是采用队列。

2） 深度优先遍历：对每一个可能的分支路径深入到不能再深入为止，而且每个结点只能访问一次。要特别注意的是，二叉树的深度优先遍历比较特殊，可以细分为先序遍历、中序遍历、后序遍历。具体说明如下：

- 先序遍历：对任一子树，先访问根，然后遍历其左子树，最后遍历其右子树。
- 中序遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树。
- 后序遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访

问根。

广度优先遍历：又叫层次遍历，从上往下对每一层依次访问，在每一层中，从左往右（也可以从右往左）访问结点，访问完一层就进入下一层，直到没有结点可以访问为止。

3）深度优先搜素算法：不全部保留结点，占用空间少；有回溯操作(即有入栈、出栈操作)，运行速度慢。

广度优先搜索算法：保留全部结点，占用空间大； 无回溯操作(即无入栈、出栈操作)，运行速度快。

通常 深度优先搜索法不全部保留结点，扩展完的结点从数据库中弹出删去，这样，一般在数据库中存储的结点数就是深度值，因此它占用空间较少。所以，当搜索树的结点较多，用其它方法易产生内存溢出时，深度优先搜索不失为一种有效的求解方法。

广度优先搜索算法，一般需存储产生的所有结点，占用的存储空间要比深度优先搜索大得多，因此，程序设计中，必须考虑溢出和节省内存空间的问题。但广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索要快些

180. 列表中有 n 个正整数范围在[0,1000]，请编程对列表中的数据进行排序；

```
def quick_sort(lista,start,stop):
    if start >= stop:
        return
    low = start
    high = stop
    mid = lista[start]
    while low < high:
        while low < high and lista[high] >= mid:
                high -= 1
        lista[low] = lista[high]
        while low < high and mid > lista[low]:
                low += 1
        lista[high] = lista[low]
    lista[low] = mid
    quick_sort(lista,start,low-1)
    quick_sort(lista,low+1,stop)
    return lista

lista = [0,1000]
print(quick_sort(lista,0,len(lista)-1))
```

181. 面向对象编程中有组合和继承的方法实现新的类，假设我们手头只有“栈”类，请用“组合”的方式使用“栈”（LIFO）来实

现“队列”（FIFO），完成以下代码。

```
Import stack
Class queue(stack):
    def __init__(self):
    def __init__(self):
     stack1 = stack() # 进来的元素都放在里面
     stack2 = stack() # 元素都从这里出去

def push(self, element):
     self.stack1.push(element)

def pop(self):
    if self.stack2.empty():#如果没有元素，就把负责放入元素的栈中元素
全部放进来
        while not self.stack1.empty():
            self.stack2.push(self.stack1.top())
            self.stack1.pop()

    ret = self.stack2.top() # 有元素后就可以弹出了
    self.stack2.pop()
    return ret

def top(self):
    if (self.stack2.empty()):
        while not self.stack1.empty():
            self.stack2.push(self.stack1.top())
            self.stack1.pop()
    return self.stack2.top()
```

182. 求和问题。给定一个数组，数组中的元素唯一，数组元素数量 N >2，若数组中的两个数相加和为 m，则认为该数对满足要求，请思考如何返回所有满足要求的数对（要求去重），并给出该算法的计算复杂度和空间复杂度。

```
def func(arr, d):
    count = list()
    for index, i in enumerate(arr):
        for j in arr[index+1:]:
            if i + j == d:
                count.append((i, j))
    return count
```

时间复杂度 O(n^2)　空间复杂度 O(n)

183. 写程序把一个单向链表顺序倒过来(尽可能写出更多的实现方法，标出所写方法的空间和时间复杂度)

```
class ListNode:
    def __init__(self,x):
        self.val=x;
        self.next=None;

def nonrecurse(head):                #循环的方法反转链表
    if head is None or head.next is None:
        return head;
    pre=None;
    cur=head;
    h=head;
    while cur
        h=cur;
        tmp=cur.next;
        cur.next=pre;
        pre=cur;
        cur=tmp;
    return h;

class ListNode:
    def __init__(self,x):
        self.val=x;
        self.next=None;


def recurse(head,newhead):     #递归，head 为原链表的头结点，newhead 为反转后链表的头结点
    if head is None:
        return ;
    if head.next is None:
        newhead=head;
    else :
        newhead=recurse(head.next,newhead);
        head.next.next=head;
        head.next=None;
    return newhead;
```

184. 有一个长度为 n 的数组 a，里面的元素都是整数，现有一个整数 B，写程序判断数组 a 中是否有两个元素的和等于 B（尽可能写出更多的实现方法，标出所写方法的空间和时间复杂度）

```
def func(arr, d):
    count = list()
    for index, i in enumerate(arr):
        for j in arr[index+1:]:
            if i + j == d:
                count.append((i, j))
    return count
```

185. 二叉树如何求两个叶节点的最近公共祖先？

二叉树是搜索二叉树

1、原理：二叉搜索树是排序过的 ，位于左子树的结点都比父结点小，位于右子树的结点都比父结点大，我们只需从根节点开始和两个输入的结点进行比较，如果当前节点的值比两个结点的值都大，那么最低的公共祖先结点一定在该结点的左子树中，下一步开遍历当前结点的左子树。如果当前节点的值比两个结点的值都小，那么最低的公共祖先结点一定在该结点的右子树中，下一步开遍历当前结点的右子树。这样从上到下找到第一个在两个输入结点的值之间的结点。

2、实现代码

```
    class TreeNode(object):
        def __init__(self, left=None, right=None, data=None):
            self.data = data
            self.left = left
            self.right = right

    def getCommonAncestor(root, node1, node2):
        while root:
            if root.data > node1.data and root.data > node2.data:
                root = root.left
            elif root.data < node1.data and root.data < node2.data:
                root = root.right
            else:
                return root
        return None
```

测试代码:

```
    root = TreeNode(data=10) #构造 二叉排序树,
    #特点是任意节点的左子树数值都比根节点数值小, 右子树的数值都比根节点数值大
    node1=TreeNode(data=3)
    node2=TreeNode(data=11)
    node3=TreeNode(data=1)
    node4=TreeNode(data=4)
    node5=TreeNode(data=5)
    node6=TreeNode(data=16)
```

```
    root.left = node1 #左子树数值都比根节点数值小，右子树的数值都比根节点数值大
    root.right = node2
    node1.left = node3
    node1.right = node4
    node2.left = node5
    node2.right = node6

    node = getCommonAncestor(root, node4, node5)
    print(node.data)
```

二叉树节点中包含指向父节点的指针

1、原理：如果树中每个结点都有父结点（根结点除外），这个问题可以转换成求两个链表的第一个公共结点，假设树结点中指向父结点的指针是 parent，树的每一个叶结点开始都由一个指针 parent 串起来的链表，每个链表的尾结点就是树的根结点。那么输入的这两个结点位于链表上，它们的最低公共祖先结点刚好是这两个链表的第一个公共结点。

2、实现代码

```
    class TreeNode(object):
        def __init__(self, data=None, parent=None, left=None, right=None):
            self.parent = parent
            self.data = data
            self.left = left
            self.right = right

    def getHeight(node):
        len = 0
        while node:
            len += 1
            node = node.parent
        return len

    def getCommonAncestor(root, node1, node2):
        len1 = getHeight(node1)
        len2 = getHeight(node2)
        #寻找两个连表的第一个交点
        while len1 != len2:
            if len1 < len2:
                len2 -= 1
            else:
                len1 -= 1
```

```
        while node1 != None and node2 != None and node1 != node2:
            node1 = node1.parent
            node2 = node2.parent

        if node1 == node2:
            return node1
        return None
```

二叉树是普通二叉树， 没有指向父结点的指针

1、原理：在二叉树根结点 的左子树和右子树中分别找输入的两个结点，如果两个结点都在左子树，遍历当前结点的左子树，如果两个结点都在右子树，遍历当前结点的右子树，直到一个在当前结点的左子树，一个在当前结点的右子树，返回当前结点就是最低的公共祖先结点。

2、实现代码

```
    def isNodeinTree(root, node):
        if root == node:
            return True
        if isNodeinTree(root.left, node) or isNodeinTree(root.right,
node):
            return True
        return False

    def getCommontAncestor(root, node1, node2):
        isnode1inleft = isNodeinTree(root.left, node1)
        isnode1inright = isNodeinTree(root.right, node1)
        isnode2inleft = isNodeinTree(root.left, node2)
        isnode2inright = isNodeinTree(root.right, node2)
        if (isnode1inleft and isnode2inright) or (isnode1inright and
isnode2inleft):
            return root
        if isnode1inleft and isnode2inleft:
            return getCommontAncestor(root.left, node1, node2)
        if isnode1inright and isnode2inright:
        return getCommontAncestor(root.right, node1, node2)
        return None
```

186. 两个字符串，如何球公共字符串？

```
    def getLCString(str1, str2):
     maxlen = len(str1) if len(str1) < len(str2) else len(str2)
     example = str1 if len(str1) < len(str2) else str2
     other = str2 if str1 == example else str1
     for i in range(maxlen):
```

```
        for j in range(maxlen, 0, -1):
            if other.find(example[i:j]) != -1:
                return example[i:j]
```

187. 桶排序（最快最简单的排序）

桶排序的基本思想是将一个数据表分割成许多 buckets，然后每个 bucket 各自排序，或用不同的排序算法，或者递归的使用 bucket sort 算法。也是典型的 divide-and-conquer 分而治之的策略。它是一个分布式的排序，介于 MSD 基数排序和 LSD 基数排序之间

```
def bucketSort(nums):
    # 选择一个最大的数
    max_num = max(nums)
    # 创建一个元素全是 0 的列表，当做桶
    bucket = [0]*(max_num+1)
    # 把所有元素放入桶中，即把对应元素个数加一
    for i in nums:
        bucket[i] += 1
    # 存储排序好的元素
    sort_nums = []
    # 取出桶中的元素
    for j in range(len(bucket)):
        if bucket[j] != 0:
            for y in range(bucket[j]):
                sort_nums.append(j)
    return sort_nums
nums = [5,6,3,2,1,65,2,0,8,0]
print bucketSort(nums)
```

1、桶排序是稳定的
2、桶排序是常见排序里最快的一种，大多数情况下比快排还要快
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

188. 斐波那契数列

斐波那契数列：简单地说，起始两项为 0 和 1，此后的项分别为它的前两项之和。

```
def fibo(num):
    numList = [0,1]
    for i in range(num - 2):
        numList.append(numList[-2] + numList[-1])
    return numList
```

189. 排序算法的分析

排序算法的稳定性：如果在对象序列中有两个对象 r[i]和 r[j] ,它们的排序码

k[i]==k[j] 。如果排序前后,对象 r[i]和 r[j] 的相对位置不变，则称排序算法是稳定的；否则排序算法是不稳定的。
时间开销：排序的时间开销可用算法执行中的数据比较次数与数据移动次数来衡量。
算法运行时间代价的大略估算一般都按平均情况进行估算。对于那些受对象排序码序列初始排列及对象个数影响较大的，需要按最好情况和最坏情况进行估算
空间开销：算法执行时所需的附加存储。

190. 冒泡

冒泡排序的思想: 每次比较两个相邻的元素, 如果他们的顺序错误就把他们交换位置。

```
def bubble_improve(l):
    print l
    flag = 1
    for index in range(len(l) - 1, 0 , -1):
        if flag:
            flag = 0
            for two_index in range(index):
                if l[two_index] > l[two_index + 1]:
                    l[two_index], l[two_index + 1] = l[two_index + 1],
l[two_index]
                    flag = 1
        else:
            break
    print l
l = [10, 20, 40, 50, 30, 60]
bubble_improve(l)
```

191. 快排

快速排序使用分治法策略来把一个序列分为两个子序列。
步骤：
从数列中挑出一个元素，称为 "基准"（pivot），
重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分割结束之后，该基准就处于数列的中间位置。这个称为分割（partition）操作。
递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

```
def quickSort(alist,first,last):
   if first<last:
       splitpoint = findpos(alist,first,last)
       quickSort(alist,first,splitpoint-1)
       quickSort(alist,splitpoint+1,last)
```

```
def findpos(lists, low, high):
    key = lists[low]
    while low < high:
       while low < high and lists[high] >= key:
           high -= 1
       lists[low] = lists[high]
       while low < high and lists[low] <= key:
           low += 1
       lists[high] = lists[low]
    lists[low] = key
    return low
alist = [54,26,93,17,77,31,44,55,20]
quickSort(alist,0,8)
print(alist)
```

第八章、基础编程题

192. 古典问题：生兔子

有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？

```
a = 1
b = 1
for i in range(1,21,2):
    print '%d %d'%(a,b),
    a += b
    b += a
```

193. 猴子吃桃问题

猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少。

```
#!/usr/bin/python

# -*- coding: UTF-8 -*-

x2 = 1for day in range(9,0,-1):

    x1 = (x2 + 1) * 2

    x2 = x1print x1
```

194. 题目：一个 5 位数，判断它是不是回文数。即 12321 是回文数，

个位与万位相同，十位与千位相同。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
a = int(raw_input("请输入一个数字:\n"))
x = str(a)
flag = True
for i in range(len(x)/2):
    if x[i] != x[-i - 1]:
        flag = False
        breakif flag:
    print "%d 是一个回文数!" % aelse:
    print "%d 不是一个回文数!" % a
```

195. 经典题目：报数

有 n 个人围成一圈，顺序排号。从第一个人开始报数（从 1 到 3 报数），凡报到 3 的人退出圈子，问最后留下的是原来第几号的那位。

```
if __name__ == '__main__':
    nmax = 50
    n = int(input('please input the total of numbers:'))
    num = []
    for i in range(n):
        num.append(i + 1)
    i = 0
    k = 0
    m = 0
    while m < n - 1:
        if num[i] != 0 : k += 1
        if k == 3:
            num[i] = 0
            k = 0
            m += 1
        i += 1
        if i == n : i = 0
    i = 0
    while num[i] == 0: i += 1
    print (num[i])
```

196. 有 1、2、3、4 个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？

```
#Filename:001.py
cnt = 0#count the sum of result
for i in range(1,5):
    for j in range(1,5):
        for k in range(1,5):
            if i!=j and i!=k and j!=k:
                print i*100+j*10+k
                cnt+=1
print cnt
```

197. 一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如 6=1＋2＋3.找出 1000 以内的所有完数。

```
from math import sqrt
n = int(raw_input('input a number:'))
sum = n*-1
k = int(sqrt(n))
for i in range(1,k+1):
    if n%i == 0:
        sum += n/i
        sum += i
if sum == n:
    print 'YES'
else:
    print 'NO'
```

198. 利用递归函数调用方式，将所输入的 5 个字符，以相反顺序打印出来。

```
def output(s,l):
    if l==0:
        return
    print s[l-1]
    output(s,l-1)

s = raw_input('Input a string:')
l = len(s)
output(s,l)
```