



黑马程序员™  
www.itheima.com

- 黑马程序员

# Python 面试宝典

Version 8.1



第一章 内容介绍.....	26
第二章 Python 基础.....	27
一、 基础语法.....	27
1. 输入与输出.....	27
2. 条件与循环.....	28
3. 文件操作.....	30
4. 异常.....	32
5. 模块与包.....	33
6. Python 特性.....	39
7. Linux 基础和 git.....	51
二、 数据类型.....	55
1. 字典.....	55
2. 字符串.....	56
3. 列表.....	57
4. 元组.....	64
5. 集合.....	64
第三章 Python 高级.....	65
一、 元类.....	65
二、 内存管理与垃圾回收机制.....	66
三、 函数.....	68
1. 函数参数.....	68
2. 内建函数.....	70

3.Lambda.....	73
四 . 设计模式.....	74
1. 单例.....	74
2. 工厂 .....	75
3. 装饰器.....	75
4. 生成器.....	76
五 . 面向对象.....	78
1. 类.....	78
2. 对象.....	78
六 . 正则表达式.....	80
七 . 系统编程.....	82
八 . 网络编程.....	92
第四章 前端.....	108
一 . Html.....	108
二 . Css.....	108
1. 什么是 CSS 初始化 ? 有什么好处 ? (2018-4-16-lxy).....	108
2. 简述浮动的特征和清除浮动的方法 ? (2018-4-16-lxy).....	108
三 . JavaScript.....	109
四 . jQurey.....	110
五 . vue.js.....	110
第五章 Web.....	110
一 . Flask.....	110

二 . Django.....	121
三 . Tornado.....	157
第六章 爬虫.....	158
一 . 常用库与模块.....	158
1. 试列出至少三种目前流行的大型数据库的名称:____、____、____,其中您最熟悉的是____,从____年开始使用 ( 2018-4-1-ydy ) .....	158
2. 列举您使用过的 Python 网络爬虫所用到的网络数据包?(2018-4-16-lxy).....	158
3. 列举您使用过的 Python 网络爬虫所用到的解析数据包 ( 2018-4-1-ydy ) .....	158
4. 爬取数据后使用哪个数据库存储数据的,为什么? ( 2018-4-1-ydy ) .....	158
5. 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?(2018-4-16-lxy).....	161
6. 写爬虫是用多进程好?还是多线程好?为什么?(2018-4-16-lxy).....	162
7. 常见的反爬虫和应对方法?(2018-4-16-lxy).....	162
8. 解析网页的解析器使用最多的是哪几个?(2018-4-16-lxy).....	164
9. 需要登录的网页,如何解决同时限制 ip, cookie,session ( 其中有一些是动态生成的 ) 在不使用动态爬取的情况下?(2018-4-16-lxy).....	164
10. 验证码的解决?(2018-4-16-lxy).....	164
11. 使用最多的数据库 ( Mysql , Mongodb , redis 等 ) , 对他们的理解?(2018-4-16-lxy).....	164
12. 字符集和字符编码(2018-4-23-lyf).....	165
13. 写一个邮箱地址的正则表达式?(2018-4-23-lyf).....	166
14. 编写过哪些爬虫中间件?(2018-4-23-lyf).....	166
15. “极验” 滑动验证码如何破解?(2018-4-23-lyf).....	166

16. 爬的那些内容数据量有多大，多久爬一次，爬下来的数据是怎么存储？( 2018-4-20-xhq )	167
17. cookie 过期的处理问题？(2018-4-20-xhq)	167
18. 动态加载又对及时性要求很高怎么处理？(2018-4-20-xhq)	167
19. HTTPS 有什么优点和缺点(2018-4-20-xhq)	167
20. HTTPS 是如何实现安全传输数据的。(2018-4-20-xhq)	168
21. TTL，MSL，RTT？(2018-4-20-xhq)	168
22. 谈一谈你对 Selenium 和 PhantomJS 了解 ( 2018-4-23-xhq )	169
23. 代理 IP 里的“透明”“匿名”“高匿”分别是指？(2018-4-23-xhq)	170
24. requests 返回的 content 和 text 的区别？	170
25. robots 协议(2018-4-23-xhq)	171
26. 为什么 requests 请求需要带上 header？(2018-4-23-xhq)	171
27. dumps,loads 与 dump,load 的区别？(2018-4-23-xhq)	172
28. 通用爬虫：通常指搜索引擎的爬虫 ( 2018-4-23-xhq )	172
29. requests 使用小技巧(2018-4-23-xhq)	173
30. 平常怎么使用代理的？(2018-4-23-xhq)	173
31. 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？(2018-4-23-xhq)	173
32. 怎么获取加密的数据？( 2018-4-23-xhq )	174
33. 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？ ( 2018-4-23-xhq )	174
34. 怎么监控爬虫的状态 ( 2018-4-23-xhq )	174
二. Scrapy	175

1. 描述下 scrapy 框架运行的机制？(2018-4-16-lxy).....	175
2. 谈谈你对 Scrapy 的理解？(2018-4-23-lyf).....	175
3. 怎么样让 scrapy 框架发送一个 post 请求（具体写出来）(2018-4-23-lyf).....	176
4. 怎么判断网站是否更新？(2018-4-23-lyf).....	176
5. 图片、视频爬取怎么绕过防盗连接，或者说怎么获取正确的链接地址？(2018-4-23-lyf).....	176
6. 你爬出来的数据量大概有多大？大概多长时间爬一次？(2018-4-23-lyf).....	177
7. 增量爬取(2018-4-23-lyf).....	177
8. 爬虫向数据库存数据开始和结束都会发一条消息，是 scrapy 哪个模块实现的？ （2018-4-20-xhq）.....	177
9. 爬取下来的数据如何去重，说一下具体的算法依据（2018-4-20-xhq）.....	178
10. Scrapy 的优缺点？（2018-4-23-xhq）.....	179
11. 怎么设置深度爬取？(2018-4-23-xhq).....	179
三 . Scrapy-redis.....	180
1. scrapy 和 scrapy-redis 有什么区别？为什么选择 redis 数据库？(2018-4-16-lxy).....	180
2. 分布式爬虫主要解决什么问题？(2018-4-16-lxy).....	180
3. 什么是分布式存储？(2018-4-23-lyf).....	180
4. 你所知道的分布式爬虫方案有哪些？(2018-4-23-lyf).....	181
5. 除了 scrapy-redis，有做过其他的分布式爬虫吗？(2018-4-23-lyf).....	183
6. 在爬取的时候遇到某些内容字段缺失怎么判断及处理？(2018-4-23-lyf).....	183
四 . 自定义框架.....	183
第七章 shell 与自动化运维.....	183
第八章 测试.....	183

一、 测试面试题.....	183
1. 禅道和 qc 的区别 ? (2018-4-23-zcz).....	183
2. 编写测试计划的目的是(2018-4-23-zcz).....	184
3. 测试人员在软件开发过程中的任务是什么(2018-4-23-zcz).....	184
4. 您以往的工作中，一条软件缺陷（ 或者叫 Bug ）记录都包含了哪些内容？如何提交高质量的软件缺陷（ Bug ）记录？(2018-4-23-zcz).....	185
5. 简述黑盒测试和白盒测试的优缺点(2018-4-23-zcz).....	185
6. 简述常用的 Bug 管理或者用例管理工具,并且描述其中一个工作流程。(2018-4-23-zcz)	186
7. 请列出你所知道的软件测试种类，至少 5 项。(2018-4-23-zcz).....	186
8. Alpha 测试与 Beta 测试的区别是什么？(2018-4-23-zcz).....	186
9. 举例说明什么是 Bug？一个 bug report 应包含什么关键字？(2018-4-23-zcz).....	186
第九章 数据库.....	187
一 . Mysql.....	187
1. Python 中操作 Mysql 步骤 ( 2018-3-31-sxd).....	187
2. SQL 的 select 语句完整的执行顺序 ( 2018-3-31-sxd ) .....	188
3. 说一下 Mysql 数据库存储的原理？(2018-4-16-lxy).....	191
4. 事务的特性？(2018-4-16-lxy).....	191
5. 数据库索引？(2018-4-16-lxy).....	192
6. 数据库怎么优化查询效率？(2018-4-16-lxy).....	192
7. Mysql 集群的优缺点？(2018-4-16-lxy).....	193
8. 你用的 Mysql 是哪个引擎，各引擎之间有什么区别？(2018-4-16-lxy).....	193
9. 数据库的优化？(2018-4-16-lxy).....	194

10. Mysql 数据库如何分区、分表 ? (2018-4-16-lxy).....	195
11. 如何对查询命令进行优化 ? (2018-4-16-lxy).....	195
12. Sql 注入是如何产生的，如何防止 ? (2018-4-16-lxy).....	196
13. NoSQL 和关系数据库的区别 ? (2018-4-16-lxy).....	197
14. Mysql 数据库中怎么实现分页 ? (2018-4-23-zcz).....	198
15. sql 语句怎么看效率 ? (2018-4-23-zcz).....	198
16. 优化数据库 ? 提高数据库的性能 ? (2018-4-23-zcz).....	198
17. 提取数据库中倒数 10 条数据 ? (2018-4-23-zcz).....	200
18. 数据库负载均衡(2018-4-23-zcz).....	200
19. Mysql 数据库的操作?(2018-5-1-lxy).....	202
20. 数据库的设计 ? (2018-5-1-lxy).....	203
21. 存储过程和函数的区别?(2018-5-1-lxy).....	203
22. Mysql 日志 (2018-5-1-lyf).....	203
二 . MongoDB.....	204
1. 数据库的一些基本操作命令（列举一些常用命令即可） ? (2018-4-23-zcz).....	204
2. Python 中调用 mongo 数据库的包叫什么 ? (2018-4-23-zcz).....	205
3. MongoDB (2018-4-23-zcz).....	205
4. MongoDB 成为优秀的 NoSQL 数据库的原因是什么? (2018-4-23-zcz).....	207
5. 分析器在 MongoDB 中的作用是什么? (2018-4-23-zcz).....	208
6. 怎么查看 MongoDB 正在使用的链接? (2018-4-23-zcz).....	208
7. MySQL 与 MongoDB 本质之间最基本的差别是什么(2018-4-23-zcz).....	208
8. 使用 MongoDB 的优点(2018-4-23-zcz).....	209



9. 关于 MongoDB 更多知识请扫描二维码查看(2018-4-23-zcz).....	210
三 . Redis.....	210
1. Redis 中 list 底层实现有哪几种？有什么区别？(2018-4-16-lxy).....	210
2. 怎样解决数据库高并发的问题？(2018-4-23-zcz).....	211
3. Redis 集群实现？(2018-4-23-zcz).....	214
4. Redis 数据库，内容是以何种结构存放在 Redis 中的？(2018-4-23-zcz).....	214
5. Redis 的并发竞争问题怎么解决？(2018-4-23-zcz).....	214
6. MySQL 和 Redis 高可用性体现在哪些方面？(2018-4-23-zcz).....	215
7. Redis 和 MongoDB 的优缺点(2018-4-23-zcz).....	216
8. Redis 基本类型、相关方法(2018-4-23-zcz).....	217
9. Redis 的事务？(2018-4-23-zcz).....	218
10. Redis 的使用场景有哪些？(2018-4-23-zcz).....	219
11. Redis 默认端口，默认过期时间，Value 最多可以容纳的数据 长度？(2018-4-23-zcz).....	219
12. Redis 有多少个库？(2018-4-23-zcz).....	220
第十章 人工智能.....	220
一 . 数据挖掘.....	220
1. 1G 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。（2018-4-23-xhq）.....	220
2. 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。（2018-4-23-xhq）.....	221
3. 怎么在海量数据中找出重复次数最多的一个？（2018-4-23-xhq）.....	221

4. 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？.....	223
二．机器学习.....	223
三．深度学习.....	223
第十一章 数据结构与算法.....	223
1. 算法的特征？(2018-4-16-lxy).....	223
2. 冒泡排序的思想？(2018-4-16-lxy).....	224
3. 快速排序的思想？(2018-4-16-lxy).....	224
4. 如何判断单向链表中是否有环？(2018-4-16-lxy).....	225
5. 基础的数据结构有哪些？(2018-4-23-lyf).....	225
6. 基本的算法有哪些，怎么评价一个算法的好坏？(2018-4-23-lyf).....	226
7. 哪种数据结构可以实现递归？(2018-4-23-lyf).....	226
8. 你知道哪些排序算法（一般是通过问题考算法）(2018-4-23-lyf).....	227
9. 斐波那契数列(2018-4-23-lyf).....	227
10. 二叉树如何求两个叶节点的最近公共祖先？(2018-4-23-lyf).....	227
11. 两个字符串，如何求公共字符串？(2018-4-23-lyf).....	228
12. 找出二叉树中最远结点的距离？(2018-4-23-lyf).....	228
13. 写一个二叉树(2018-4-23-lyf).....	228
14. 写一个霍夫曼数(2018-4-23-lyf).....	229
15. 写一个二分查找(2018-4-23-lyf).....	230
16. set 用 in 时间复杂度是多少，为什么？(2018-4-23-lyf).....	230
17. 深度优先遍历和广度优先遍历的区别？(2018-4-23-lyf).....	231

18. 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 ;(2018-4-23-lyf)	232
19. 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? (2018-4-23-lyf)	232
20. 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(2018-4-23-lyf)	233
21. 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）(2018-4-23-lyf)	233
22. 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 ) (2018-4-23-lyf)	234
23. 桶排序（最快最简单的排序）(2018-4-23-lyf)	235
24. 青蛙跳台阶问题(2018-4-23-lyf)	235
25. 删除排序数组中的重复数字 Remove Duplicates from Sorted Array(2018-4-23-lyf)	237
26. 两数之和 Two Sum(2018-4-23-lyf)	239
27. 删除元素 Remove Element(2018-4-23-lyf)	241
28. 加一 Plus One(2018-4-23-lyf)	242
29. 用两个队列如何实现一个栈，用两个栈如何实现一个队列？(2018-4-23-lyf)	243
30. 爬楼梯 Climbing Stairs(2018-4-23-lyf)	243
31. 落单的数 Single Number(2018-4-23-lyf)	244
32. 搜索旋转排序数组 Search in Rotated Sorted Array(2018-4-23-lyf)	245

33. 三数之和 3Sum(2018-4-23-lyf).....	248
34. 四数之和 4Sum(2018-4-23-lyf).....	249
35. 下一个排列 Next Permutation(2018-4-23-lyf).....	251
36. 第 K 个排列 Permutation Sequence(2018-4-23-lyf).....	252
37. 判断数独是否合法 Valid Sudoku(2018-4-23-lyf).....	253
38. 链表求和 Add Two Numbers(2018-4-23-lyf).....	255
39. 链表划分 Partition List(2018-4-23-lyf).....	257
40. 如何翻转一个单链表 ? (2018-4-23-lyf).....	258
41. 旋转链表(2018-4-23-lyf).....	259
42. 两两交换链表中的节点(2018-4-23-lyf).....	260
43. 重排链表(2018-4-23-lyf).....	261
第十二章 企业真题实战.....	263
一、 360 面试题.....	263
1. 请拿出 B 表中的 accd , (A 表中和 B 表中一样的数据) ? (2018-4-16-lxy).....	263
2. a = “abbbccc”, 用正则匹配为 abccc,不管有多少 b , 就出现一次 ? (2018-4-16-lxy).....	263
3. xpath 使用的什么库 ? (2018-4-16-lxy).....	263
4. py2 和 py3 的区别 ? (2018-4-16-lxy).....	264
5. Redis 里面 list 内容的长度 ? (2018-4-16-lxy).....	264
6. 多线程交互, 访问数据, 如果访问到了就不访问了, 怎么避免重读 ? (2018-4-16-lxy).....	264
7. Mysql 怎么限制 IP 访问 ? (2018-4-16-lxy).....	264
8. 带参数的装饰器? (2018-4-16-lxy).....	264
二、 妙计旅行面试题.....	266

1. Python 主要的内置数据类型有哪些 ? (2018-4-16-lxy).....	266
2. print(dir('a'))输出的是什么 ? (2018-4-16-lxy).....	266
3. 给定两个 list , A 和 B , 找出相同元素和不同元素 ? (2018-4-16-lxy).....	266
4. 请反转字符串 ? (2018-4-16-lxy).....	266
5. 交换变量 a,b 的值 ? (2018-4-16-lxy).....	267
6. 用 select 语句输出每个城市中心距离市中心大于 20km 酒店数 ? (2018-4-16-lxy).....	267
7. 给定一个有序列表 , 请输出要插入值 k 所在的索引位置 ? (2018-4-16-lxy).....	267
8. 正则表达式贪婪与非贪婪模式的区别 ? (2018-4-16-lxy).....	267
9. 写出开头匹配字母和下划线 , 末尾是数字的正则表达式 ? (2018-4-16-lxy).....	267
10. 请说明 HTTP 状态码的用途 , 请说明常见的状态码机器意义 ? (2018-4-16-lxy).....	267
11. 当输入 http://www.itheima.com 时 , 返回页面的过程中发生了什么 ? (2018-4-16-lxy).....	268
12. 有一个多层嵌套列表 A=[1,2,[3.4["434",[...]]]]请写一段代码遍历 A 中的每一个元素并打印出来。 (2018-5-2-xhq).....	268
13. 关系型数据库中 , 表和表之间有左连接 , 内连接 , 外连接 , 分别解释下他们的含义和区别 ? (2018-5-2-xhq).....	269
14. 如何定时启动你的爬虫项目 : ( 2018-5-2-xhq ) .....	270
15. 什么是 scrapy-redis 中的指纹,是如何去重的 ? ( 2018-5-2-xhq ) .....	271
16. 代码优化从哪些方面考虑 ? 有什么想法 ? ( 2018-5-2-xhq ) .....	271
17. Django 项目的优化 ( web 通用 ) ( 2018-5-2-xhq ) .....	273
三、 智慧星光面试题.....	274
1. 定义 A=(“a”, “b”, “c”, “d”),执行 delA[2]后的结果为 : D(2018-5-1-lxy).....	274

2. String = “{1},{0}”; string = string.format(“Hello”, “Python”),请问将 string 打印出来为 ( C )	
(2018-5-1-lxy).....	275
3. 定义 A=[1,2,3,4],使用列表生成式[i*i for i in A]生成列表为 : B(2018-5-1-lxy).....	275
4. 请对 Python 数据结构 Tuple,List,Dict 进行操作(2018-5-1-lxy).....	275
5. 请用 Python 内置函数处理以下问题 ? (2018-5-1-lxy).....	276
7. 现在需要从一个简单的登陆网站获取信息 ,请使用 Python 写出简要的登陆函数的具体实现 ? ( 登录信息只包含用户名 , 密码 ) (2018-5-1-lxy).....	278
8. 正则表达式操作(2018-5-1-lxy).....	278
四、 壹讯面试题.....	278
1. Python 中 pass 语句的作用是什么 ? (2018-5-1-lxy).....	278
2. 尽可能写出多的 str 方法 ? (2018-5-1-lxy).....	279
3. 生成一个斐波那契数列 ? (2018-5-1-lxy).....	283
4. 说明一下 os.path 和 sys.path 分别代表什么 ? (2018-5-1-lxy).....	284
5. 什么是 lambda 函数 ? 有什么好处 ? (2018-5-1-lxy).....	284
五、 H3C 面试题.....	284
1. 下列哪个语句在 Python 中是非法的?(B)(2018-5-1-lxy).....	284
2. 关于 Python 内存管理,下列说法错误的是(B)(2018-5-1-lxy).....	285
3. 下面哪个不是 Python 合法的标识符(b)(2018-5-1-lxy).....	285
4. 下列哪种说法是错误的 ( A ) (2018-5-1-lxy).....	285
5. 下列表达式的值为 True 的是 ( C ) (2018-5-1-lxy).....	285
6. Python 不支持的数据类型有(A)(2018-5-1-lxy).....	286
7. 关于 Python 中的复数,下列说法错误的是(C)(2018-5-1-lxy).....	286

六、 通联数据 (2018-5-2-lyf).....	286
1. 说一下你对多线程的看法?.....	286
2. 多线程和多线程有什么区别?.....	286
3. 进程间的数据共享和线程间数据共享?.....	286
4. Redis 数据库结构有那些?.....	288
5. 你一般用 Redis 来做什么?.....	288
6. MonggoDB 中存入了 100 万条数据，如何提高查询速度?.....	288
7. 如何提高并发性能?.....	289
8. 说一下你对 Django 框架的认识?.....	290
9. Flask 框架了解吗，说下 Flask 和 Django 的区别?.....	290
10. 有一个无序数组，如何获取第 K 大的数，说下思路，实现后的时间复杂度?.....	292
11. 归并排序的时间复杂度?.....	292
七、 北京号外科技爬虫面试题 ( 2018-5-2 lyf ) .....	294
1. 单引号、双引号、三引号的区别?.....	294
2. 如何在一个 function 里面设置一个全局变量?.....	294
3. 描述 yield 使用场景?.....	294
4. 生成 1~10 之间的整数?.....	295
5. Python 如何生成缩略图?.....	295
6. 列出比较熟悉的爬虫框架，并简要说明?.....	296
7. 列举常见的反爬技术，并给出应对方案?.....	296
8. 网络协议 http 和 https 区别?.....	299
9. 什么是 cookie，session 有什么区别?.....	299

10. Mysql 中 myisam 与 innodb 的区别？ .....	300
八、 首信 Python 研发面试题（2018-5-2 lyf） .....	302
1. Python 中 list、tuple、dict、set 有什么区别，主要应用在什么样的场景？并用 for 语句遍历？ .....	302
2. Python 中静态函数、类函数、成员函数的区别？并写一个示例？ .....	303
3. 用 Python 语言写一个函数，输入一个字符串，返回倒序结果？ .....	304
4. 介绍一下 Python 的异常处理机制和自己开发过程中的体会？ .....	304
5. jQuery 库中 \$() 是什么？网上有 5 个 <div> 元素，如何使用 jQuery 来选择它们？ .....	306
6. 写一个 Bash Shell 脚本来得到当前的日期、时间、用户名和当前工作目录？ .....	307
7. Django 中使用 memcached 作为缓存的具体方法？有缺点说明？ .....	307
九、 微影时代（2018-5-2 lyf） .....	307
1. 装饰器的理解？ .....	307
2. scrapy 框架，反扒机制，手写中间件？ .....	308
3. HTTP 头有什么字段？ .....	308
4. POST 登录数据方式？ .....	309
5. 加密页面如何解析？ .....	309
6. 爬虫如何定时增量更新数据 .....	310
7. scrapy 怎么设置时间间隔？ .....	310
8. 爬虫遇到验证码如何解决？ .....	311
十、 斯沃创智 .....	311
1. 简述 Python 中 is 和 = 的区别（2018-5-2-zcz） .....	311
2. 简述 read, readline 和 readlines 的区别（2018-5-2-zcz） .....	312



3. 举例说明创建字典的至少两种方法 ( 2018-5-2-zcz ) .....	312
4. 简述 Python 里面 search 和 match 的区别 ( 2018-5-2-zcz ) .....	312
5. Python 代码实现:删除一个 list 里面的重复元素 ( 2018-5-2-zcz ) .....	313
6. Python 代码中(*args, **kwargs)是什么意思 ( 2018-5-2-zcz ) .....	314
十一、 天广汇通.....	314
1. 说明 os.path 和 sys.path 分别代表什么? ( 2018-5-2-zcz ) .....	314
2. 解释一下并行 ( parallel ) 和并发 ( concurrency ) 的区别 ( 2018-5-2-zcz ) .....	314
3. 在 Python 中可以实现并发的库有哪些? ( 2018-5-2-zcz ) .....	314
4. 如果一个程序需要进行大量的 IO 操作, 应当使用并行还是并发? ( 2018-5-2-zcz ) .....	315
5. 如果程序需要进行大量的逻辑运算操作, 应当使用并行还是并发? ( 2018-5-2-zcz ) .....	315
十二、 信德数据.....	315
1. 网络七层协议是哪几层? HTTP 协议输入是第几层? ( 2018-5-2-zcz ) .....	315
2. 什么是 HTTP 协议? HTTP 请求有哪几种? ( 2018-5-2-zcz ) .....	315
3. 什么是 HTTP 代理? 作用是什么? ( 2018-5-2-zcz ) .....	316
4. 什么是反向代理? 作用是什么? ( 2018-5-2-zcz ) .....	316
5. HTTPS 和 HTTP 的区别 ( 2018-5-2-zcz ) .....	316
6. 什么是进程? 什么是协程? ( 2018-5-2-zcz ) .....	316
7. 什么是死锁? 死锁产生的四个必要条件? .....	317
8. 什么是内存泄漏? ( 2018-5-2-zcz ) .....	317
9. Python 代码中(*args, **kwargs)是什么意思? ( 2018-5-2-zcz ) .....	317
十三、 成安.....	317
1. Python 中使用%与.format 格式化文本 ( 2018-5-2-zcz ) .....	317

2. Python 的 logging 模块常用的几个等级？（2018-5-2-zcz）	317
3. 在 HTTP1.1 中常见的状态码有哪些，如何设置状态码？（2018-5-2-zcz）	318
4. Python 如何处理上传文件？（2018-5-2-zcz）	318
5. 用 Python 写一段排序算法。（2018-5-2-zcz）	319
十四、 博派通达	320
1. 请列举你使用过的 Python 代码检测工具(2018-5-2-xhq)	320
2. 简述 Python 垃圾回收机制和如何解决循环引用(2018-5-2-xhq)	321
3. 请简述如何编写清晰可读的代码（2018-5-2-xhq）	323
4. 请列举出常见的 Mysql 存储引擎（2018-5-2-xhq）	325
5. InnoDB 有哪些特性（2018-5-2-xhq）	325
6. 请列出 MySQL 数据库查询的技巧（2018-5-2-xhq）	325
7. 请简述 SQL 注入的原理及如何在代码层面防止 SQL 注入（2018-5-2-xhq）	328
8. 请列出常见的 HTTP 头及其作用（2018-5-2-xhq）	328
9. 请列举常见的 HTTP 状态码及其意义（2018-5-2-xhq）	330
10. 请简述 RESTfulAPI 设计规范的理解（2018-5-2-xhq）	331
11. 请简述标准库中 functools.wraps 的作用（2018-5-2-xhq）	335
十五、 乐飞天下	335
1. 如何判断一个 python 对象的类型（2018-5-2-xhq）	335
2. Python 里面如何生成随机数（2018-5-2-xhq）	335
3. 请写出匹配 ip 的 Python 正则表达式（2018-5-2-xhq）	335
4. 尽可能多的 str 方法（2018-5-2-xhq）	336

5. 全局变量和局部变量的区别，如何在 function 里面给一个全局变量赋值 ( 2018-5-2-xhq )	336
6. Tuple 和 list 的区别，有两个 list b1 = [1,2,3] b2=[2,3,4]写出合并代码 ( 2018-5-2-xhq )	337
7. 请写出一段代码删除一个 list 里面的重复元素 l=[1,1,2,3,4,5,4](2018-5-2-xhq).....	338
8. 写出 list 的交集与差集的代码 b1 =[1,2,3] b2=[2,3,4](2018-5-2-xhq).....	338
9. 写一段 Python 代码实现 list 里排 a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3](2018-5-2-xhq).....	338
10. D= [i**2 for i in range(1,10)]请写出 D 的输出结果 ( 2018-5-2-xhq ) .....	339
11. 什么时 lambda 函数，下面这段代码的输出是什么 ( 2018-5-2-xhq ) .....	339
12. 说说用过的几种爬虫框架及他们的优缺点，用过哪些解析 html 的库 ( 2018-5-2-xhq )	339
13. 谈一下对于多线程的理解，对于 cpu 密集性 IO 密集性怎样使用线程，说说线程池，线程锁的用法，有么有用过 multiprocessing 或者 concurrent.futures?(2018-5-2-xhq).....	339
十六、 莉莉丝广告开发工程师初始题目.....	341
1. 用递归实现快速排序 quik_sort(A)(2018-5-2-xhq).....	341
2. 简述 Python 在异常处理中，else 和 finally 的作用分别是什么？ ( 2018-5-2-xhq ) ....	342
3. 简述 Python GIL 的概念，以及它对 Python 多线程的影响。如何实现一个抓取网页的程序，使用多线程是否比单线程有性能提升，并解释原因 ( 2018-5-2-xhq ) .....	342
十七、 罗格数据.....	343
1. 滑动验证码如何解决 ( 2018-5-2-xhq ) .....	343
2. ajax 请求页面如何加载 ( 2018-5-2-xhq ) .....	344
3. Selenium+Phantom JS 的使用(2018-5-2-xhq).....	344
4. 数据如何存储，数据库字段如何设计 ( 2018-5-2-xhq ) .....	345
5. 加密页面如何解析 ( 2018-5-2-xhq ) .....	346

6. HTTPS 网站如何爬取(2018-5-2-xhq).....	347
十八、 牧游科技.....	347
1. 函数参数传递，下面程序运行的结果是？（2018-5-2-xhq）.....	347
2. Python 中类方法，静态方法的区别及调用（2018-5-2-xhq）.....	347
3. 类变量，实例变量（2018-5-2-xhq）.....	347
4. 函数式编程与内置函数（2018-5-2-xhq）.....	348
十九、 上海金台灯.....	348
1. 什么是 lambda 函数，它有什么好处（2018-5-2-xhq）.....	348
2. 什么是 Python 的 list and dict comprehensions,写一段代码（2018-5-2-xhq）.....	348
3. Python 里面如何实现 tuple 和 list 的转换（2018-5-2-xhq）.....	349
4. Python 里面如何拷贝一个对象（2018-5-2-xhq）.....	349
5. 写一段 except 的函数（2018-5-2-xhq）.....	349
6. Python 里面 pass 语句的作用是什么？(2018-5-2-xhq).....	349
7. 如何知道 Python 对象的类型？(2018-5-2-xhq).....	349
8. Python 中 range()函数的用法(2018-5-2-xhq).....	350
9. Python re 模块匹配 HTML tag 的时候，<.*>和<.*?>有什么区别(2018-5-2-xhq).....	350
10. Python 里面如何生成随机数(2018-5-2-xhq).....	350
11. 如何在 function 里面设置一个全局变量(2018-5-2-xhq).....	350
12. Python 程序中中文乱码如何解决(2018-5-2-xhq).....	350
13. Python 的传参是传值还是传址(2018-5-2-xhq).....	350
14. with 语句的作用,写一段代码(2018-5-2-xhq).....	351
二十、 钱方好近.....	351

1. 求字符串" 是一个 test 字符串" 的字符个数字符编码为 utf8(2018-5-2-xhq).....	351
2. 一个 list 对象 a = [1,2,4,3,2,2,3,4]需要去掉里面重复的的值(2018-5-2-xhq).....	351
3. 有一个文件 test.txt 里面有数据(2018-5-2-xhq).....	351
二十一、 西北莜面村.....	352
1. 列举出一些常用的设计模式 ? (2018-5-11-lxy).....	352
2. Python 关键字 yield 的用法 ? (2018-5-11-lxy).....	354
3. 深拷贝 , 浅拷贝的区别 ? (2018-5-11-lxy).....	354
4. 简化代码 ? (2018-5-11-lxy).....	354
5. 解释 list dict tuple set 区别 ? (2018-5-11-lxy).....	355
6. 这两个参数什么意思 *args **kwargs , 我们为什么要使用他们 ? (2018-5-11-lxy).....	355
7. 数据库连表查询 ? (2018-5-11-lxy).....	355
二十二、 浙江从泰.....	355
1. 数据库优化 ? (2018-5-11-lxy).....	355
2. 反爬虫措施 ? (2018-5-11-lxy).....	355
3. 分布式爬虫原理 ? (2018-5-11-lxy).....	357
二十三、 tataUFO.....	358
1. 将字符串 : "k:1 k1:2 k2:3 k3:4" , 处理成 Python 字典 : {k:1 , k1:2 , ...} # 字典里的 K 作为字符串处理(2018-5-11-lxy).....	358
2. 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 ? (2018-5-11-lxy).....	358
3. 写一个装饰器 ? (2018-5-11-lxy).....	358
4. Python 中可变类型和不可变类型有哪些 ? (2018-5-11-lxy).....	359

二十四、 全品教育.....	360
1. Tuple 和 list 区别 ( 2018-5-11-xhq ) .....	360
2. 这两个参数*args **kwargs 是什么意思 ( 2018-5-11-xhq ) .....	360
3. Python 里面如何实现 tuple 和 list 的转换 ( 2018-5-11-xhq ) .....	360
4. Python 里面 range 和 xrange 的区别(2018-5-11-xhq).....	360
5. Python 里面 classmethod 和 staticmethod 的区别 ( 2018-5-11-xhq ) .....	361
6. 如何反向输出序列比如[2,6,5,3],输出为[3,5,6,2](2018-5-11-xhq).....	361
7. Python 里面实现删除重复的元素(2018-5-11-xhq).....	361
8. Python 里面 copy 和 deepcopy 的区别(2018-5-11-xhq).....	361
9. Python 里面的 search 和 match 的区别(2018-5-11-xhq).....	362
10. 输出下列代码的结果 ( 2018-5-11-xhq ) .....	362
11. Python 代码如何得到列表的交集和差集 ? ( 2018-5-11-xhq ) .....	363
12. 请写出一段代码求出 1 到 100 的和 ? (2018-5-11-xhq).....	363
13. Python 中正则表达式提取出字符串中的数字(2018-5-11-xhq).....	363
14. 补全下列代码(2018-5-11-xhq).....	363
二十五、 名企爬虫面试题.....	364
1. 简述一次完整的 http 的通信过程、常用的响应状态码、http 的无状态性、Cookies 等这些概念(2018-5-11-xhq).....	364
2. 说说进程和线程和锁之间的关系(2018-5-17-ydy).....	366
3. MySQL 操作 : 为 person 表的 name 创建普通的索引(2018-5-11-xhq).....	367
4. *args and **kwargs 的区别(2018-5-11-xhq).....	367
5. 写一个匹配 Email 地址的正则表达式(2018-5-11-xhq).....	367

6. 常见的反爬虫措施有哪些？一般怎么克服(2018-5-11-xhq).....	368
7. 编写爬虫的常用模块或者框架有哪些?请说明一个爬虫的行为步骤(2018-5-11-ydy).....	369
8. 排序算法有哪些用 Python 写一种排序算法(2018-5-11-xhq).....	370
二十六、 欧特咨询.....	372
1. 对 Cookie 的理解，遇到没有 Cookie 登陆的问题(2018-5-11-xhq).....	372
2. 如何解决验证码的问题，用什么模块，听过哪些人工打码平台(2018-5-11-xhq).....	372
3. 对于 scrapy_redis 的理解(2018-5-11-xhq).....	372
4. ip 被封了怎么解决，自己做过 ip 池么？(2018-5-11-xhq).....	373
二十七、 多来点.....	374
1. 请阐述 Python 的特点(2018-5-8-zcz).....	374
2. Python 字典参数如何传递？(2018-5-8-zcz).....	375
3. 请举例说明 Python 闭包的使用场景(2018-5-8-zcz).....	375
4. 阐述 Python 变量作用域(2018-5-8-zcz).....	376
二十八、 傲盾.....	376
1. Python 线程和进程的区别？(2018-5-8-zcz).....	376
2. 如何保证线程安全？(2018-5-8-zcz).....	376
3. 编程实现 list 转 dict(2018-5-8-zcz).....	377
4. 编程实现两个 list 的交集、并集、差集(2018-5-8-zcz).....	377
二十九、 汉迪.....	378
1. 在 Python 中，list，tuple，dict，set 有什么区别，主要应用在什么场景？(2018-5-8-zcz) .....	378
2. 说明 Session 和 Cookie 的联系(2018-5-8-zcz).....	378

3. 说明 Session 和 Cookie 的区别(2018-5-8-zcz).....	379
4. 简述 decorator 的用法(2018-5-8-zcz).....	380
三十、 大会信统 Python 工程师 ( 2018-5-9-lyf ) .....	380
1. 请写出一段 python 代码实现删除一个 list 里面的重复元素.....	380
2. 编程用 sort 进行排序，然后从最后一个元素开始判断.....	381
3. Python 里面如何拷贝一个对象？（赋值，浅拷贝，深拷贝的区别）.....	382
4. Python 里面 match()和 search()的区别？.....	382
5. 用 Python 匹配 HTML tag 的时候，<.*>和<.*?>有什么区别？.....	382
6. Python 里面如何生成随机数？.....	383
三十一、 倍通供应链 信息&数据中心工程师 笔试题 ( 2018-5-9 lyf ) .....	383
1. OOP 编程三大特点是什么，多态应用的基础是什么？.....	383
2. 请描述抽象类和接口类的区别和联系？.....	384
3. 请解释委托的定义和作用？.....	384
4. 请描述方法重载与方法重写？.....	385
5. 请用代码实现一个冒泡排序？.....	385
6. 请用代码实现输出：1，2，3，5，8，13，21，34，55，89.....	385
7. 请解释下 TCP/IP 协议和 HTTP 协议？.....	387
8. Python 里面如何实现 tuple 和 list 的转换？.....	387
9. 请写出以下 Linux 的 SHELL 命令？.....	387
三十二、 上海行知道教育 Python 程序员笔试题 ( 2018-5-9 lyf ) .....	388
1. Python 如何实现单例模式？请写出两种实现方法？.....	388
2. 什么是 lambda 函数？请举例说明？.....	390



3. 如何反序地迭代一个序列？ .....	391
4. Python 如何生成随机数？ .....	392

# 第一章 内容介绍

该宝典是一份知识点全面又能不断更新，与时俱进的学习手册，不仅收录了作者亲身面试遇到的问题，还收录了近上万名黑马学子面试时遇到的问题。我们会一直不断地更新和充实该宝典，同时也希望读者朋友能够多多提供优质的面试题，也许下一个版本就有你提供的面试题哦。

该面试的最新版发布在黑马论坛：<http://bbs.itheima.com/thread-402667-1-1.html>

**注意**：该面试宝典仅供参考，由于作者的知识水平有限加之编写时间仓促因此难免有 bug 的存在，希望大家见谅。

该宝典的一个明确目标是能够让 90% 以上的 Python 技术面试题都落到该宝典中，如果您有不错的知识或者面试题，您可以发送到 [wangzhenyang@itcast.cn](mailto:wangzhenyang@itcast.cn)，本人将不胜感激。让天下没有难学的知识，希望你我的努力能帮到更多的莘莘学子。

世间事，很多都可投机取巧，但技术却必须靠日积月累的努力来提高。本宝典更加注重的是知识的掌握，而不仅仅是对面试题的应付。在展示常见的面试问题以及回答技巧的同时还详细讲解了每一道题所包含的知识点，让读者不仅知其然，更知其所以然。

参与该面试宝典收集、整理、编写、审核的人员名单如下：刘鑫洋、刘永峰、辛华侨、张程喆、宋美龙、闫大宇、樊杨俊、郭帅、苏洵、王刚、于健、张晓楠、左文彬、沈夏冬。

## 第二章 Python 基础

### 一、基础语法

#### 1. 输入与输出

##### 1.1 代码中要修改不可变数据会出现什么问题？抛出什么异常？(2018-3-29-lxy)

代码不会正常运行，抛出 `TypeError` 异常。

##### 1.2 $a=1, b=2$ , 不用中间变量交换 $a$ 和 $b$ 的值？(2018-3-29-lxy)

方法一：

```
1. a = a+b
2. b = a-b
3. a = a-b
```

方法二：

```
1. a = a^b
2. b = b^a
3. a = a^b
```

方法三：

```
1. a,b = b,a
```

##### 1.3 `print` 调用 Python 中底层的什么方法？(2018-3-30-lxy)

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

##### 1.4 下面这段代码的输出结果将是什么？请解释？(2018-3-30-lxy)

```
1. class Parent(object):
```

```
2.     x = 1
3.     class Child1(Parent):
4.         pass
5.     class Child2(Parent):
6.         pass
7.     print Parent.x, Child1.x, Child2.x
8.     Child1.x = 2
9.     print parent.x, Child1.x, Child2.x
10.    parent.x = 3
11.    print Parent.x, Child1.x, Child2.x
```

结果为：

1 1 1 #继承自父类的类属性 x，所以都一样，指向同一块内存地址。

1 2 1 #更改 Child1，Child1 的 x 指向了新的内存地址。

3 2 3 #更改 Parent，Parent 的 x 指向了新的内存地址。

## 1.5 简述你对 input()函数的理解？

在 Python3 中，input()获取用户输入，不论用户输入的是什么，获取到的都是字符串类型的。

在 Python2 中有 raw\_input()和 input()，raw\_input()和 Python3 中的 input()作用是一样的，input()输入的是什么数据类型的，获取到的就是什么数据类型的。

## 2. 条件与循环

### 2.1 阅读下面的代码，写出 A0，A1 至 An 的最终值。(2018-3-30-lxy)

```
1.  A0 = dict(zip(('a', 'b', 'c', 'd', 'e'), (1, 2, 3, 4, 5)))
2.  A1 = range(10)
3.  A2 = [i for i in A1 if i in A0]
4.  A3 = [A0[s] for s in A0]
5.  A4 = [i for i in A1 if i in A3]
6.  A5 = {i:i*i for i in A1}
7.  A6 = [[i, i*i] for i in A1]
```

答：

```
1.  A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}
```

```
2. A1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3. A2 = []
4. A3 = [1, 3, 2, 5, 4]
1. A4 = [1, 2, 3, 4, 5]
2. A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
3. A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]
```

## 2.2 range 和 xrange 的区别 ? (2018-3-30-lxy)

两者用法相同，不同的是 range 返回的结果是一个列表，而 xrange 的结果是一个生成器，前者是直接开辟一块内存空间来保存列表，后者是边循环边使用，只有使用时才会开辟内存空间，所以当列表很长时，使用 xrange 性能要比 range 好。

## 2.3 考虑以下 Python 代码，如果运行结束，命令行中的运行结果是什么？

(2018-3-30-lxy)

```
1. l = []
2. for i in xrange(10):
3.     l.append({'num': i})
4. print l
```

在考虑以下代码，运行结束后的结果是什么？

```
1. l = []
2. a = {'num': 0}
3. for i in xrange(10):
4.     a['num'] = i
5.     l.append(a)
6. print l
```

以上两段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：

```
1. [{ '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：

```
1. [{ '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。

## 2.4 以下 Python 程序的输出？(2018-3-30-lxy)

```
1. for i in range(5, 0, -1):  
2.     print(i)
```

答：5 4 3 2 1

## 3. 文件操作

### 3.1 4G 内存怎么读取一个 5G 的数据？(2018-3-30-lxy)

方法一：

可以通过生成器，分多次读取，每次读取数量相对少的数据（比如 500MB）进行处理，处理结束后在读取后面的 500MB 的数据。

方法二：

可以通过 linux 命令 `split` 切割成小文件，然后再对数据进行处理，此方法效率比较高。可以按照行数切割，可以按照文件大小切割。

更多知识请点击网址或扫描二维码。

<https://blog.csdn.net/mxgsgtc/article/details/12048919>



3.2 现在考虑有一个 jsonline 格式的文件 file.txt 大小约为 10K，之前处理文件的代码如下所示：(2018-3-30-lxy)

```
1. def get_lines():
2.     l = []
3.     with open( 'file.txt' , 'rb' ) as f:
4.         for eachline in f:
5.             l.append(eachline)
6.     return l
7. if __name__ == '__main__':
8.     for e in get_lines():
9.         process(e) #处理每一行数据
```

现在要处理一个大小为 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？

```
1. def get_lines():
2.     l = []
3.     with open( 'file.txt' , 'rb' ) as f:
4.         data = f.readlines(60000)
5.         l.append(data)
6.     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。分批读入数据要记录每次读入数据的位置。分批每次读入数据的大小，太小就会在读取操作上花费过多时间。

### 3.3 read、readline 和 readlines 的区别? (2018-4-16-lxy)

read:读取整个文件。

readline：读取下一行，使用生成器方法。

readlines：读取整个文件到一个迭代器以供我们遍历。

### 3.4.补充缺失的代码？(2018-4-16-lxy)

```
1 . def print_directory_contents(sPath):
2 .     """
3 .     这个函数接收文件夹的名称作为输入参数
4 .     返回该文件夹中文件的路径
5 .     以及其包含文件夹中文件的路径
6 .     """
7 .     # 补充代码
8 .     -----代码如下-----
9 .     import os
10 .     for sChild in os.listdir(sPath):
11 .         sChildPath = os.path.join(sPath, sChild)
12 .         if os.path.isdir(sChildPath):
13 .             print_directory_contents(sChildPath)
14 .         else:
15 .             print(sChildPath)
```

## 4. 异常

### 4.1 在 except 中 return 后还会不会执行 finally 中的代码？怎么抛出自定义异常？

(2018-3-30-lxy)

会继续处理 finally 中的代码；用 raise 方法可以抛出自定义异常。



## 4.2 介绍一下 except 的作用和用法 ? (2018-4-16-lxy)

except: #捕获所有异常

except: <异常名>: #捕获指定异常

except:<异常名 1, 异常名 2> : 捕获异常 1 或者异常 2

except:<异常名>,<数据>:捕获指定异常及其附加的数据

except:<异常名 1,异常名 2>:<数据>:捕获异常名 1 或者异常名 2,及附加的数据

## 5. 模块与包

### 5.1 常用的 Python 标准库都有哪些 ? (2018-3-30-lxy)

os 操作系统 ,time 时间 ,random 随机 ,pymysql 连接数据库 ,threading 线程 ,multiprocessing 进程 , queue 队列。

第三方库 :

django 和 flask 也是第三方库 , requests , virtualenv , selenium , scrapy , xadmin , celery , re , hashlib , md5。

常用的科学计算库 ( 如 Numpy , Scipy , Pandas)。

### 5.2 赋值、浅拷贝和深拷贝的区别 ? (2018-3-30-lxy)

#### 一、赋值

在 Python 中 , 对象的赋值就是简单的对象引用 , 这点和 C++不同 , 如下所示 :

```
16 . a = [1,2,"hello",['python', 'C++']]
17 . b = a
```

在上述情况下 , a 和 b 是一样的 , 他们指向同一片内存 , b 不过是 a 的别名 , 是引用。

我们可以使用 `b is a` 去判断，返回 `True`，表明他们地址相同，内容相同，也可以使用 `id()` 函数来查看两个列表的地址是否相同。

赋值操作(包括对象作为参数、返回值)不会开辟新的内存空间，它只是复制了对象的引用。也就是说除了 `b` 这个名字之外，没有其他的内存开销。修改了 `a`，也就影响了 `b`，同理，修改了 `b`，也就影响了 `a`。

## 二、浅拷贝(shallow copy)

浅拷贝会创建新对象，其内容非原对象本身的引用，而是原对象内第一层对象的引用。

浅拷贝有三种形式:切片操作、工厂函数、`copy` 模块中的 `copy` 函数。

比如上述的列表 `a`；

切片操作：`b = a[:]` 或者 `b = [x for x in a]`；

工厂函数：`b = list(a)`；

`copy` 函数：`b = copy.copy(a)`；

浅拷贝产生的列表 `b` 不再是列表 `a` 了，使用 `is` 判断可以发现他们不是同一个对象，使用 `id` 查看，他们也不指向同一片内存空间。但是当我们使用 `id(x) for x in a` 和 `id(x) for x in b` 来查看 `a` 和 `b` 中元素的地址时，可以看到二者包含的元素的地址是相同的。

在这种情况下，列表 `a` 和 `b` 是不同的对象，修改列表 `b` 理论上不会影响到列表 `a`。

但是要注意的是，浅拷贝之所以称之为浅拷贝，是它仅仅只拷贝了一层，在列表 `a` 中有一个嵌套的 `list`，如果我们修改了它，情况就不一样了。

比如：`a[3].append('java')`。查看列表 `b`，会发现列表 `b` 也发生了变化，这是因为，我们修改了嵌套的 `list`，修改外层元素，会修改它的引用，让它们指向别的位置，修改嵌套列表中的元素，列表的地址并未发生变化，指向的都是用一个位置。

## 三、深拷贝(deep copy)

深拷贝只有一种形式，copy 模块中的 `deepcopy()` 函数。

深拷贝和浅拷贝对应，深拷贝拷贝了对象的所有元素，包括多层嵌套的元素。因此，它的时间和空间开销要高。

同样的对列表 `a`，如果使用 `b = copy.deepcopy(a)`，再修改列表 `b` 将不会影响到列表 `a`，即使嵌套的列表具有更深的层次，也不会产生任何影响，因为深拷贝拷贝出来的对象根本就是一个全新的对象，不再与原来的对象有任何的关联。

#### 四、拷贝的注意事项？

对于非容器类型，如数字、字符，以及其他的“原子”类型，没有拷贝一说，产生的都是原对象的引用。

如果元组变量值包含原子类型对象，即使采用了深拷贝，也只能得到浅拷贝。

### 5.3 `__init__` 和 `__new__` 的区别？(2018-3-30-lxy)

`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`。

### 5.4 Python 里面如何生成随机数？(2018-3-30-lxy)

在 Python 中用于生成随机数的模块是 `random`，在使用前需要 `import`。如下例子可以酌情列举：

`random.random()`：生成一个 0-1 之间的随机浮点数；

`random.uniform(a, b)`：生成 `[a,b]` 之间的浮点数；

`random.randint(a, b)`：生成 `[a,b]` 之间的整数；

`random.randrange(a, b, step)`：在指定的集合 `[a,b]` 中，以 `step` 为基数随机取一个数；

`random.choice(sequence)`：从特定序列中随机取一个元素，这里的序列可以是字符串，列表，元组等。

### 5.5 输入某年某月某日，判断这一天是这一年的第几天？(可以用 Python 标准库)(2018-3-30-lxy)

```
1. import datetime
2. def dayofyear():
3.     year = input("请输入年份：")
4.     month = input("请输入月份：")
5.     day = input("请输入天：")
6.     date1 = datetime.date(year=int(year), month=int(month), day=int(day))
7.     date2 = datetime.date(year=int(year), month=1, day=1)
8.     return (date1 - date2 + 1).days
```

### 5.6 打乱一个排好序的 list 对象 alist？(2018-3-30-lxy)

```
1. import random
2. random.shuffle(alist)
```

### 5.7 说明一下 `os.path` 和 `sys.path` 分别代表什么？(2018-3-30-lxy)

`os.path` 主要是用于对系统路径文件的操作。

`sys.path` 主要是对 Python 解释器的系统环境参数的操作（动态的改变 Python 解释器搜索路径）。

### 5.8 Python 中的 `os` 模块常见方法？(2018-4-16-lxy)

- `os.remove()`删除文件
- `os.rename()`重命名文件
- `os.walk()`生成目录树下的所有文件名
- `os.chdir()`改变目录

- `os.mkdir/makedirs` 创建目录/多层目录
- `os.rmdir/removedirs` 删除目录/多层目录
- `os.listdir()` 列出指定目录的文件
- `os.getcwd()` 取得当前工作目录
- `os.chmod()` 改变目录权限
- `os.path.basename()` 去掉目录路径，返回文件名
- `os.path.dirname()` 去掉文件名，返回目录路径
- `os.path.join()` 将分离的各部分组合成一个路径名
- `os.path.split()` 返回 ( `dirname()`, `basename()` ) 元组
- `os.path.splitext()` (返回 `filename`, `extension`) 元组
- `os.path.getatime\ctime\mtime` 分别返回最近访问、创建、修改时间
- `os.path.getsize()` 返回文件大小
- `os.path.exists()` 是否存在
- `os.path.isabs()` 是否为绝对路径
- `os.path.isdir()` 是否为目录
- `os.path.isfile()` 是否为文件

## 5.9 Python 的 sys 模块常用方法 ? (2018-4-16-lxy)

- `sys.argv` 命令行参数 List，第一个元素是程序本身路径
- `sys.modules.keys()` 返回所有已经导入的模块列表
- `sys.exc_info()` 获取当前正在处理的异常类, `exc_type`、`exc_value`、`exc_traceback` 当前处理的异常详细信息

- `sys.exit(n)` 退出程序，正常退出时 `exit(0)`
- `sys.hexversion` 获取 Python 解释程序的版本值，16 进制格式如：0x020403F0
- `sys.version` 获取 Python 解释程序的版本信息
- `sys.maxint` 最大的 Int 值
- `sys.maxunicode` 最大的 Unicode 值
- `sys.modules` 返回系统导入的模块字段，key 是模块名，value 是模块
- `sys.path` 返回模块的搜索路径，初始化时使用 `PYTHONPATH` 环境变量的值
- `sys.platform` 返回操作系统平台名称
- `sys.stdout` 标准输出
- `sys.stdin` 标准输入
- `sys.stderr` 错误输出
- `sys.exc_clear()` 用来清除当前线程所出现的当前的或最近的错误信息
- `sys.exec_prefix` 返回平台独立的 python 文件安装的位置
- `sys.byteorder` 本地字节规则的指示器，big-endian 平台的值是 'big', little-endian 平台的值是 'little'
- `sys.copyright` 记录 python 版权相关的东西
- `sys.api_version` 解释器的 C 的 API 版本
- `sys.version_info` 元组则提供一个更简单的方法来使你的程序具备 Python 版本要求功能

## 5.10 unittest 是什么？(2018-4-16-lxy)

在 Python 中，unittest 是 Python 中的单元测试框架。它拥有支持共享搭建、自动测试、在测试中暂停代码、将不同测试迭代成一组，等的功能。

## 5.11 模块和包是什么？(2018-4-16-lxy)

在 Python 中，模块是搭建程序的一种方式。每一个 Python 代码文件都是一个模块，并可以引用其他的模块，比如对象和属性。

一个包含许多 Python 代码的文件夹是一个包。一个包可以包含模块和子文件夹。

## 6. Python 特性

### 6.1 Python 是强语言类型还是弱语言类型？(2018-3-30-lxy)

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。

### 6.2 谈一下什么是解释性语言，什么是编译性语言？(2018-3-30-lxy)

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，计算机才能执行高级语言编写的程序。

解释性语言在运行程序的时候才会进行翻译。

编译型语言写的程序在执行之前，需要一个专门的编译过程，把程序编译成机器语言（可执行文件）。

### 6.3 Python 中有日志吗？怎么使用？(2018-3-30-lxy)

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。

## 6.4 Python 是如何进行类型转换的 ? (2018-3-30-lxy)

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 , 进制之间的转换可以用 `int( 'str' , base=' n' )` 将特定进制的字符串转换为十进制 , 再用相应的进制转换函数将十进制转换为目标进制。

可以使用内置函数直接转换的有 :

list---->tuple      tuple(list)

tuple---->list      list(tuple)

## 6.5 Python2 与 Python3 的区别 ? (2018-3-30-lxy)

### 1) 核心类差异

#### 1. Python3 对 Unicode 字符的原生支持。

Python2 中使用 ASCII 码作为默认编码方式导致 string 有两种类型 str 和 unicode , Python3 只支持 unicode 的 string。Python2 和 Python3 字节和字符对应关系为 :

python2	python3	表现	转换	作用
str	bytes	字节	encode	存储
unicode	str	字符	decode	显示

#### 2. Python3 采用的是绝对路径的方式进行 import。



Python2 中相对路径的 import 会导致标准库导入变得困难（想象一下，同一目录下有 file.py，如何同时导入这个文件和标准库 file）。Python3 中这一点将被修改，如果还需要导入同一目录的文件必须使用绝对路径，否则只能使用相关导入的方式来进行导入。

3. Python2 中存在老式类和新式类的区别，Python3 统一采用新式类。新式类声明要求继承 object，必须用新式类应用多重继承。

4. Python3 使用更加严格的缩进。Python2 的缩进机制中，1 个 tab 和 8 个 space 是等价的，所以在缩进中可以同时允许 tab 和 space 在代码中共存。这种等价机制会导致部分 IDE 使用存在问题。Python3 中 1 个 tab 只能找另外一个 tab 替代，因此 tab 和 space 共存会导致报错：TabError: inconsistent use of tabs and spaces in indentation.

## 2) 废弃类差异

1. print 语句被 Python3 废弃，统一使用 print 函数

2. exec 语句被 python3 废弃，统一使用 exec 函数

3. execfile 语句被 Python3 废弃，推荐使用 `exec(open("./filename").read())`

4. 不相等操作符 "`<>`" 被 Python3 废弃，统一使用 "`!=`"

5. long 整数类型被 Python3 废弃，统一使用 int

6. xrange 函数被 Python3 废弃，统一使用 range，Python3 中 range 的机制也进行修改并提高了大数据集生成效率

7. Python3 中这些方法不再返回 list 对象：dictionary 关联的 keys()、values()、items()，zip()，map()，filter()，但是可以通过 list 强行转换：

```
1. mydict={"a":1,"b":2,"c":3}
2. mydict.keys() #<built-in method keys of dict object at 0x000000000040B4C8>
3. list(mydict.keys()) #['a', 'c', 'b']
```

8. 迭代器 iterator 的 next() 函数被 Python3 废弃，统一使用 next(iterator)

9. raw\_input 函数被 Python3 废弃，统一使用 input 函数
10. 字典变量的 has\_key 函数被 Python 废弃，统一使用 in 关键词
11. file 函数被 Python3 废弃，统一使用 open 来处理文件，可以通过 io.IOBase 检查文件类型
12. apply 函数被 Python3 废弃
13. 异常 StandardError 被 Python3 废弃，统一使用 Exception

### 3) 修改类差异

1. 浮点数除法操作符 "/" 和 "//" 的区别

" / " :

Python2：若为两个整形数进行运算，结果为整形，但若两个数中有一个为浮点数，则结果为浮点数；

Python3:为真除法，运算结果不再根据参加运算的数的类型。

"//" :

Python2：返回小于除法运算结果的最大整数；从类型上讲，与"/"运算符返回类型逻辑一致。

Python3：和 Python2 运算结果一样。

2. 异常抛出和捕捉机制区别

Python2

1. raise IOError, "file error" #抛出异常
2. except NameError, err: #捕捉异常

Python3

1. raise IOError("file error") #抛出异常
2. except NameError as err: #捕捉异常

3. for 循环中变量值区别

### Python2 , for 循环会修改外部相同名称变量的值

```
1. i = 1
2. print ( 'comprehension: ', [i for i in range(5)] )
3. print ( 'after: i =', i ) #i=4
```

### Python3 , for 循环不会修改外部相同名称变量的值

```
1. i = 1
2. print ( 'comprehension: ', [i for i in range(5)] )
3. print ( 'after: i =', i ) #i=1
```

### 4. round 函数返回值区别

#### Python2 , round 函数返回 float 类型值

```
1. isinstance(round(15.5),int) #True
```

#### Python3 , round 函数返回 int 类型值

```
1. isinstance(round(15.5),float) #True
```

### 5. 比较操作符区别

#### Python2 中任意两个对象都可以比较

```
1. 11 < 'test' #True
```

#### Python3 中只有同一数据类型的对象可以比较

```
1. 11 < 'test' # TypeError: unorderable types: int() < str()
```

#### 4) 第三方工具包差异

我们在 pip 官方下载源 **pypi** 搜索 Python2.7 和 Python3.5 的第三方工具包数可以发现 ,Python2.7 版本对应的第三方工具类目数量是 28523,Python3.5 版本的数量是 12457 , 这两个版本在第三方工具包支持数量差距相当大。

我们从数据分析的应用角度列举了常见实用的第三方工具包（如下表），并分析这些工具包在 Python2.7 和 Python3.5 的支持情况：

分类	工具名	用途
数据收集	scrapy	网页采集，爬虫
数据收集	scrapy-redis	分布式爬虫
数据收集	selenium	web 测试，仿真浏览器
数据处理	beautifulsoup	网页解释库，提供 lxml 的支持
数据处理	lxml	xml 解释库
数据处理	xlrd	excel 文件读取
数据处理	xlwt	excel 文件写入
数据处理	xlutils	excel 文件简单格式修改
数据处理	pywin32	excel 文件的读取写入及复杂格式定制
数据处理	Python-docx	Word 文件的读取写入
数据分析	numpy	基于矩阵的数学计算库
数据分析	pandas	基于表格的统计分析库

分类	工具名	用途
数据分析	scipy	科学计算库，支持高阶抽象和复杂模型
数据分析	statsmodels	统计建模和计量经济学工具包
数据分析	scikit-learn	机器学习工具库
数据分析	gensim	自然语言处理工具库
数据分析	jieba	中文分词工具库
数据存储	MySQL-python	mysql 的读写接口库
数据存储	mysqlclient	mysql 的读写接口库
数据存储	SQLAlchemy	数据库的 ORM 封装
数据存储	pymssql	sql server 读写接口库
数据存储	redis	redis 的读写接口
数据存储	PyMongo	mongodb 的读写接口
数据呈现	matplotlib	流行的数据可视化库
数据呈现	seaborn	美观的数据可视化库，基于 matplotlib
工具辅助	jupyter	基于 web 的 python IDE，常用于数据分析
工具辅助	chardet	字符检查工具
工具辅助	ConfigParser	配置文件读写支持
工具辅助	requests	HTTP 库，用于网络访问

## 5) 工具安装问题

windows 环境

Python2 无法安装 mysqlclient。Python3 无法安装 MySQL-python、flup、functools32、Gooey、Pywin32、webencodings。

matplotlib 在 python3 环境中安装报错：The following required packages can not be built: freetype, png。需要手动下载安装源码包安装解决。

scipy 在 Python3 环境中安装报错，numpy.distutils.system\_info.NotFoundError，需要自己手工下载对应的安装包，依赖 numpy, pandas 必须严格根据 python 版本、操作系统、64 位与否。运行 matplotlib 后发现基础包 numpy+mkl 安装失败，需要自己下载，国内暂无下载源

centos 环境下

Python2 无法安装 mysql-python 和 mysqlclient 包 报错 :EnvironmentError: mysql\_config not found，解决方案是安装 mysql-devel 包解决。使用 matplotlib 报错：no module named \_tkinter，安装 Tkinter、tk-devel、tc-devel 解决。

pywin32 也无法在 centos 环境下安装。

## 6.6 关于 Python 程序的运行方面，有什么手段能提升性能？(2018-3-30-lxy)

- 1、使用多进程，充分利用机器的多核性能
- 2、对于性能影响较大的部分代码，可以使用 C 或 C++ 编写
- 3、对于 IO 阻塞造成的性能影响，可以使用 IO 多路复用来解决
- 4、尽量使用 Python 的内建函数
- 5、尽量使用局部变量

## 6.7 Python 中的作用域 ? (2018-4-16-lxy)

Python 中，一个变量的作用域总是由在代码中被赋值的地方所决定。当 Python 遇到一个变量的话它会按照这的顺序进行搜索：

本地作用域(Local)--->当前作用域被嵌入的本地作用域(Enclosing locals)--->全局/模块作用域(Global)--->内置作用域(Built-in)。

## 6.8 什么是 Python ? (2018-4-16-lxy)

- Python 是一种编程语言，它有对象、模块、线程、异常处理和自动内存管理，可以加入其他语言的对比。
- Python 是一种解释型语言，Python 在代码运行之前不需要解释。
- Python 是动态类型语言，在声明变量时，不需要说明变量的类型。
- Python 适合面向对象的编程，因为它支持通过组合与继承的方式定义类。
- 在 Python 语言中，函数是第一类对象。
- Python 代码编写快，但是运行速度比编译型语言通常要慢。
- Python 用途广泛，常被用走"胶水语言"，可帮助其他语言和组件改善运行状况。
- 使用 Python，程序员可以专注于算法和数据结构的设计，而不用处理底层的细节。

## 6.9 什么是 Python 自省 ? (2018-4-16-lxy)

Python 自省是 Python 具有的一种能力，使程序员面向对象的语言所写的程序在运行时,能够获得对象的类 Python 型。Python 是一种解释型语言，为程序员提供了极大的灵活性和控制力。

更多知识，扫描如下二维码。



## 6.10 什么是 Python 的命名空间？(2018-4-16-lxy)

在 Python 中，所有的名字都存在于一个空间中，它们在该空间中存在和被操作——这就是命名空间。它就好像一个盒子，每一个变量名字都对应装着一个对象。当查询变量的时候，会从该盒子里面寻找相应的对象。

## 6.11 你所遵循的代码规范是什么？请举例说明其要求？(2018-4-20-lxy)

PEP8 规范。

### 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 huafu'}
```

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。

## 7. Linux 基础和 git

### 7.1 Linux 的基本命令（怎么区分一个文件还是文件夹）(2018-3-30-lxy)

ls -F 在显示名称的时候会在文件夹后添加 "/" ，在文件后面加 "\*" 。

### 7.2 日志以什么格式，存放在哪里？(2018-3-30-lxy)

日志以文本可以存储在 "/var/log/" 目录下后缀名为.log。

### 7.3 Linux 查看某个服务的端口？(2018-3-30-lxy)

```
1. netstat -anp | grep service_name
```

### 7.4 ubuntu 系统如何设置开机自启动一个程序？(2018-3-30-lxy)

直接修改/etc/rc0.d ~ /etc/rc6.d 和/etc/rcS.d 文件夹的内容，添加需启动的程序，S 开头的表示启动，K 开头的表示不启动。

### 7.5 在 linux 中 find 和 grep 的区别(2018-4-16-lxy)

Linux 系统中 grep 命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。grep 全称是 Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

linux 下的 find：

功能：在目录结构中搜索文件，并执行指定的操作。此命令提供了相当多的查找条件，功能很强大。

语法：find 起始目录寻找条件操作

说明：find 命令从指定的起始目录开始，递归地搜索其各个子目录，查找满足寻找条件的文件并对之采取相关的操作。

简单点说说，grep 是查找匹配条件的行，find 是搜索匹配条件的文件。

## 7.6 Linux 重定向命令有哪些？有什么区别？(2018-4-16-lxy)

### 1、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)。

### 2、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。

## 7.7 软连接和硬链接的区别？(2018-4-16-lxy)

软连接类似 Windows 的快捷方式，当删除源文件时，那么软链接也失效了。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，文件被删除。

## 7.8 10 个常用的 Linux 命令？(2018-4-20-lxy)

`pwd` 显示工作路径

`ls` 查看目录中的文件

`cd /home` 进入 '/home' 目录'

`cd ..` 返回上一级目录

`cd ../..` 返回上两级目录

`mkdir dir1` 创建一个叫做 'dir1' 的目录'

`rm -f file1` 删除一个叫做 'file1' 的文件', -f 参数, 忽略不存在的文件, 从不给出提示。

`rmdir dir1` 删除一个叫做 'dir1' 的目录'

`groupadd group_name` 创建一个新用户组

`groupdel group_name` 删除一个用户组

`tar -cvf archive.tar file1` 创建一个非压缩的 tarball

`tar -cvf archive.tar file1 file2 dir1` 创建一个包含了 'file1', 'file2' 以及 'dir1'的档案文件

`tar -tf archive.tar` 显示一个包中的内容

`tar -xvf archive.tar` 释放一个包

`tar -xvf archive.tar -C /tmp` 将压缩包释放到 /tmp 目录下

`tar -cvfj archive.tar.bz2 dir1` 创建一个 bzip2 格式的压缩包

`tar -xvfj archive.tar.bz2` 解压一个 bzip2 格式的压缩包

`tar -cvfz archive.tar.gz dir1` 创建一个 gzip 格式的压缩包

`tar -xvfz archive.tar.gz` 解压一个 gzip 格式的压缩包



## 7.9 Linux 关机命令有哪些？(2018-4-20-lxy)

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

## 7.10 git 合并文件有冲突，如何处理？(2018-3-30-lxy)

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  
直接 `push` 就可以了，因为刚刚已经执行过相关 `merge` 操作了。

## 二．数据类型

### 1. 字典

**dict:**字典，字典是一组键(key)和值(value)的组合，通过键(key)进行查找，没有顺序，使用大括号“{}”；

应用场景：

**dict**，使用键和值进行关联的数据；

1.1 现有字典 `d={'a':24, 'g':52, 'i':12, 'k':33}`请按字典中的 `value` 值进行排序？(2018-3-30-lxy)

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

1.2 说一下字典和json的区别？(2018-3-30-lxy)

字典是一种数据结构，json 是一种数据的表现形式，字典的 `key` 值只要是能 `hash` 的就行，json 的必须是字符串。

1.3 什么是可变、不可变类型？(2018-3-30-lxy)

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。

## 1.4 存入字典里的数据有没有先后排序？(2018-3-30-lxy)

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。

## 1.5 字典推导式？(2018-4-23-lxy)

```
1. d = {key: value for (key, value) in iterable}
```

1.6 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？ (2018-4-23-lxy)

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

## 2. 字符串

str:字符串是 Python 中最常用的数据类型。我们可以使用引号('或")来创建字符串。

### 2.1 如何理解 Python 中字符串中的\字符？(2018-3-30-lxy)

有三种不同的含义：

1、转义字符 2、路径名中用来连接路径名 3、编写太长代码手动软换行。

### 2.2 请反转字符串 "aStr" ?(2018-3-30-lxy)

```
1. print( 'aStr'[::-1])
```

2.3 将字符串"k:1|k1:2|k2:3|k3:4"，处理成 Python 字典：{k:1， k1:2， ...} # 字典里的 K 作为字符串处理(2018-3-30-lxy)

```
1. str1 = "k:1|k1:2|k2:3|k3:4"
2. def str2dict(str1):
3.     dict1 = {}
```



```
4.     for items in str1.split('|'):
5.         key, value = items.split(':')
6.         dict1[key] = value
7.     return dict1
```

## 2.4 请按 alist 中元素的 age 由大到小排序(2018-3-30-lxy)

```
1. alist [{'name':'a', 'age':20}, {'name':'b', 'age':30}, {'name':'c', 'age':25}]
2. def sort_by_age(list1):
3.     return sorted(alist, key=lambda x:x['age'], reverse=True)
```

## 3. 列表

list:是 Python 中使用最频繁的数据类型，在其他语言中通常叫做数组，通过索引进行查找，使用方括号“[]”，列表是有序的集合。

应用场景：定义列表使用 [] 定义，数据之间使用 “,” 分割。

列表的索引从 0 开始：索引就是数据在列表中的位置编号，索引又可以被称为下标。

【注意】：从列表中取值时,如果超出索引范围,程序会产生异常。

IndexError: list index out of range

列表的常用操作：

```
1. name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
```

### 1) 增加

列表名.insert(index, 数据)：在指定位置插入数据(位置前有空元素会补位)。

```
1. # 往列表 name_list 下标为 0 的地方插入数据
2. In [3]: name_list.insert(0, "Sasuke")
3. In [4]: name_list
4. Out[4]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu']
5. # 现有的列表下标是 0-4，如果我们要在下标是 6 的地方插入数据，那个会自动插入到下标为 5 的地方，也就是# 插入到最后
6. In [5]: name_list.insert(6, "Tom")
7. In [6]: name_list
8. Out[6]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu', 'Tom']
```

列表名.append(数据)：在列表的末尾追加数据(最常用的方法)。

```
1. In [7]: name_list.append("Python")
```

```
2 . In [8]: name_list
3 . Out[8]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu', 'Tom', 'Python']
```

列表.extend(Iterable)：将可迭代对象中的元素追加到列表。

```
1 . # 有两个列表 a 和 b a.extend(b) 会将 b 中的元素追加到列表 a 中
2 . In [10]: a = [11, 22, 33]
3 . In [11]: b = [44, 55, 66]
4 . In [12]: a.extend(b)
5 . In [13]: a
6 . Out[13]: [11, 22, 33, 44, 55, 66]# 有列表 c 和 字符串 d c.extend(d) 会将字符串 d 中的每个字符拆开作为元素插入到列表
c
7 . In [14]: c = ['j', 'a', 'v', 'a']
8 . In [15]: d = "python"
9 . In [16]: c.extend(d)
10 . In [17]: c
11 . Out[17]: ['j', 'a', 'v', 'a', 'p', 'y', 't', 'h', 'o', 'n']
```

## 2) 取值和修改

取值：列表名[index]：根据下标来取值。

```
1 . In [19]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
2 . In [20]: name_list[0]
3 . Out[20]: 'zhangsan'
4 . In [21]: name_list[3]
5 . Out[21]: 'zhaoliu'
```

修改：列表名[index] = 数据：修改指定索引的数据。

```
1 . In [22]: name_list[0] = "Sasuke"
2 . In [23]: name_list
3 . Out[23]: ['Sasuke', 'lisi', 'wangwu', 'zhaoliu']
```

## 3) 删除

del 列表名[index]：删除指定索引的数据。

```
1 . In [25]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]# 删除索引是 1 的数据
2 . In [26]: del name_list[1]
3 . In [27]: name_list
4 . Out[27]: ['zhangsan', 'wangwu', 'zhaoliu']
```

列表名.remove(数据)：删除第一个出现的指定数据。

```
5 . In [30]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu", "lisi"]# 删除 第一次出现的 lisi 的数据
6 . In [31]: name_list.remove("lisi")
7 . In [32]: name_list
8 . Out[32]: ['zhangsan', 'wangwu', 'zhaoliu', 'lisi']
```

列表名.pop()：删除末尾的数据,返回值: 返回被删除的元素。

```
9 . In [33]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]# 删除最后一个元素 zhaoliu 并将元素 zhaoliu 返回
10 . In [34]: name_list.pop()
11 . Out[34]: 'zhaoliu'
12 . In [35]: name_list
13 . Out[35]: ['zhangsan', 'lisi', 'wangwu']
```

列表名.pop(index)：删除指定索引的数据，返回被删除的元素。

```
14 . In [36]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]# 删除索引为 1 的数据 lisi
15 . In [37]: name_list.pop(1)
16 . Out[37]: 'lisi'
17 . In [38]: name_list
18 . Out[38]: ['zhangsan', 'wangwu', 'zhaoliu']
```

列表名.clear()：清空整个列表的元素。

```
19 . In [40]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
20 . In [41]: name_list.clear()
21 . In [42]: name_list
22 . Out[42]: []
```

#### 4) 排序

列表名.sort()：升序排序 从小到大。

```
23 . In [43]: a = [33, 44, 22, 66, 11]
24 . In [44]: a.sort()
25 . In [45]: a
26 . Out[45]: [11, 22, 33, 44, 66]
```

列表名.sort(reverse=True)：降序排序 从大到小。

```
27 . In [46]: a = [33, 44, 22, 66, 11]
28 . In [47]: a.sort(reverse=True)
29 . In [48]: a
30 . Out[48]: [66, 44, 33, 22, 11]
```

列表名.reverse()：列表逆序、反转。

```
31 . In [50]: a = [11, 22, 33, 44, 55]
32 . In [51]: a.reverse()
33 . In [52]: a
34 . Out[52]: [55, 44, 33, 22, 11]
```

#### 5) 统计相关

len(列表名)：得到列表的长度。

```
35 . In [53]: a = [11, 22, 33, 44, 55]
36 . In [54]: len(a)
37 . Out[54]: 5
```

列表名.count(数据)：数据在列表中出现的次数。

```
38 . In [56]: a = [11, 22, 11, 33, 11]
39 . In [57]: a.count(11)
40 . Out[57]: 3
```

列表名.index(数据)：数据在列表中首次出现时的索引，没有查到会报错。

```
41 . In [59]: a = [11, 22, 33, 44, 22]
42 . In [60]: a.index(22)
43 . Out[60]: 1
```

if 数据 in 列表：判断列表中是否包含某元素。

```
44 . a = [11, 22, 33, 44, 55]
45 . if 33 in a:
46 .     print("找到了....")
```

## 6) 循环遍历

使用 while 循环：

```
1. a = [11, 22, 33, 44, 55]
2.
3. i = 0
4. while i < len(a):
5.     print(a[i])
6.     i += 1
```

使用 for 循环：

```
1. a = [11, 22, 33, 44, 55]
2. for i in a:
3.     print(i)
```

### 3.1 下面代码的输出结果将是什么？(2018-3-30-lxy)

```
1. list = ['a', 'b', 'c', 'd', 'e']
2. print list[10:]
```

下面的代码将输出[] 不会产生IndexError 错误。就像所期望的那样，尝试用超出成员的个数的index 来获取某个列表的成员。

例如，尝试获取 list[10]和之后的成员，会导致IndexError。

然而，尝试获取列表的切片，开始的 index 超过了成员个数不会产生 `IndexError`，而是仅仅返回一个空列表。这成为特别让人恶心的疑难杂症，因为运行的时候没有错误产生，导致 bug 很难被追踪到。

### 3.2 写一个列表生成式，产生一个公差为 11 的等差数列(2018-3-30-lxy)

```
1. print([x*11 for x in range(10)])
```

### 3.3 给定两个列表，怎么找出他们相同的元素和不同的元素? (2018-3-30-lxy)

```
1. list1 = [1, 2, 3]
2. list2 = [3, 4, 5]
3. set1 = set(list1)
4. set2 = set(list2)
5. print(set1&set2)
6. print(set1^set2)
```

### 3.4 请写出一段 Python 代码实现删除一个 list 里面的重复元素?(2018-3-30-lxy)

比较容易记忆的是用内置的 `set`：

```
1. l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a']
2. l2 = list(set(l1))
3. print l2
```

如果保持他们原来的排序：

用 `list` 类的 `sort` 方法：

```
1. l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a']
2. l2 = list(set(l1))
3. l2.sort(key=l1.index)
4. print l2
```

也可以这样写：

```
1. l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a']
2. l2 = sorted(set(l1), key=l1.index)
3. print l2
```

也可以用遍历：

```
1. l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a']
2. l2 = []
```

```
3. for i in l1:
4.     if not i in l2:
5.         l2.append(i)
6. print l2
```

3.5 给定两个 list A ,B , 请用找出 A ,B 中相同的元素 , A ,B 中不同的元素  
(2018-3-30-lxy)

A、 B 中相同元素 : print(set(A)&set(B))

A、 B 中不同元素 : print(set(A)^set(B))

3.6 有如下数组 list = range(10)我想取以下几个数组 , 应该如何切片 ?  
(2018-3-30-lxy)

```
1. [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9]
2. [1 , 2 , 3 , 4 , 5 , 6]
3. [3 , 4 , 5 , 6]
4. [9]
5. [1 , 3 , 5 , 7 , 9]
```

答 :

```
1. [1:]
2. [1:7]
3. [3:7]
4. [-1]
5. [1::2]
```

3.7 下面这段代码的输出结果是什么 ? 请解释 ? (2018-4-16-lxy)

```
1. def extendlist(val, list=[]):
2.     list.append(val)
3.     return list
4.
5. list1 = extendlist(10)
6. list2 = extendlist(123, [])
7. list3 = extendlist('a')
8.
9. print("list1 = %s" %list1)
```

```
10. print("list2 = %s" %list2)
11. print("list3 = %s" %list3)
```

输出结果：

```
12. list1 = [10, 'a']
13. list2 = [123]
14. list3 = [10, 'a']
```

新的默认列表只在函数被定义的那一刻创建一次。当 extendList 被没有指定特定参数 list 调用时，这组 list 的值随后将被使用。这是因为带有默认参数的表达式在函数被定义的时候被计算，不是在调用的时候被计算。

### 3.8.将以下 3 个函数按照执行效率高低排序(2018-4-16-lxy)

```
1. def f1(lIn):
2.     l1 = sorted(lIn)
3.     l2 = [i for i in l1 if i<0.5]
4.     return [i*i for i in l2]
5.
6.
7. def f2(lIn):
8.     l1 = [i for i in l1 if i<0.5]
9.     l2 = sorted(l1)
10.    return [i*i for i in l2]
11.
12.
13. def f3(lIn):
14.     l1 = [i*i for i in lIn]
15.     l2 = sorted(l1)
16.     return [i for i in l1 if i<(0.5*0.5)]
```

按执行效率从高到低排列 :f2、f1 和 f3。要证明这个答案是正确的，你应该知道如何分析自己代码的性能。Python 中有一个很好的程序分析包，可以满足这个需求。

```
1. import random
2. import cProfile
3. lIn = [random.random() for i in range(100000)]
4. cProfile.run('f1(lIn)')
5. cProfile.run('f2(lIn)')
6. cProfile.run('f3(lIn)')
```

### 3.9 获取 1~100 被 6 整除的偶数 ? (2018-4-23-lxy)

```
1. def A():
2.     alist = []
3.     for i in range(1, 100):
4.         if i % 6 == 0:
5.             alist.append(i)
6.     last_num = alist[-3:]
7.     print(last_num)
```

## 4. 元祖

tuple:元组，元组将多样的对象集合到一起，不能修改，通过索引进行查找，使用括号“ ()”；

应用场景：把一些数据当做一个整体去使用，不能修改；

## 5. 集合

set:set 集合，在 Python 中的书写方式的{}，集合与之前列表、元组类似，可以存储多个数据，但是这些数据是不重复的。集合对象还支持 union(联合), intersection(交), difference(差)和 symmmetric\_difference(对称差集)等数学运算。

快速去除列表中的重复元素

```
1 . In [4]: a = [11,22,33,33,44,22,55]
2 .
3 . In [5]: set(a)
4 . Out[5]: {11, 22, 33, 44, 55}
```

交集：共有的部分

```
1 . In [7]: a = {11,22,33,44,55}
2 . In [8]: b = {22,44,55,66,77}
3 . In [9]: a&b
4 . Out[9]: {22, 44, 55}
```

并集：总共的部分

```
1 . In [11]: a = {11,22,33,44,55}
2 . In [12]: b = {22,44,55,66,77}
3 . In [13]: a | b
```



```
4 . Out[13]: {11, 22, 33, 44, 55, 66, 77}
```

差集：另一个集合中没有的部分

```
1 . In [15]: a = {11,22,33,44,55}
2 . In [16]: b = {22,44,55,66,77}
3 . In [17]: b - a
4 . Out[17]: {66, 77}
```

对称差集(在 a 或 b 中，但不会同时出现在二者中)

```
1 . In [19]: a = {11,22,33,44,55}
2 . In [20]: b = {22,44,55,66,77}
3 . In [21]: a ^ b
4 . Out[21]: {11, 33, 66, 77}
```

## 第三章 Python 高级

### 一 . 元类

#### 1.Python 中类方法、类实例方法、静态方法有何区别？(2018-3-30-lxy)

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。

#### 2.Python 中如何动态获取和设置对象的属性？(2018-3-30-lxy)

```
1. if hasattr(Parent, 'x'):
2.     print(getattr(Parent, 'x'))
3.     setattr(Parent, 'x', 3)
4. print(getattr(Parent, 'x'))
```

## 二．内存管理与垃圾回收机制

### 1. Python 的内存管理机制及调优手段？(2018-3-30-lxy)

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）

## 2.内存泄露是什么？如何避免？(2018-3-30-lxy)

指由于疏忽或错误造成程序未能释放已经不再使用的内存的情况。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，失去了对该段内存的控制，因而造成了内存的浪费。导致程序运行速度减慢甚至系统崩溃等严重后果。

有 `__del__()` 函数的对象间的循环引用是导致内存泄漏的主凶。

不使用一个对象时使用`del object` 来删除一个对象的引用计数就可以有效防止内存泄漏问题。

通过 Python 扩展模块 `gc` 来查看不能回收的对象的详细信息。

可以通过 `sys.getrefcount(obj)` 来获取对象的引用计数，并根据返回值是否为 0 来判断是否内存泄漏。

## 三．函数

### 1. 函数参数

#### 1.1 Python 函数调用的时候参数的传递方式是值传递还是引用传递？

(2018-3-30-lxy)

Python 的参数传递有：位置参数、默认参数、可变参数、关键字参数。

函数的传值到底是值传递还是引用传递，要分情况：

不可变参数用值传递：

像整数和字符串这样的不可变对象，是通过拷贝进行传递的，因为你无论如何都不可能在原处改变不可变对象

可变参数是引用传递的：

比如像列表，字典这样的对象是通过引用传递、和 C 语言里面的用指针传递数组很相似，可变对象能在函数内部改变。

## 1.2 对缺省参数的理解 ? (2018-3-30-lxy)

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。

\*args 是不定长参数，他可以表示输入参数是不确定的，可以是任意多个。

\*\*kwargs 是关键字参数，赋值的时候是以键 = 值的方式，参数是可以任意多对在定义函数的时候不确定会有多少参数会传入时，就可以使用两个参数。

## 1.3 为什么函数名字可以当做参数用?(2018-3-30-lxy)

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。

## 1.4 Python 中 pass 语句的作用是什么 ? (2018-3-30-lxy)

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。

## 1.5 有这样一段代码，print c 会输出什么，为什么 ? (2018-3-30-lxy)

```
1. a = 10
2. b = 20
3. c = [a]
```

```
4. a = 15
```

答：10 对于字符串、数字，传递是相应的值。

## 1.6 交换两个变量的值？(2018-4-16-lxy)

```
1. a,b = b,a
```

## 2. 内建函数

### 2.1 map 函数和 reduce 函数？(2018-3-30-lxy)

①从参数方面来讲：

map()包含两个参数，第一个参数是一个函数，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。

reduce()是将传入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）。

### 2.2 递归函数停止的条件？(2018-3-30-lxy)

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

## 2.3 回调函数，如何通信的? (2018-3-30-lxy)

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。

## 2.4 Python 主要的内置数据类型都有哪些？ print dir( 'a ' ) 的输出？ (2018-3-30-lxy)

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；

## 2.5 print(list(map(lambda x: x \* x, [y for y in range(3)])))的输出？ (2018-3-30-lxy)

```
1. [0, 1, 4]
```

## 2.6 hasattr() getattr() setattr() 函数使用详解？ (2018-4-16-lxy)

hasattr(object, name)函数：

判断一个对象里面是否有 name 属性或者 name 方法 返回 bool 值 有 name 属性(方法)返回 True，否则返回 False。

注意：name 要使用引号括起来。

```
1. class function_demo(object):
2.     name = 'demo'
3.     def run(self):
4.         return "hello function"
5. functiondemo = function_demo()
6. res = hasattr(functiondemo, 'name') #判断对象是否有 name 属性，True
7. res = hasattr(functiondemo, "run") #判断对象是否有 run 方法，True
8. res = hasattr(functiondemo, "age") #判断对象是否有 age 属性，False
9. print(res)
```

getattr(object, name[,default]) 函数：

获取对象 object 的属性或者方法，如果存在则打印出来，如果不存在，打印默认值，默认值可选。

注意：如果返回的是对象的方法，则打印结果是：方法的内存地址，如果需要运行这个方法，可以在后面添加括号()。

```
1. functiondemo = function_demo()
2. getattr(functiondemo, 'name') #获取 name 属性，存在就打印出来--- demo
3. getattr(functiondemo, "run") #获取 run 方法，存在打印出 方法的内存地址---<bound method function_demo.run of
<__main__.function_demo object at 0x10244f320>>
4. getattr(functiondemo, "age") #获取不存在的属性，报错如下：
5. Traceback (most recent call last):
6.   File "/Users/liuhuiling/Desktop/MT_code/OpAPIDemo/conf/OPCommUtil.py", line 39, in <module>
7.     res = getattr(functiondemo, "age")
8. AttributeError: 'function_demo' object has no attribute 'age'
9. getattr(functiondemo, "age", 18) #获取不存在的属性，返回一个默认值
```

setattr(object,name,values)函数：

给对象的属性赋值，若属性不存在，先创建再赋值

```
1. class function_demo(object):
2.     name = 'demo'
3.     def run(self):
4.         return "hello function"
5. functiondemo = function_demo()
6. res = hasattr(functiondemo, 'age') # 判断 age 属性是否存在，False
7. print(res)
8. setattr(functiondemo, 'age', 18) #对 age 属性进行赋值，无返回值
9. res1 = hasattr(functiondemo, 'age') #再次判断属性是否存在，True
```

综合使用：

```
1. class function_demo(object):
2.     name = 'demo'
3.     def run(self):
4.         return "hello function"
5. functiondemo = function_demo()
6. res = hasattr(functiondemo, 'addr') # 先判断是否存在 if res:
7.     addr = getattr(functiondemo, 'addr')
8.     print(addr)else:
9.     addr = getattr(functiondemo, 'addr', setattr(functiondemo, 'addr', '北京首都'))
10.    #addr = getattr(functiondemo, 'addr', '美国纽约')
11.    print(addr)
```



## 2.7 一句话解决阶乘函数？(2018-4-16-lxy)

在 Python2 中：

```
1. reduce(lambda x,y: x*y, range(1,n+1))
```

注意：Python3 中取消了该函数。

## 3.Lambda

### 3.1 什么是 lambda 函数？有什么好处？(2018-4-16-lxy)

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数

- 1、lambda 函数比较轻便，即用即仍，很适合需要完成一项功能，但是此功能只在此一处使用，连名字都很随意的情况下；
- 2、匿名函数，一般用来给 filter，map 这样的函数式编程服务；
- 3、作为回调函数，传递给某些应用，比如消息处理

### 3.2 下面这段代码的输出结果将是什么？请解释。(2018-3-30-lxy)

```
1. def multipliers():  
2.     return [lambda x: i * x for i in range(4)]  
3.     print [m(2) for m in multipliers()]
```

上面代码输出的结果是[6， 6， 6， 6] (不是我们想的[0， 2， 4， 6])。

你如何修改上面的 multipliers 的定义产生想要的结果？

上述问题产生的原因是 Python 闭包的延迟绑定。这意味着内部函数被调用时，参数的值在闭包内进行查找。因此，当任何由 multipliers()返回的函数被调用时，i 的值将在附近的范围进行查找。那时，不管返回的函数是否被调用，for 循环已经完成，i 被赋予了最终的值 3。

因此，每次返回的函数乘以传递过来的值 3，因为上段代码传过来的值是 2，它们最终返回的都是 6。  
(3\*2)碰巧的是，《The Hitchhiker's Guide to Python》也指出，在与 lambdas 函数相关也有一个被广泛被误解的知识点，不过跟这个 case 不一样。由 lambda 表达式创造的函数没有什么特殊的地方，它其实是和 def 创造的函数式一样的。

下面是解决这一问题的一些方法。

一种解决方法就是用 Python 生成器。

```
1. def multipliers():  
2.     for i in range(4): yield lambda x: i * x
```

另外一个解决方案就是创建一个闭包，利用默认函数立即绑定。

```
1. def multipliers():  
2.     return [lambda x, i=i: i * x for i in range(4)]
```

### 3.3 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？

(2018-3-30-lxy)

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；

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

## 四．设计模式

### 1. 单例

#### 1.1 请手写一个单例(2018-3-30-lxy)

```
1. class A(object):  
2.     __instance = None  
3.     def __new__(cls, *args, **kwargs):  
4.         if cls.__instance is None:  
5.             cls.__instance = object.__new__(cls)
```

```
6.         return cls.__instance
7.     else:
8.         return cls.__instance
```

## 1.2 单例模式的应用场景有哪些？(2018-4-16-lxy)

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....

## 2. 工厂

## 3. 装饰器

### 3.1 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？

(2018-3-30-lxy)

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。

```
1. import time
2. def timeit(func):
3.     def wrapper():
4.         start = time.clock()
5.         func() end =time.clock()
6.         print 'used:', end - start
7.         return wrapper
8. @timeit
9. def foo():
10.     print 'in foo()'foo()
```

### 3.2 解释一下什么是闭包?(2018-3-30-lxy)

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。

### 3.3 函数装饰器有什么作用？ (2018-4-16-lxy)

装饰器本质上是一个 Python 函数，它可以在让其他函数在不需要做任何代码的变动的前提下增加额外的功能。装饰器的返回值也是一个函数的对象，它经常用于有切面需求的场景。比如：插入日志、性能测试、事务处理、缓存、权限的校验等场景。有了装饰器就可以抽离出大量的与函数功能本身无关的雷同代码并继续使用。

## 4. 生成器

### 4.1 生成器、迭代器的区别？(2018-3-30-lxy)

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。

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，当生成器终结时，还会自动抛出 `StopIteration` 异常。

## 4.2 X 是什么类型 ? (2018-3-30-lxy)

```
1. X = (for i in range(10))
```

答：X 是 generator 类型。

## 4.3 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? (2018-4-20-lxy)

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

## 4.4 Python 中 yield 的用法 ? (2018-4-16-lxy)

yield 就是保存当前程序执行状态。你用 for 循环的时候，每次取一个元素的时候就会计算一次。用 yield 的函数叫 generator 和 iterator 一样，它的好处是不用一次计算所有元素，而是用一次算一次，可以节省很多空间。generator 每次计算需要上一次计算结果，所以用 yield，否则一 return，上次计算结果就没了。

```
1. >>> def createGenerator():
2. ...     mylist = range(3)
3. ...     for i in mylist:
4. ...         yield i*i
5. ...
6. >>> mygenerator = createGenerator() # create a generator
7. >>> print(mygenerator) # mygenerator is an object!
8. <generator object createGenerator at 0xb7555c34>
9. >>> for i in mygenerator:
10. ...     print(i)
11. 0
12. 1
13. 4
```

## 五．面向对象

### 1. 类

### 2. 对象

#### 2.1 Python 中的可变对象和不可变对象？(2018-3-30-lxy)

不可变对象，该对象所指向的内存中的值不能被改变。当改变某个变量时候，由于其所指的值不能被改变，相当于把原来的值复制一份后再改变，这会开辟一个新的地址，变量再指向这个新的地址。

可变对象，该对象所指向的内存中的值可以被改变。变量（准确的说是引用）改变后，实际上是其所指的值直接发生改变，并没有发生复制行为，也没有开辟新的地址，通俗点说就是原地改变。

Python 中，数值类型（int 和 float）、字符串 str、元组 tuple 都是不可变类型。而列表 list、字典 dict、集合 set 是可变类型。

#### 2.2 Python 中 is 和 == 的区别？(2018-3-30-lxy)

is 判断的是 a 对象是否就是 b 对象，是通过 id 来判断的。

== 判断的是 a 对象的值是否和 b 对象的值相等，是通过 value 来判断的。

#### 2.3 Python 的魔法方法 (2018-3-30-lxy)

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。

## 2.4 面向对象中怎么实现只读属性? (2018-3-30-lxy)

将对象私有化，通过共有方法提供一个读取数据的接口。

```
1. class person:
2.     def __init__(self,x):
3.         self.__age = 10;
4.     def age(self):
5.         return self.__age;
6. t = person(22)
7. # t.__age = 100
8. print(t.age())
```

### 最好的方法

```
9. class MyClass(object):
10.     __weight = 50
11.
12.     @property #以访问属性的方式来访问 weight 方法
13.     def weight(self):
14.         return self.__weight
15.
16. if __name__ == '__main__':
17.     obj = MyClass()
18.     print(obj.weight)
19.     obj.weight = 12
```

```
1 . Traceback (most recent call last):
2 . 50
3 . File "C:/PythonTest/test.py", line 11, in <module>
4 .     obj.weight = 12
5 . AttributeError: can't set attribute
```

## 2.5 谈谈你对面向对象的理解？

面向对象是相对于面向过程而言的。面向过程语言是一种基于功能分析的、以算法为中心的程序设计方法；而面向对象是一种基于结构分析的、以数据为中心的程序设计思想。在面向对象语言中有一个有很重要东西，叫做类。

面向对象有三大特性：封装、继承、多态。

## 六．正则表达式

### 1. Python 里 match 与 search 的区别？(2018-3-30-lxy)

match()函数只检测 RE 是不是在 string 的开始位置匹配，

search()会扫描整个 string 查找匹配；

也就是说 match()只有在 0 位置匹配成功的话才有返回，

如果不是开始位置匹配成功的话，match()就返回 none。

### 2. Python 字符串查找和替换？(2018-3-30-lxy)

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



## 3. 用 Python 匹配 HTML g tag 的时候，&lt;.\*&gt; 和 &lt;.\*?&gt; 有什么区别？

(2018-3-30-lxy)

<.\*>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 "<>"。

## 4.请写出下列正则关键字的含义？(2018-3-30-lxy)

语法	说明	表达式实例	完整匹配的字符串
字符			
一般字符	匹配自身	abc	abc
.	匹配任意除换行符"\n"外的字符。 在 DOTALL 模式中也能匹配换行符。	a.c	abc
\	转义字符,使后一个字符改变原来的意思。 如果字符串中有字符*需要匹配,可以使用\*或者字符集[*]	a\.c a\\c	a. c a\c
[...]	字符集(字符类)。对应的位置可以是字符集中任意字符。字符集中的字符可以逐个列出,也可以给出范围,如[abc]或[a-c]。第一个字符如果是^则表示取反,如[^abc]表示不是 abc 的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用],-或^,可以在前面加上反斜杠,或把],-放在第一个字符,把^放在非第一个字符。	a[bcd]e	abe ace ade
预定义字符集(可以写在字符集[...]中)			
\d	数字:[0-9]	a\dc	a1c
\D	非数字:[^d]	a\Dc	abc

\s	空白字符:[<空格>\t\r\n\f\v]	a\sc	ac
\S	非空白字符:[^\s]	a\Sc	abc
\w	单词字符:[A-Za-z0-9_]	a\wc	abc
\W	非单词字符:[^\W]	a\Wc	ac
数量词(用在字符或(···)之后)			
*	匹配前一个字符 0 或无限次。	abc*	ab abccc
+	匹配前一个字符 1 次或无限次	abc+	abc abccc
?	匹配前一个字符 0 次或 1 次。	abc?	Ab abc
{m}	匹配前一个字符 m 次	ab{2}c	abbc

## 七．系统编程

### 1. 进程总结

进程：程序运行在操作系统上的一个实例，就称之为进程。进程需要相应的系统资源：内存、时间片、pid。

创建进程：

1.首先要导入 multiprocessing 中的 Process；

2.创建一个 Process 对象；

3.创建 Process 对象时，可以传递参数；

```
1 . p = Process(target=XXX, args=(元组,) , kwargs={key:value})
2 . target = XXX 指定的任务函数,不用加()
3 . args=(元组,) , kwargs={key:value} 给任务函数传递的参数
```

4.使用 start()启动进程；

5.结束进程。

Process 语法结构：

```
Process([group [, target [, name [, args [, kwargs]]]])
```

target : 如果传递了函数的引用，可以让这个子进程就执行函数中的代码

args : 给 target 指定的函数传递的参数，以元组的形式进行传递

kwargs : 给 target 指定的函数传递参数，以字典的形式进行传递

name : 给进程设定一个名字，可以省略

group : 指定进程组，大多数情况下用不到

Process 创建的实例对象的常用方法有：

start() : 启动子进程实例(创建子进程)

is\_alive() : 判断进程子进程是否还在活着

join(timeout) : 是否等待子进程执行结束，或者等待多少秒

terminate() : 不管任务是否完成，立即终止子进程

Process 创建的实例对象的常用属性：

name : 当前进程的别名，默认为 Process-N,N 为从 1 开始递增的整数

pid : 当前进程的 pid(进程号)

给子进程指定函数传递参数 Demo：

```
1 . import os from multiprocessing import Process
2 . import time
3 .
4 . def pro_func(name, age, **kwargs):
5 .     for i in range(5):
6 .         print("子进程正在运行中,name=%s, age=%d, pid=%d" %(name, age, os.getpid()))
7 .         print(kwargs)
8 .         time.sleep(0.2)
9 .
10 . if __name__ == '__main__':
11 .     # 创建 Process 对象
12 .     p = Process(target=pro_func, args=('小明',18), kwargs={'m': 20})
13 .     # 启动进程
14 .     p.start()
```

```
15 .    time.sleep(1)
16 .    # 1 秒钟之后，立刻结束子进程
17 .    p.terminate()
18 .    p.join()
```

注意：进程间不共享全局变量。

## 进程之间的通信-Queue

在初始化 Queue() 对象时，(例如 q=Queue())，若在括号中没有指定最大可接受的消息数量，或数量为负值时，那么就代表可接受的消息数量没有上限-直到内存的尽头)

Queue.qsize()：返回当前队列包含的消息数量。

Queue.empty()：如果队列为空，返回 True,反之 False。

Queue.full()：如果队列满了，返回 True，反之 False。

Queue.get([block[,timeout]])：获取队列中的一条消息，然后将其从队列中移除，block 默认值为 True。

如果 block 使用默认值，且没有设置 timeout（单位秒），消息列队如果为空，此时程序将被阻塞（停在读取状态），直到从消息列队读到消息为止，如果设置了 timeout，则会等待 timeout 秒，若还没读取到任何消息，则抛出"Queue.Empty"异常；

如果 block 值为 False，消息列队如果为空，则会立刻抛出"Queue.Empty"异常；

Queue.get\_nowait()：相当 Queue.get(False)；

Queue.put(item,[block[, timeout]])：将 item 消息写入队列，block 默认值为 True；

如果 block 使用默认值，且没有设置 timeout（单位秒），消息列队如果已经没有空间可写入，此时程序将被阻塞（停在写入状态），直到从消息列队腾出空间为止，如果设置了 timeout，则会等待 timeout 秒，若还没空间，则抛出"Queue.Full"异常；

如果 block 值为 False，消息列队如果没有空间可写入，则会立刻抛出"Queue.Full"异常；

Queue.put\_nowait(item)：相当 Queue.put(item, False)；

## 进程间通信 Demo :

```
1 . from multiprocessing import Process, Queue
import os, time, random
2 . # 写数据进程执行的代码:
def write(q):
3 .     for value in ['A', 'B', 'C']:
4 .         print("Put %s to queue..." % value)
5 .         q.put(value)
6 .         time.sleep(random.random())
7 . # 读数据进程执行的代码:
def read(q):
8 .     while True:
9 .         if not q.empty():
10 .             value = q.get(True)
11 .             print('Get %s from queue.' % value)
12 .             time.sleep(random.random())
13 .         else:
14 .             break
15 . if __name__ == '__main__':
16 .     # 父进程创建 Queue，并传给各个子进程：
17 .     q = Queue()
18 .     pw = Process(target=write, args=(q,))
19 .     pr = Process(target=read, args=(q,))
20 .     # 启动子进程 pw，写入:
21 .     pw.start()
22 .     # 等待 pw 结束:
23 .     pw.join()
24 .     # 启动子进程 pr，读取:
25 .     pr.start()
26 .     pr.join()
27 .     # pr 进程里是死循环，无法等待其结束，只能强行终止:
28 .     print("")
29 .     print('所有数据都写入并且读完')
```

## 进程池 Pool

```
1 . # -*- coding:utf-8 -*-
2 . from multiprocessing import Pool
import os, time, random
3 . def worker(msg):
4 .     t_start = time.time()
5 .     print("%s 开始执行,进程号为%d" % (msg, os.getpid()))
6 .     # random.random()随机生成 0~1 之间的浮点数
7 .     time.sleep(random.random()*2)
8 .     t_stop = time.time()
9 .     print(msg, "执行完毕，耗时%.2f" % (t_stop-t_start))
10 .
11 . po = Pool(3) # 定义一个进程池，最大进程数 3
```

```
12 . for i in range(0,10):
13 .     # Pool().apply_async(要调用的目标,(传递给目标的参数元祖,))
14 .     # 每次循环将会用空闲出来的子进程去调用目标
15 .     po.apply_async(worker,(i,))
16 .
17 . print("----start----")
18 . po.close() # 关闭进程池，关闭后 po 不再接收新的请求
19 . po.join() # 等待 po 中所有子进程执行完成，必须放在 close 语句之后
20 . print("-----end-----")
```

multiprocessing.Pool 常用函数解析：

- `apply_async(func[, args[, kwds]])`：使用非阻塞方式调用 `func`（并行执行，堵塞方式必须等待上一个进程退出才能执行下一个进程），`args` 为传递给 `func` 的参数列表，`kwds` 为传递给 `func` 的关键字参数列表；
- `close()`：关闭 Pool，使其不再接受新的任务；
- `terminate()`：不管任务是否完成，立即终止；
- `join()`：主进程阻塞，等待子进程的退出，必须在 `close` 或 `terminate` 之后使用；

进程池中使用 Queue

如果要使用 Pool 创建进程，就需要使用 `multiprocessing.Manager()` 中的 `Queue()`，而不是 `multiprocessing.Queue()`，否则会得到一条如下的错误信息：

RuntimeError: Queue objects should only be shared between processes through inheritance.

```
1 . from multiprocessing import Manager, Pool
import os, time, random
2 . def reader(q):
3 .     print("reader 启动(%s),父进程为(%s)" % (os.getpid(), os.getppid()))
4 .     for i in range(q.qsize()):
5 .         print("reader 从 Queue 获取到消息：%s" % q.get(True))
6 . def writer(q):
7 .     print("writer 启动(%s),父进程为(%s)" % (os.getpid(), os.getppid()))
8 .     for i in "itcast":
9 .         q.put(i)
10 . if __name__ == "__main__":
11 .     print("(%s) start" % os.getpid())
```

```
12 .    q = Manager().Queue() # 使用 Manager 中的 Queue
13 .    po = Pool()
14 .    po.apply_async(writer, (q,))
15 .
16 .    time.sleep(1) # 先让上面的任务向 Queue 存入数据，然后再让下面的任务开始从中取数据
17 .
18 .    po.apply_async(reader, (q,))
19 .    po.close()
20 .    po.join()
21 .    print("(%s) End" % os.getpid())
```

## 2. 谈谈你对多进程，多线程，以及协程的理解，项目是否用？(2018-3-30-lxy)

这个问题被问的概率相当之大，其实多线程，多进程，在实际开发中用到的很少，除非是那些对项目性能要求特别高的，有的开发工作几年了，也确实没用过，你可以这么回答，给他扯扯什么是进程，线程（cpython 中是伪多线程）的概念就行，实在不行你就说你之前写过下载文件时，用过多线程技术，或者业余时间用过多线程写爬虫，提升效率。

进程：一个运行的程序（代码）就是一个进程，没有运行的代码叫程序，进程是系统资源分配的最小单位，进程拥有自己独立的内存空间，所以进程间数据不共享，开销大。

线程：调度执行的最小单位，也叫执行路径，不能独立存在，依赖进程存在一个进程至少有一个线程，叫主线程，而多个线程共享内存(数据共享，共享全局变量)，从而极大地提高了程序的运行效率。

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。

### 3. 什么是多线程竞争？(2018-3-30-lxy)

线程是非独立的，同一个进程里线程是数据共享的，当各个线程访问数据资源时会出现竞争状态即：数据几乎同步会被多个线程占用，造成数据混乱，即所谓的线程不安全

那么怎么解决多线程竞争问题？-- 锁。

锁的好处：

确保了某段关键代码(共享数据资源)只能由一个线程从头到尾完整地执行能解决多线程资源竞争下的原子操作问题。

锁的坏处：

阻止了多线程并发执行，包含锁的某段代码实际上只能以单线程模式执行，效率就大大地下降了

锁的致命问题：死锁。

### 4. 解释一下什么是锁，有哪几种锁？(2018-3-30-lxy)

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。

### 5. 什么是死锁呢？(2018-3-30-lxy)

若干子线程在系统资源竞争时，都在等待对方对某部分资源解除占用状态，结果是谁也不愿先解锁，互相干等着，程序无法执行下去，这就是死锁。

GIL 锁（有时候，面试官不问，你自己要主动说，增加 b 格，尽量别一问一答的尬聊，不然最后等到的一句话就是：你还有什么想问的么？）

GIL 锁 全局解释器锁（只在 cpython 里才有）

作用：限制多线程同时执行，保证同一时间只有一个线程执行，所以 cpython 里的多线程其实是伪多线程！



所以 Python 里常常使用协程技术来代替多线程，协程是一种更轻量级的线程，

进程和线程的切换时由系统决定，而协程由我们程序员自己决定，而模块 `gevent` 下切换是遇到了耗时操作才会切换。

三者的关系：进程里有线程，线程里有协程。

## 6. 什么是线程安全，什么是互斥锁？(2018-3-30-lxy)

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。

## 7. 说说下面几个概念：同步，异步，阻塞，非阻塞?(2018-3-30-lxy)

同步：多个任务之间有先后顺序执行，一个执行完下个才能执行。

异步：多个任务之间没有先后顺序，可以同时执行有时候一个任务可能要在必要的时候获取另一个同时执行的任务的结果，这个就叫回调！

阻塞：如果卡住了调用者，调用者不能继续往下执行，就是说调用者阻塞了。

非阻塞：如果不会卡住，可以继续执行，就是说非阻塞的。

同步异步相对于多任务而言，阻塞非阻塞相对于代码执行而言。

## 8. 什么是僵尸进程和孤儿进程？怎么避免僵尸进程？(2018-3-30-lxy)

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。

## 9. Python 中的进程与线程的使用场景？(2018-3-30-lxy)

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)。

## 10.线程是并发还是并行，进程是并发还是并行？(2018-3-30-lxy)

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。

## 11.并行 ( parallel ) 和并发 ( concurrency ) ？(2018-3-30-lxy)

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。

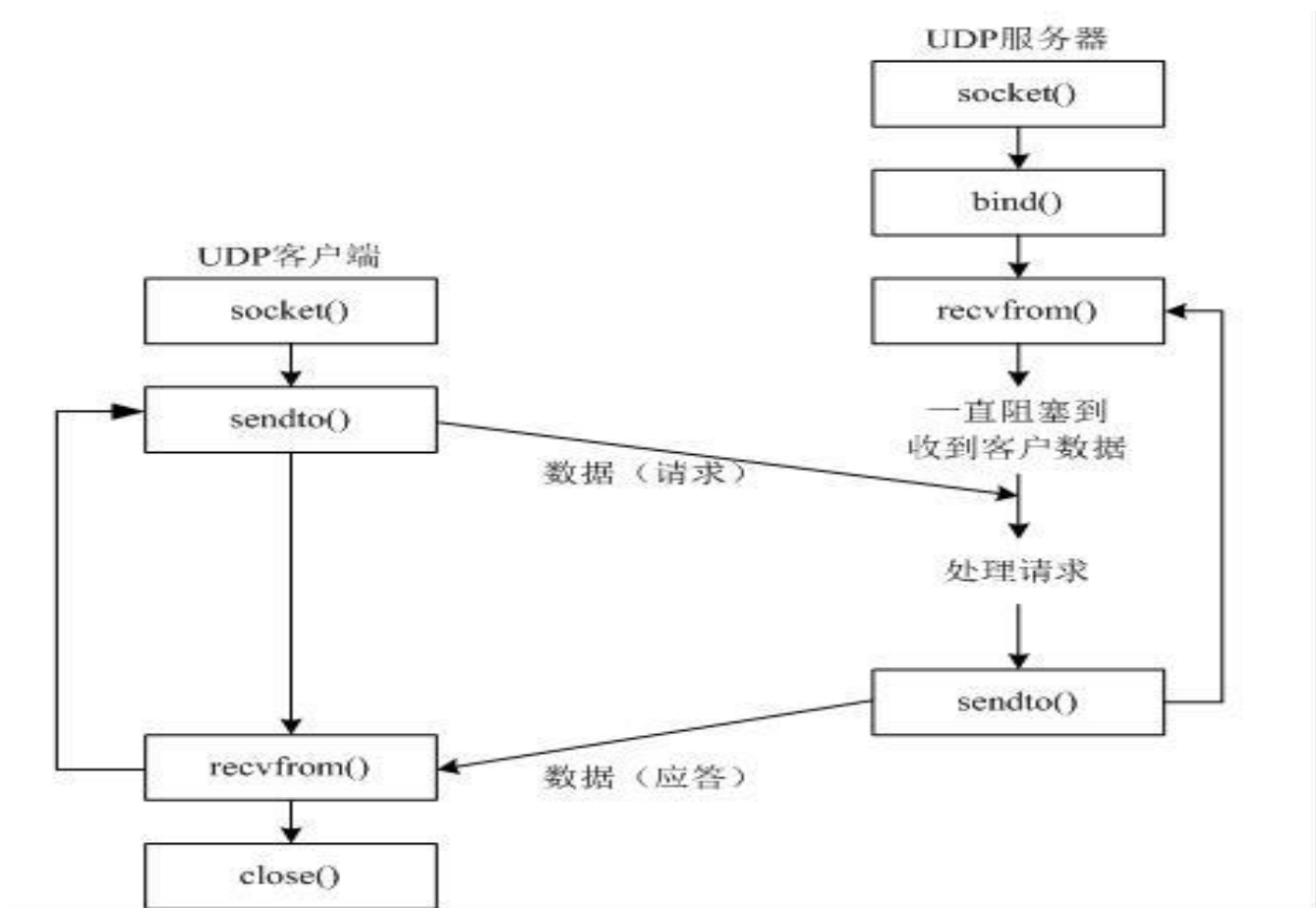
## 12.IO 密集型和 CPU 密集型区别？(2018-4-16-lxy)

IO 密集型：系统运作，大部分的状况是 CPU 在等 I/O (硬盘/内存)的读/写。

CPU 密集型：大部份时间用来做计算、逻辑判断等 CPU 动作的程序称之 CPU 密集型。

## 八. 网络编程

### 1. UDP 总结



使用 udp 发送/接收数据步骤：

1. 创建客户端套接字
2. 发送/接收数据
3. 关闭套接字

```
1. import socket
2. def main():
3.     # 1、创建 udp 套接字
4.     # socket.AF_INET 表示 IPv4 协议 AF_INET6 表示 IPv6 协议
5.     # socket.SOCK_DGRAM 数据报套接字，只用于 udp 协议
6.     udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7.
```

```
8. # 2、准备接收方的地址
9. # 元组类型 ip 是字符串类型 端口号是整型
10. dest_addr = ('192.168.113.111', 8888)
11. # 要发送的数据
12. send_data = "我是要发送的数据"
13. # 3、发送数据
14. udp_socket.sendto(send_data.encode("utf-8"), dest_addr)
15. # 4、等待接收方发送的数据 如果没有收到数据则会阻塞等待，直到收到数据
16. # 接收到的数据是一个元组 (接收到的数据, 发送方的 ip 和端口)
17. # 1024 表示本次接收的最大字节数
18. recv_data, addr = udp_socket.recvfrom(1024)
19. # 5、关闭套接字
20. udp_socket.close()
21. if __name__ == '__main__':
22.     main()
```

## 编码的转换

str --> bytes: encode 编码

bytes--> str: decode()解码

## UDP 绑定端口号：

1.创建 socket 套接字

2.绑定端口号

3.接收/发送数据

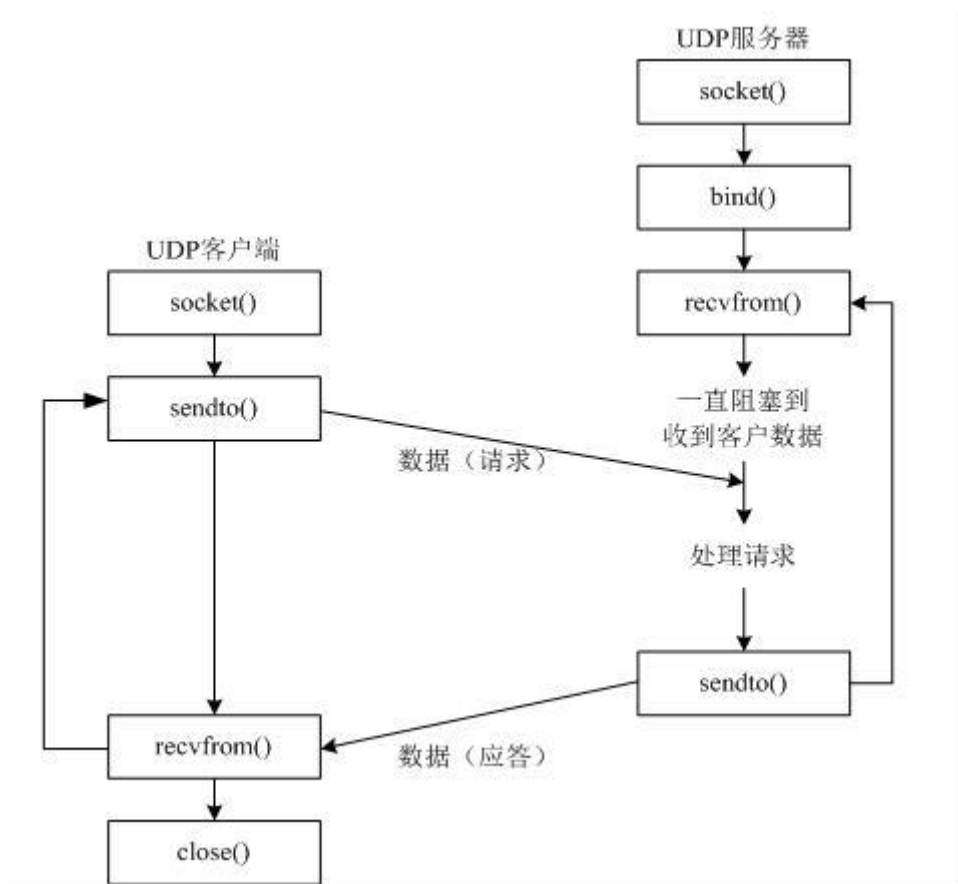
4.关闭套接字

```
1. import socket
2. def main():
3.     # 1、创建 udp 套接字
4.     # socket.AF_INET 表示 IPv4 协议 AF_INET6 表示 IPv6 协议
5.     # socket.SOCK_DGRAM 数据报套接字，只用于 udp 协议
6.     udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7.     # 2、绑定端口
8.     # 元组类型 ip 一般不写 表示本机的任何一个 ip
9.     local_addr = ('', 7777)
10.    udp_socket.bind(local_addr)
11.    # 3、准备接收方的地址
12.    # 元组类型 ip 是字符串类型 端口号是整型
13.    dest_addr = ('192.168.113.111', 8888)
```

```
14 . # 要发送的数据
15 . send_data = "我是要发送的数据"
16 . # 4、发送数据
17 . udp_socket.sendto(send_data.encode("utf-8"), dest_addr)
18 . # 5、等待接收方发送的数据 如果没有收到数据则会阻塞等待，直到收到数据
19 . # 接收到的数据是一个元组 (接收到的数据, 发送方的 ip 和端口)
20 . # 1024 表示本次接收的最大字节数
21 . recv_data, addr = udp_socket.recvfrom(1024)
22 . # 6、关闭套接字
23 . udp_socket.close()
24 . if __name__ == '__main__':
25 .     main()
```

注意点：绑定端口要在发送数据之前进行绑定。

## 2. TCP 总结



TCP 客户端的创建流程：

### 1. 创建 TCP 的 socket 套接字

## 2.连接服务器

## 3.发送数据给服务器端

## 4.接收服务器端发送来的消息

## 5.关闭套接字

```
1 . import socket
2 . def main():
3 .     # 1、创建客户端的 socket
4 .     # socket.AF_INET 表示 IPv4 协议 AF_INET6 表示 IPv6 协议
5 .     # socket.SOCK_STREAM 流式套接字，只用于 TCP 协议
6 .     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 .     # 2、构建目标地址
8 .     server_ip = input("请输入服务器端的 IP 地址：")
9 .     server_port = int(input("请输入服务器端的端口号："))
10 .    # 3、连接服务器
11 .    # 参数：元组类型 ip 是字符串类型 端口号是整型
12 .    client_socket.connect((server_ip, server_port))
13 .    # 要发送给服务器端的数据
14 .    send_data = "我要发送给服务器端的数据"
15 .    # 4、发送数据
16 .    client_socket.send(send_data.encode("gbk"))
17 .    # 5、接收服务器端恢复的消息， 没有消息会阻塞
18 .    # 1024 表示接收的最大字节数
19 .    recv_data = client_socket.recv(1024)
20 .    print("接收到的数据是：", recv_data.decode('gbk'))
21 .    # 6、关闭套接字
22 .    client_socket.close()
23 . if __name__ == '__main__':
24 .    main()
```

## TCP 服务器端的创建流程

### 1.创建 TCP 服务端的 socket

### 2.bing 绑定 ip 地址和端口号

### 3.listen 使套接字变为被动套接字

### 4.accept 取出一个客户端连接，用于服务

### 5.recv/send 接收和发送消息

## 6.关闭套接字

```
1 . import socket
2 .
3 . def main():
4 .     # 1、创建 tcp 服务端的 socket
5 .     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 .
7 .     # 2、绑定
8 .     server_socket.bind(('', 8888))
9 .
10 .    # 3、listen 使套接字变为被动套接字
11 .    server_socket.listen(128)
12 .
13 .    # 4、如果有新的客户端来链接服务器，那么就产生一个新的套接字专门为这个客户端服务
14 .    # client_socket 用来为这个客户端服务
15 .    # tcp_server_socket 就可以省下来专门等待其他新客户端的连接
16 .    client_socket, client_addr = server_socket.accept()
17 .
18 .    # 5、接收客户端发来的消息
19 .    recv_data = client_socket.recv(1024)
20 .    print("接收到客户端%s 的数据：%s" % (str(client_addr), recv_data.decode('gbk')))
21 .
22 .    # 6、回复数据给客户端
23 .    client_socket.send("收到消息".encode('gbk'))
24 .
25 .    # 7、关闭套接字
26 .    client_socket.close()
27 .    server_socket.close()
28 .
29 . if __name__ == '__main__':
30 .     main()
```

注意点：

- tcp 服务器一般都需要绑定，否则客户端找不到服务器
- tcp 客户端一般不绑定，因为是主动链接服务器，所以只要确定好服务器的 ip、port 等信息就好，本地客户端可以随机
- tcp 服务器中通过 listen 可以将 socket 创建出来的主动套接字变为被动的，这是做 tcp 服务器时必须做的



- 当客户端需要链接服务器时，就需要使用 connect 进行链接，udp 是不需要链接的而是直接发送，但是 tcp 必须先链接，只有链接成功才能通信
- 当一个 tcp 客户端连接服务器时，服务器端会有 1 个新的套接字，这个套接字用来标记这个客户端，单独为这个客户端服务
- listen 后的套接字是被动套接字，用来接收新的客户端的连接请求的，而 accept 返回的新套接字是标识这个新客户端的
- 关闭 listen 后的套接字意味着被动套接字关闭了，会导致新的客户端不能够链接服务器，但是之前已经链接成功的客户端正常通信。
- 关闭 accept 返回的套接字意味着这个客户端已经服务完毕
- 当客户端的套接字调用 close 后，服务器端会 recv 解阻塞，并且返回的长度为 0，因此服务器可以通过返回数据的长度来区别客户端是否已经下线；同理 当服务器断开 tcp 连接的时候 客户端同样也会收到 0 字节数据。

### 3. 怎么实现强行关闭客户端和服务端之间的连接? (2018-3-30-lxy)

在 socket 通信过程中不断循环检测一个全局变量(开关标记变量)，一旦标记变量变为关闭，则调用 socket 的 close 方法，循环结束，从而达到关闭连接的目的。

### 4. 简述 TCP 和 UDP 的区别以及优缺点? (2018-4-16-lxy)

UDP 是面向无连接的通讯协议，UDP 数据包括目的端口号和源端口号信息。

优点：UDP 速度快、操作简单、要求系统资源较少，由于通讯不需要连接，可以实现广播发送

缺点：UDP 传送数据前并不与对方建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，也不重复发送，不可靠。

TCP 是面向连接的通讯协议，通过三次握手建立连接，通讯完成时四次挥手

优点：TCP 在数据传递时，有确认、窗口、重传、阻塞等控制机制，能保证数据正确性，较为可靠。

缺点：TCP 相对于 UDP 速度慢一点，要求系统资源较多。

## 5. 简述浏览器通过 WSGI 请求动态资源的过程? (2018-4-16-lxy)

- 1.发送 http 请求动态资源给 web 服务器
- 2.web 服务器收到请求后通过 WSGI 调用一个属性给应用程序框架
- 3.应用程序框架通过引用 WSGI 调用 web 服务器的方法，设置返回的状态和头信息。
- 4.调用后返回，此时 web 服务器保存了刚刚设置的信息
- 5.应用程序框架查询数据库，生成动态页面的 body 的信息
- 6.把生成的 body 信息返回给 web 服务器
- 7.web 服务器吧数据返回给浏览器

## 6. 描述用浏览器访问 www.baidu.com 的过程(2018-4-16-lxy)

先要解析出 baidu.com 对应的 ip 地址

- 要先使用 arp 获取默认网关的 mac 地址
- 组织数据发送给默认网关(ip 还是 dns 服务器的 ip，但是 mac 地址是默认网关的 mac 地址)
- 默认网关拥有转发数据的能力，把数据转发给路由器
- 路由器根据自己的路由协议，来选择一个合适的较快的路径转发数据给目的网关
- 目的网关(dns 服务器所在的网关)，把数据转发给 dns 服务器
- dns 服务器查询解析出 baidu.com 对应的 ip 地址，并原路返回请求这个域名的 client

得到了 baidu.com 对应的 ip 地址之后，会发送 tcp 的 3 次握手，进行连接

- 使用 http 协议发送请求数据给 web 服务器
- web 服务器收到数据请求之后，通过查询自己的服务器得到相应的结果，原路返回给浏览器。
- 浏览器接收到数据之后通过浏览器自己的渲染功能来显示这个网页。
- 浏览器关闭 tcp 连接，即 4 次挥手结束，完成整个访问过程

## 7. Post 和 Get 请求的区别? (2018-4-16-lxy)

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 unicode，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。

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。

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 forgery 攻击。

效率：GET 比 POST 效率高。

POST 请求的过程：

- 1.浏览器请求 tcp 连接（第一次握手）
- 2.服务器答应进行 tcp 连接（第二次握手）
- 3.浏览器确认，并发送 post 请求头（第三次握手，这个报文比较小，所以 http 会在此时进行

第一次数据发送）

- 4.服务器返回 100 continue 响应
- 5.浏览器开始发送数据
- 6.服务器返回 200 ok 响应

GET 请求的过程：

- 1.浏览器请求 tcp 连接（第一次握手）
- 2.服务器答应进行 tcp 连接（第二次握手）
- 3.浏览器确认，并发送 get 请求头和数据（第三次握手，这个报文比较小，所以 http 会在此时

进行第一次数据发送）

- 4.服务器返回 200 OK 响应

## 8. cookie 和 session 的区别？(2018-4-16-lxy)

- 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

9. 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？(2018-4-16-lxy)

通过状态码告诉客户端服务器的执行状态，以判断下一步该执行什么操作。

常见的状态码有：

100-199：表示服务器成功接收部分请求，要求客户端继续提交其余请求才能完成整个处理过程。

200-299：表示服务器成功接收请求并已完成处理过程，常用 200（OK 请求成功）。

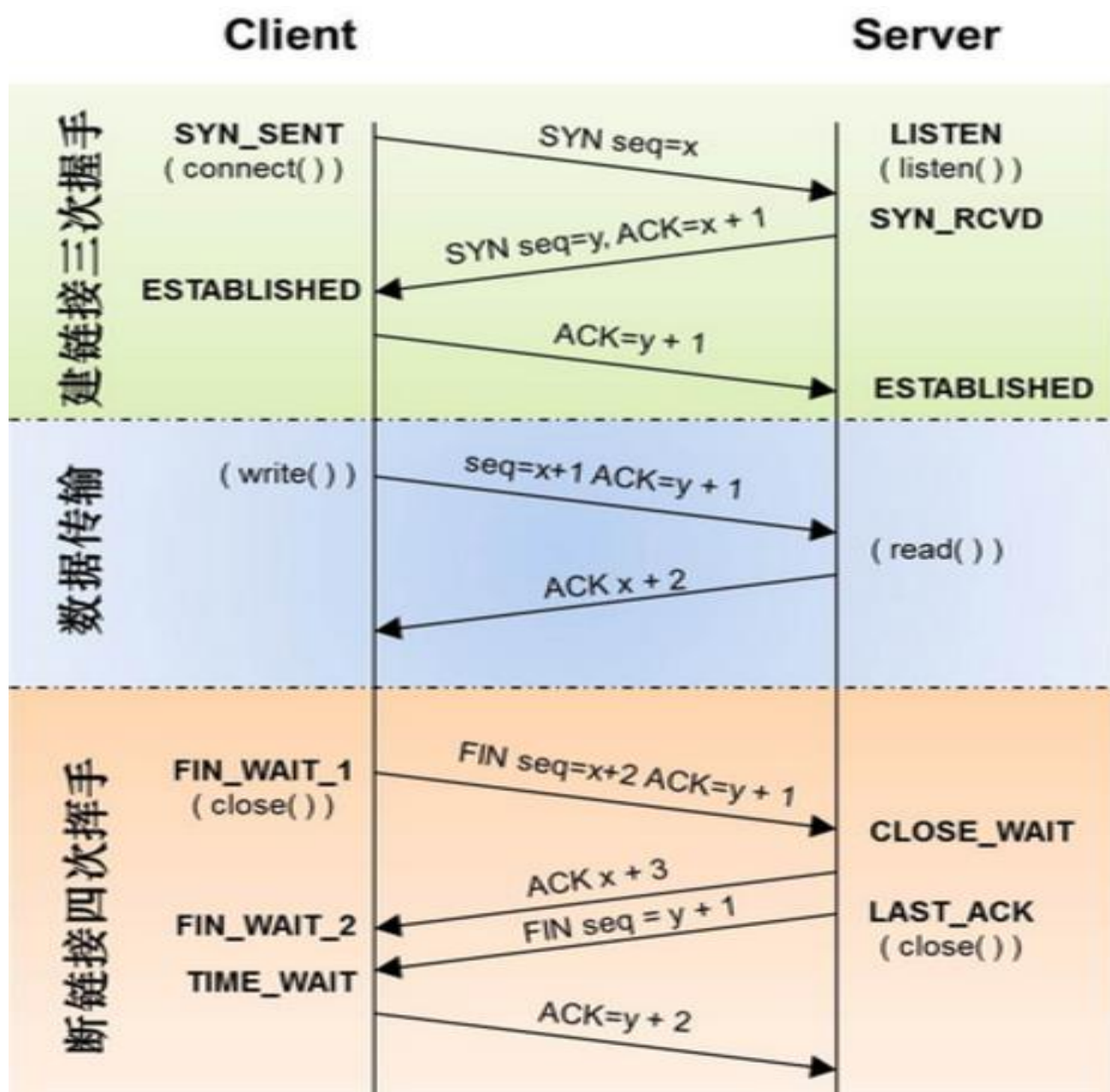
300-399：为完成请求，客户需要进一步细化请求。302（所有请求页面已经临时转移到新的 url）。

304、307（使用缓存资源）。

400-499：客户端请求有错误，常用 404（服务器无法找到被请求页面），403（服务器拒绝访问，权限不够）。

500-599：服务器端出现错误，常用 500（请求未完成，服务器遇到不可预知的情况）。

10.请简单说一下三次握手和四次挥手？(2018-4-20-lxy)



三次握手过程：

- 1 首先客户端向服务端发送一个带有 SYN 标志，以及随机生成的序号 100(0 字节)的报文
- 2 服务端收到报文后返回一个报文(SYN200(0 字节)，ACK1001(字节+1))给客户端
- 3 客户端再次发送带有 ACK 标志 201(字节+)序号的报文给服务端

至此三次握手过程结束，客户端开始向服务端发送数据。

1 客户端向服务端发起请求：我想给你通信，你准备好了么？

2 服务端收到请求后回应客户端：I'ok，你准备好了么

3 客户端礼貌的再次回一下客户端：准备就绪，咱们开始通信吧！

整个过程跟打电话的过程一模一样：1 喂，你在吗 2 在，我说的你听得到不 3 恩，听得到(接下来请开始你的表演)

补充：SYN：请求询问，ACK：回复，回应。

四次挥手过程：

由于 TCP 连接是可以双向通信的（全双工），因此每个方向都必须单独进行关闭（这句话才是精辟，后面四个挥手过程都是其具体实现的语言描述）

四次挥手过程，客户端和服务端都可以先开始断开连接

1 客户端发送带有 fin 标识的报文给服务端，请求通信关闭

2 服务端收到信息后，回复 ACK 答应关闭客户端通信(连接)请求

3 服务端发送带有 fin 标识的报文给客户端，也请求关闭通信

4 客户端回应 ack 给服务端，答应关闭服务端的通信(连接)请求

## 11.说一下什么是 tcp 的 2MSL ? (2018-4-20-lxy)

主动发送 fin 关闭的一方，在 4 次挥手最后一次要等待一段时间我们称这段时间为 2MSL

TIME\_WAIT 状态的存在有两个理由：

1.让 4 次挥手关闭流程更加可靠

2.防止丢包后对后续新建的正常连接的传输造成破坏



## 12.为什么客户端在 TIME-WAIT 状态必须等待 2MSL 的时间？(2018-4-20-lxy)

1、为了保证客户端发送的最后一个 ACK 报文段能够达到服务器。这个 ACK 报文段可能丢失，因而使处在 LAST-ACK 状态的服务器收不到确认。服务器会超时重传 FIN+ACK 报文段，客户端就能在 2MSL 时间内收到这个重传的 FIN+ACK 报文段，接着客户端重传一次确认，重启计时器。最好，客户端和服务器都正常进入到 CLOSED 状态。如果客户端在 TIME-WAIT 状态不等待一段时间，而是再发送完 ACK 报文后立即释放连接，那么就无法收到服务器重传的 FIN+ACK 报文段，因而也不会再发送一次确认报文。这样，服务器就无法按照正常步骤进入 CLOSED 状态。

2、防止已失效的连接请求报文段出现在本连接中。客户端在发送完最后一个 ACK 确认报文段后，再经过时间 2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段。

## 13.说说 HTTP 和 HTTPS 区别？(2018-4-23-lxy)

HTTP 协议传输的数据都是未加密的，也就是明文的，因此使用 HTTP 协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了 SSL ( Secure Sockets Layer ) 协议用于对 HTTP 协议传输的数据进行加密，从而就诞生了 HTTPS。简单来说，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全。

HTTPS 和 HTTP 的区别主要如下：

- 1、https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用。
- 2、http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。
- 3、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。



4、http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

#### 14.谈一下 HTTP 协议以及协议头部中表示数据类型的字段？（2018-4-23-lxy）

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。它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第六版，HTTP/1.1 的规范化工作正在进行之中，而且 HTTP-NG(Next Generation of HTTP)的建议已经提出。

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

#### 15.HTTP 请求方法都有什么？（2018-4-23-lxy）

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法：GET，POST 和 HEAD 方法。

HTTP1.1 新增了五种请求方法：OPTIONS，PUT，DELETE，TRACE 和 CONNECT 方法。

1、GET 请求指定的页面信息，并返回实体主体。

2、HEAD 类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头

3、POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。

4、PUT 从客户端向服务器传送的数据取代指定的文档的内容。

5、DELETE 请求服务器删除指定的页面。

6、CONNECT HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

7、OPTIONS 允许客户端查看服务器的性能。

8、TRACE 回显服务器收到的请求，主要用于测试或诊断。

## 16.使用 Socket 套接字需要传入哪些参数？（2018-4-23-lxy）

Address Family 和 Type，分别表示套接字应用场景和类型。

family 的值可以是 AF\_UNIX(Unix 域，用于同一台机器上的进程间通讯)，也可以是 AF\_INET（对于 IPV4 协议的 TCP 和 UDP），至于 type 参数，SOCK\_STREAM（流套接字）或者 SOCK\_DGRAM（数据报文套接字），SOCK\_RAW（raw 套接字）。

## 17.HTTP 常见请求头？（2018-4-23-lxy）

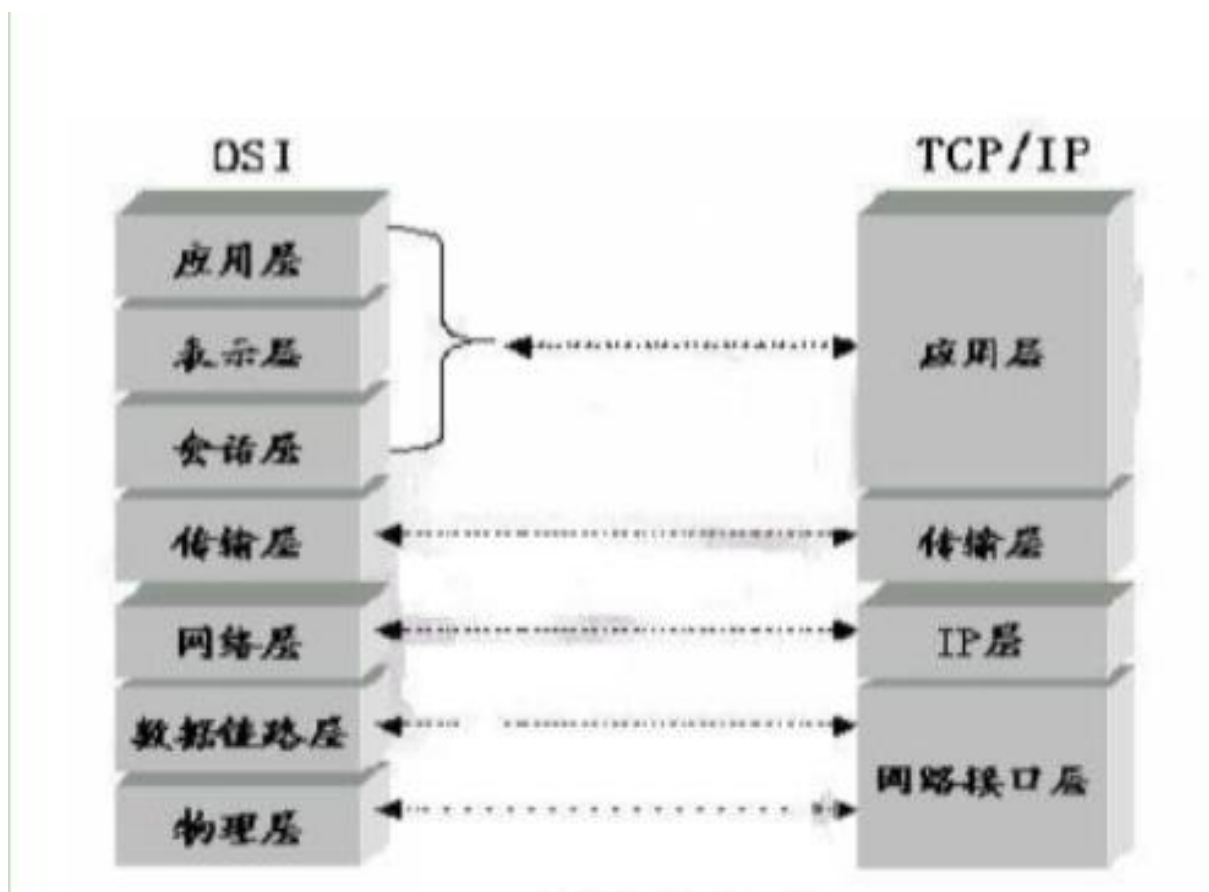
1. Host (主机和端口号)
2. Connection (链接类型)
3. Upgrade-Insecure-Requests (升级为 HTTPS 请求)
4. User-Agent (浏览器名称)
5. Accept (传输文件类型)
6. Referer (页面跳转处)
7. Accept-Encoding (文件编解码格式)
8. Cookie (Cookie)
9. x-requested-with :XMLHttpRequest (是 Ajax 异步请求)

## 18.七层模型？IP，TCP/UDP，HTTP，RTSP，FTP 分别在哪层？(2018-4-23-lxy)

IP：网络层

TCP/UDP：传输层

HTTP、RTSP、FTP：应用层协议



## 19.url 的形式？(2018-4-23-lxy)

形式：scheme://host[:port#]/path/.../[?query-string][#anchor]

scheme：协议(例如：http，https，ftp)

host：服务器的 IP 地址或者域名

port : 服务器的端口 ( 如果是走协议默认端口 , 80 or 443 )

path : 访问资源的路径

query-string : 参数 , 发送给 http 服务器的数据

anchor : 锚 ( 跳转到网页的指定锚点位置 )

http://localhost:4000/file/part01/1.2.html

## 第四章 前端

### 一 . Html

### 二 . Css

#### 1. 什么是 CSS 初始化 ? 有什么好处 ? (2018-4-16-lxy)

CSS 初始化是指重设浏览器的样式。不同的浏览器默认的样式可能不尽相同 , 如果没对 CSS 初始化往往会出现浏览器之间的页面差异。

好处 : 能够统一标签在各大主流浏览器中的默认样式 , 使得我们开发网页内容时更加方便简洁 , 同时减少 CSS 代码量 , 节约网页下载时间。

#### 2. 简述浮动的特征和清除浮动的方法 ? (2018-4-16-lxy)

浮动的特征 :

浮动元素有左浮动(float:left)和右浮动(float:right)两种。

浮动的元素会向左或向右浮动 , 碰到父元素边界、其他元素才停下来。

相邻浮动的块元素可以并在一行，超出父级宽度就换行。

浮动让行内元素或块元素转化为有浮动特性的行内块元素(此时不会有行内块元素间隙问题)。

父元素如果没有设置尺寸(一般是高度不设置)，父元素内整体浮动的子元素无法撑开父元素，父元素需要清除浮动。

清除浮动的方法：

父级上增加属性 `overflow : hidden`。

在最后一个子元素的后面加一个空的 `div`，给它样式属性 `clear:both`。

使用成熟的清浮动样式类，`clearfix`。

```
1. .clearfix:after,.clearfix:before{ content: "";display: table;}
2. .clearfix:after{ clear:both;}
3. .clearfix{zoom:1;}
```

### 三 . JavaScript

#### 1. AJAX 是什么？如何使用 AJAX？(2018-4-16-lxy)

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。

## 四 . jQurey

## 五 . vue.js

# 第五章 Web

## 一 . Flask

### 1. Flask 中正则 URL 的实现？ (2018-4-14-lxy)

@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。

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

第二步：把正则类赋值给我们定义的正则规则。

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

第三步：在 URL 中使用正则。

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

### 2. Flask 中请求上下文和应用上下文的区别和作用？ (2018-4-14-lxy)

current\_app、g 是应用上下文。

request、session 是请求上下文。

手动创建上下文的两种方法：

1. with app.app\_context()
2. app = current\_app.\_get\_current\_object()

两者区别：

请求上下文：保存了客户端和服务端交互的数据。

应用上下文：flask 应用程序运行过程中，保存的一些配置信息，比如程序名、数据库连接、应用信息等。

两者作用：

请求上下文(request context)：

Flask 从客户端收到请求时，要让视图函数能访问一些对象，这样才能处理请求。请求对象是一个很好的例子，它封装了客户端发送的 HTTP 请求。

要想让视图函数能够访问请求对象，一个显而易见的方式是将其作为参数传入视图函数，不过这会导致程序中的每个视图函数都增加一个参数，除了访问请求对象，如果视图函数在处理请求时还要访问其他对象，情况会变得更糟。为了避免大量可有可无的参数把视图函数弄得一团糟，Flask 使用上下文临时把某些对象变为全局可访问。

应用上下文(application context)：

它的字面意思是 应用上下文，但它不是一直存在的，它只是 request context 中的一个对 app 的代理(人)，所谓 local proxy。它的作用主要是帮助 request 获取当前的应用，它是伴 request 而生，随 request 而灭的。

### 3. Flask 中数据库设置？(2018-4-14-lxy)

1. app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:mysql@127.0.0.1:3306/test'

# 动态追踪修改设置，如未设置只会提示警告

1. app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = True

## #查询时会显示原始 SQL 语句

1. app.config['SQLALCHEMY\_ECHO'] = True

名字	备注
SQLALCHEMY_DATABASE_URI	用于连接的数据库 URI 。例 如:sqlite:////tmp/test.dbmysql://username:password@server/db
SQLALCHEMY_BINDS	一个映射 binds 到连接 URI 的字典。更多 binds 的信息见 <a href="#">用 Binds 操作多个数据库</a> 。
SQLALCHEMY_ECHO	如果设置为True， SQLAlchemy 会记录所有 发给 stderr 的语句，这对调试有用。(打印sql语句)
SQLALCHEMY_RECORD_QUERIES	可以用于显式地禁用或启用查询记录。查询记录 在调试或测试模式自动启用。更多信息见 <a href="#">get_debug_queries()</a> 。
SQLALCHEMY_NATIVE_UNICODE	可以用于显式禁用原生 unicode 支持。当使用 不合适的指定无编码的数据库默认值时，这对于 一些数据库适配器是必须的（比如 Ubuntu 上 某些版本的 PostgreSQL ）。
SQLALCHEMY_POOL_SIZE	数据库连接池的大小。默认是引擎默认值（通常是 5 ）
SQLALCHEMY_POOL_TIMEOUT	设定连接池的连接超时时间。默认是 10 。
SQLALCHEMY_POOL_RECYCLE	多少秒后自动回收连接。这对 MySQL 是必要的，它默认移除闲置多于 8 小时的连接。注意如果 使用了 MySQL ， Flask-SQLAlchemy 自动设定 这个值为 2 小时。

补充：(2018-4-19-lyf)

app.config['SQLALCHEMY\_COMMIT\_ON\_TEARDOWN']：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()；

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。



#### 4. 常用的 SQLAlchemy 查询过滤器？(2018-4-14-lxy)

过滤器	说明
<code>filter()</code>	把过滤器添加到原查询上，返回一个新查询
<code>filter_by()</code>	把等值过滤器添加到原查询上，返回一个新查询
<code>limit</code>	使用指定的值限定原查询返回的结果
<code>offset()</code>	偏移原查询返回的结果，返回一个新查询
<code>order_by()</code>	根据指定条件对原查询结果进行排序，返回一个新查询
<code>group_by()</code>	根据指定条件对原查询结果进行分组，返回一个新查询

#### 5. 对 Flask 蓝图(Blueprint)的理解？(2018-4-14-lxy)

##### 1) 蓝图的定义

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

```
2. 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...

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

3.在应用对象上注册这个蓝图对象

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

### 6. Flask 中 WTF 表单数据验证？(2018-4-14-lxy)

在 Flask 中，为了处理 web 表单，我们一般使用 Flask-WTF 扩展，它封装了 WTForms，并且它有验证表单数据的功能。

WTForms 支持的 HTML 标准字段：

字段对象	说明
StringField	文本字段
TextAreaField	多行文本字段

字段对象	说明
PasswordField	密码文本字段
HiddenField	隐藏文件字段
DateField	文本字段，值为 datetime.date 文本格式
DateTimeField	文本字段，值为 datetime.datetime 文本格式
IntegerField	文本字段，值为整数
DecimalField	文本字段，值为 decimal.Decimal
FloatField	文本字段，值为浮点数
BooleanField	复选框，值为 True 和 False
RadioField	一组单选框
SelectField	下拉列表
SelectMultipleField	下拉列表，可选择多个值
FileField	文件上传字段
SubmitField	表单提交按钮
FormField	把表单作为字段嵌入另一个表单

字段对象	说明
FieldList	一组指定类型的字段

## WTForms 常用验证函数

验证函数	说明
InputRequired	确保字段中有数据
DataRequired	确保字段中有数据并且数据为真
EqualTo	比较两个字段的值，常用于比较两次密码输入
Length	验证输入的字符串长度
NumberRange	验证输入的值在数字范围内
URL	验证 URL
AnyOf	验证输入值在可选列表中
NoneOf	验证输入值不在可选列表中

使用 Flask-WTF 需要配置参数 SECRET\_KEY。

CSRF\_ENABLED 是为了 CSRF（跨站请求伪造）保护。 SECRET\_KEY 用来生成加密令牌，当 CSRF 激活的时候，该设置会根据设置的密匙生成加密令牌。

## 7. Flask 项目中如何实现 session 信息的写入？(2018-4-14-lxy)

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。

## 8. 项目接口实现后路由访问不到怎么办？(2018-4-14-lxy)

可以通过 postman 测试工具测试，或者看 log 日志信息找到错误信息的大概位置。

## 9. Flask 中 `url_for` 函数？(2018-4-14-lxy)

1.URL 反转：根据视图函数名称得到当前所指向的 url。

2.`url_for()` 函数最简单的用法是以视图函数名作为参数，返回对应的 url，还可以用作加载静态文件。

```
1. <link rel="stylesheet" href="{{url_for('static',filename='css/index.css')}}">
```

该条语句就是在模版中加载 css 静态文件。

3.`url_for` 和 `redirect` 区别

`url_for` 是用来拼接 URL 的，可以使用程序 URL 映射中保存的信息生成 URL。`url_for()` 函数最简单的用法是以视图函数名作为参数，返回对应的 URL。例如，在示例程序中 `hello.py` 中调用 `url_for('index')` 得到的结果是 `/`。

`redirect` 是重定向函数，输入一个 URL 后，自动跳转到另一个 URL 所在的地址，例如，你在函数中写 `return redirect('https://www.baidu.com')` 页面就会跳转向百度页面。

```
1. from flask import Flask, redirect, url_for
2. app = Flask(__name__)
3. @app.route('/')
4. def index():
5.     login_url = url_for('login')
6.     return redirect(login_url)
7.     return u'这是首页'
8.
9. @app.route('/login/')
10. def login():
11.     return u'这是登陆页面'
12.
13. @app.route('/question/<is_login>/')
14. def question(is_login):
15.     if is_login == '1':
16.         return u'这是发布问答的页面'
17.     else:
18.         return redirect(url_for('login'))
19.
20. if __name__ == '__main__':
21.     app.run(debug=True)
```

## 10.Flask 中请求钩子的理解和应用？(2018-4-14-lxy)

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

```
1. @api.after_request
2. def after_request(response):
3.     """设置默认的响应报文格式为 application/json"""
4.     # 如果响应报文 response 的 Content-Type 是以 text 开头，则将其改为
5.     # 默认的 json 类型
```

```
6.         if response.headers.get("Content-Type").startswith("text"):
7.             response.headers["Content-Type"] = "application/json"
8.         return respon
```

## 11. 一个变量后写多个过滤器是如何执行的？ (2018-4-14-lxy)

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

## 12. 如何把整个数据库导出来，再导入指定数据库中？ (2018-4-14-lxy)

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;
```

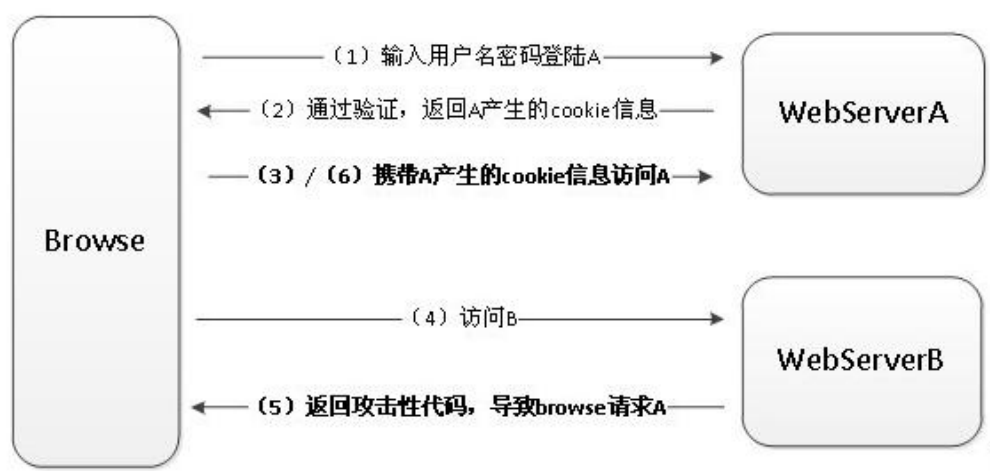
导入指定 sql 文件：

```
mysql>source /path/example.sql;
```

### 13.Flask 和 Django 路由映射的区别？(2018-4-19-lyf)

在 django 中，路由是浏览器访问服务器时，先访问的项目中的 url，再由项目中的 url 找到应用中 url，这些 url 是放在一个列表里，遵从从前往后匹配的规则。在 flask 中，路由是通过装饰器给每个视图函数提供的，而且根据请求方式的不同可以一个 url 用于不同的作用。

### 14.跨站请求伪造和跨站请求保护的实现？(2018-4-19-lyf)



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/点击网站 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。

## 15.Flask(\_\_name\_\_)中的\_\_name\_\_可以传入哪些值？(2018-4-19-lyf)

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

## 二 . Django

### 1. Django 创建项目的命令？(2018-4-14-lxy)

django-admin startproject 项目名称

`python manage.py startapp` 应用 app 名

## 2. Django 创建项目后 项目文件夹下的组成部分( 对 mvt 的理解 )? (2018-4-14-lxy)

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。

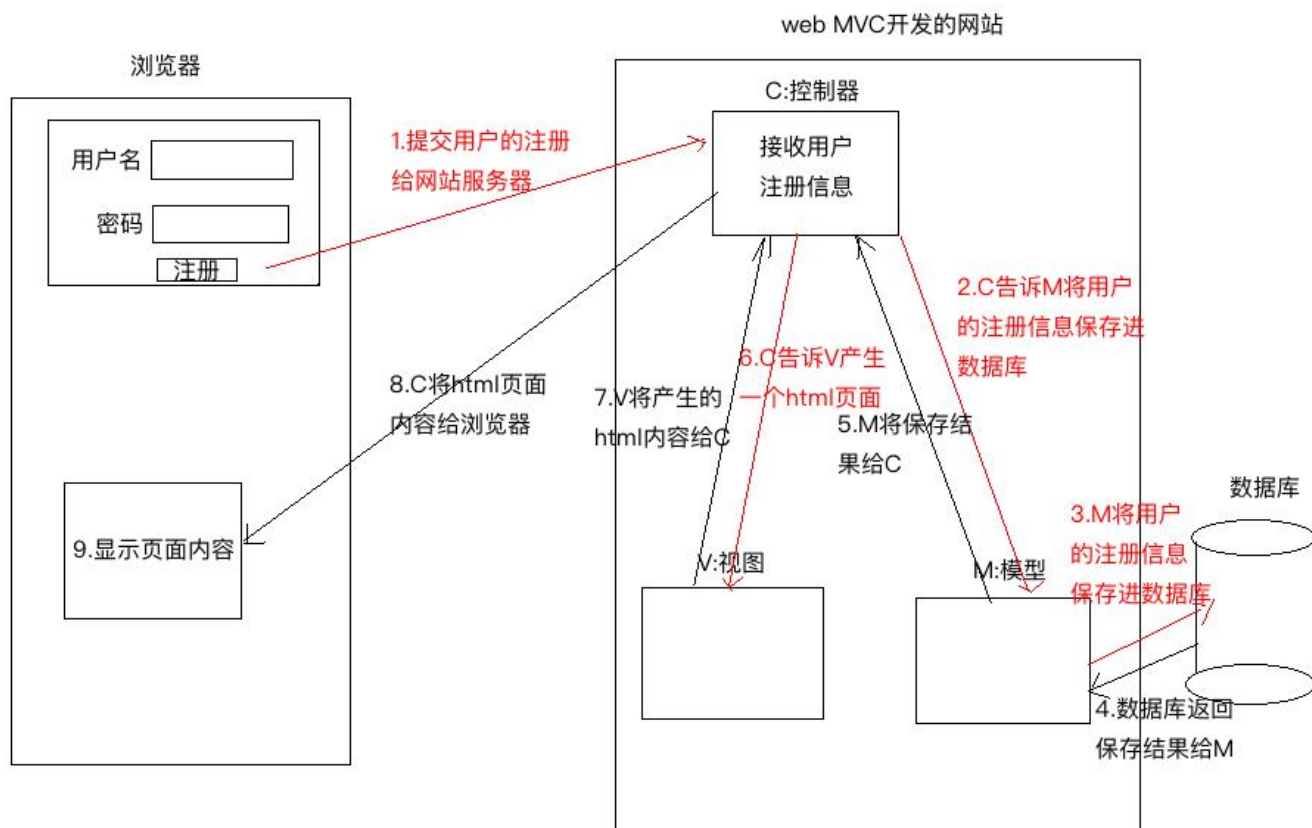
`wsgi.py` 是项目与 WSGI 兼容的 Web 服务器。

## 3. 对 MVC,MVT 解读的理解 ? (2018-4-14-lxy)

M : Model，模型，和数据库进行交互

V : View，视图，负责产生 Html 页面

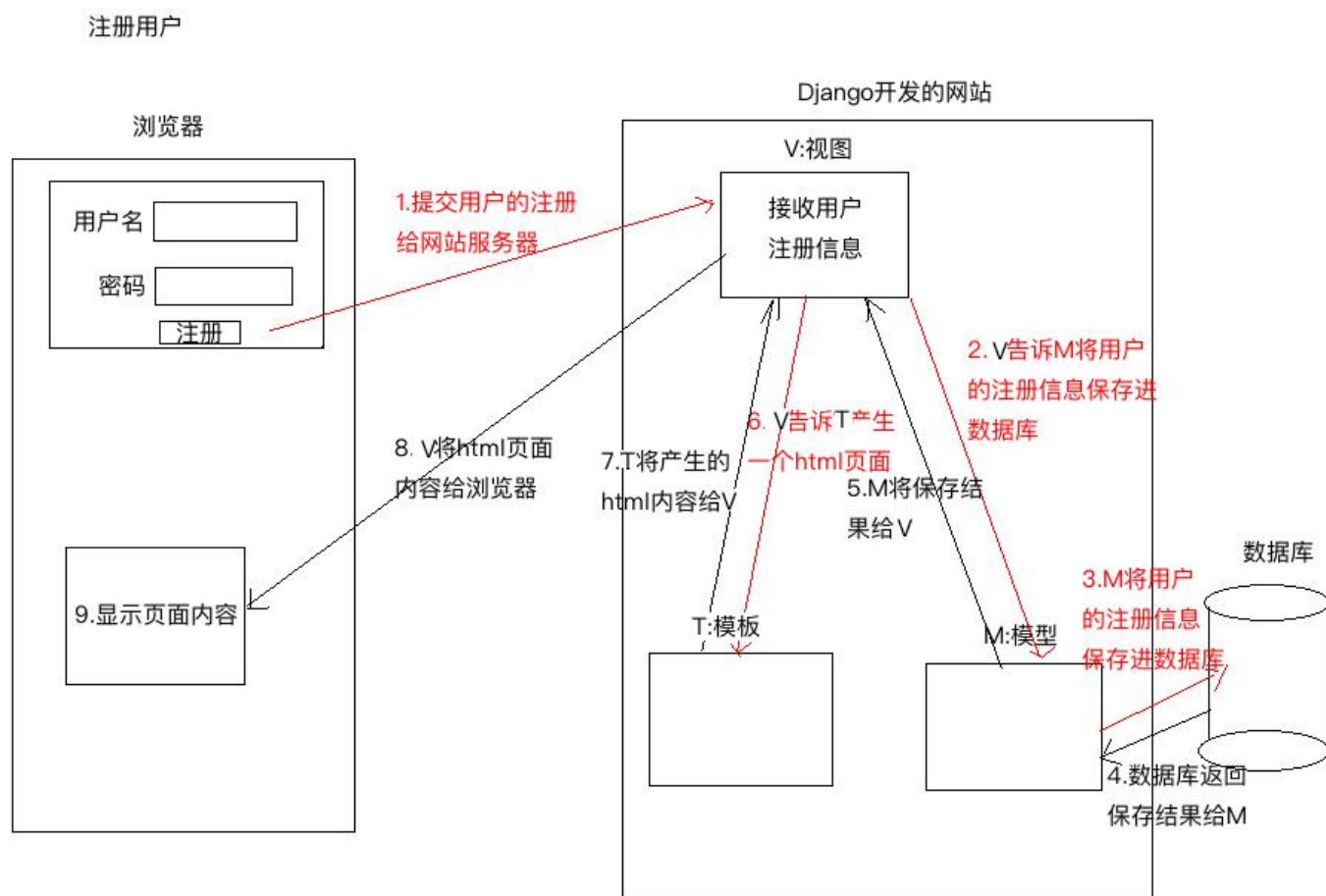
C : Controller，控制器，接收请求，进行处理，与 M 和 V 进行交互，返回应答。



- 1、 用户点击注册按钮，将要注册的信息发送给网站服务器。
  - 2、 Controller 控制器接收到用户的注册信息，Controller 会告诉 Model 层将用户的注册信息保存到数据库
  - 3、 Model 层将用户的注册信息保存到数据库
  - 4、 数据保存之后将保存的结果返回给 Model 模型，
  - 5、 Model 层将保存的结果返回给 Controller 控制器。
  - 6、 Controller 控制器收到保存的结果之后，或告诉 View 视图，view 视图产生一个 html 页面。
  - 7、 View 将产生的 Html 页面的内容给了 Controller 控制器。
  - 8、 Controller 将 Html 页面的内容返回给浏览器。
  - 9、 浏览器接受到服务器 Controller 返回的 Html 页面进行解析展示。
- M：Model，模型，和 MVC 中的 M 功能相同，和数据库进行交互。

V : view , 视图 , 和 MVC 中的 C 功能相同 , 接收请求 , 进行处理 , 与 M 和 T 进行交互 , 返回应答。

T : Template , 模板 , 和 MVC 中的 V 功能相同 , 产生 Html 页面



- 1、 用户点击注册按钮，将要注册的内容发送给网站的服务器。
- 2、 View 视图，接收到用户发来的注册数据，View 告诉 Model 将用户的注册信息保存进数据库。
- 3、 Model 层将用户的注册信息保存到数据库中。
- 4、 数据库将保存的结果返回给 Model
- 5、 Model 将保存的结果给 View 视图。
- 6、 View 视图告诉 Template 模板去产生一个 Html 页面。
- 7、 Template 生成 html 内容返回给 View 视图。

8、 View 将 html 页面内容返回给浏览器。

9、 浏览器拿到 view 返回的 html 页面内容进行解析，展示。

#### 4. Django 中 models 利用 ORM 对 Mysql 进行查表的语句（多个语句）？

(2018-4-14-lxy)

字段查询

all():返回模型类对应表格中的所有数据。

get():返回表格中满足条件的一条数据，如果查到多条数据，则抛异常：MultipleObjectsReturned，

查询不到数据，则抛异常：DoesNotExist。

filter():参数写查询条件，返回满足条件 QuerySet 集合数据。

条件格式：

**\*\*模型类属性名\*\*\_\_条件名=值**

注意：此处是模型类属性名，不是表中的字段名

关于 filter 具体案例如下：

判等 exact。

```
1. BookInfo.object.filter(id=1)
2. BookInfo.object.filter(id__exact=1)此处的__exact 可以省略
```

模糊查询 like

例：查询书名包含'传'的图书。contains

```
1. contains BookInfo.objects.filter(bttitle__contains=' 传' )
```

空查询 where 字段名 isnull

```
1. BookInfo.objects.filter(bttitle__isnull=False)
```

范围查询 where id in (1, 3, 5)

```
1. BookInfo.objects.filter(id__in=[1, 3, 5])
```

比较查询 gt lt(less than) gte(equal) lte

```
1. BookInfo.objects.filter(id__gte=3)
```

### 日期查询

```
1. BookInfo.objects.filter(bpub_date__year = 1980)
2. BookInfo.objects.filter(bpub_date__gt = date(1980 , 1 , 1))
```

exclude:返回不满足条件的数据。

```
3. BookInfo.objects.exclude(id=3)
```

### F 对象

作用：用于类属性之间的比较条件。

```
1. from django.db.models import F
2. 例：where bread > bcomment BookInfo.objects.filter(bread__gt =F( 'bcomment' ))
3. 例：BookInfo.objects.filter(bread__gt=F( 'bcomment' )*2)
```

### Q 对象

作用：用于查询时的逻辑条件。可以对 Q 对象进行&|~操作。

```
1. from django.db.models import Q
2. BookInfo.objects.filter(id__gt=3 , bread__gt=30)
3. BooInfo.objects.filter(Q(id__gt=3) & Q(bread__gt=3))
4. 例：BookInfo.objects.filter(Q(id__gt=3) | Q(bread__gt=30))
5. 例：BookInfo.objects.filter(~Q(id=3))
```

### order\_by 返回 QuerySet

作用：对查询结果进行排序。

```
1. 例： BookInfo.objects.all().order_by('id')
2. 例： BookInfo.objects.all().order_by('-id')
3. 例： BookInfo.objects.filter(id__gt=3).order_by('-bread')
```

### 聚合函数

作用：对查询结果进行聚合操作。

```
1. sum count max min avg
```

aggregate：调用这个函数来使用聚合。

```
1. from django.db.models import Sum , Count , Max , Min , Avg
2. 例：BookInfo.objects.aggregate(Count('id'))
```

{'id\_\_count': 5} 注意返回值类型及键名

```
1. 例：BookInfo.objects.aggregate(Sum( 'bread' ))
```

```
{ 'bread__sum' :120} 注意返回值类型及键名
```

count 函数

作用：统计满足条件数据的数目。

例：统计所有图书的数目。

```
1. BookInfo.objects.all().count()
```

例：统计 id 大于 3 的所有图书的数目。

```
1. BookInfo.objects.filter(id__gt = 3).count()
```

模型类关系

一对多关系

例：图书类-英雄类

models.ForeignKey() 定义在多的类中。

2 ) 多对多关系

例：新闻类-新闻类型类

models.ManyToManyField() 定义在哪个类中都可以。

3 ) 一对一关系

例：员工基本信息类-员工详细信息类

models.OneToOneField() 定义在哪个类中都可以。

## 5. django 中间件的使用 ? (2018-4-14-lxy)

Django 在中间件中预置了六个方法，这六个方法的区别在于不同的阶段执行，对输入或输出进行干预，方法如下：

1.初始化：无需任何参数，服务器响应第一个请求的时候调用一次，用于确定是否启用当前中间件。

```
1. def __init__():  
2.     pass
```

2.处理请求前：在每个请求上调用，返回 None 或 HttpResponse 对象。

```
1 . def process_request(request):  
2 .     pass
```

3.处理视图前：在每个请求上调用，返回 None 或 HttpResponse 对象。

```
1 . def process_view(request, view_func, view_args, view_kwargs):  
2 .     pass
```

4.处理模板响应前：在每个请求上调用，返回实现了 render 方法的响应对象。

```
1 . def process_template_response(request, response):  
2 .     pass
```

5.处理响应后：所有响应返回浏览器之前被调用，在每个请求上调用，返回 HttpResponse 对象。

```
1 . def process_response(request, response):  
2 .     pass
```

6.异常处理：当视图抛出异常时调用，在每个请求上调用，返回一个 HttpResponse 对象。

```
1 . def process_exception(request,exception):  
2 .     pass
```

## 6. 谈一下你对 uWSGI 和 nginx 的理解？(2018-4-14-lxy)

1.uWSGI 是一个 Web 服务器，它实现了 WSGI 协议、uwsgi、http 等协议。Nginx 中 HttpUwsgiModule 的作用是与 uWSGI 服务器进行交换。WSGI 是一种 Web 服务器网关接口。它是一个 Web 服务器（如 nginx，uWSGI 等服务器）与 web 应用（如用 Flask 框架写的程序）通信的一种规范。

要注意 WSGI / uwsgi / uWSGI 这三个概念的区分。

WSGI 是一种通信协议。

uwsgi 是一种线路协议而不是通信协议，在此常用于在 uWSGI 服务器与其他网络服务器的数据通信。

uWSGI 是实现了 uwsgi 和 WSGI 两种协议的 Web 服务器。

2. nginx 是一个开源的高性能的 HTTP 服务器和反向代理：

1.作为 web 服务器，它处理静态文件和索引文件效果非常高；

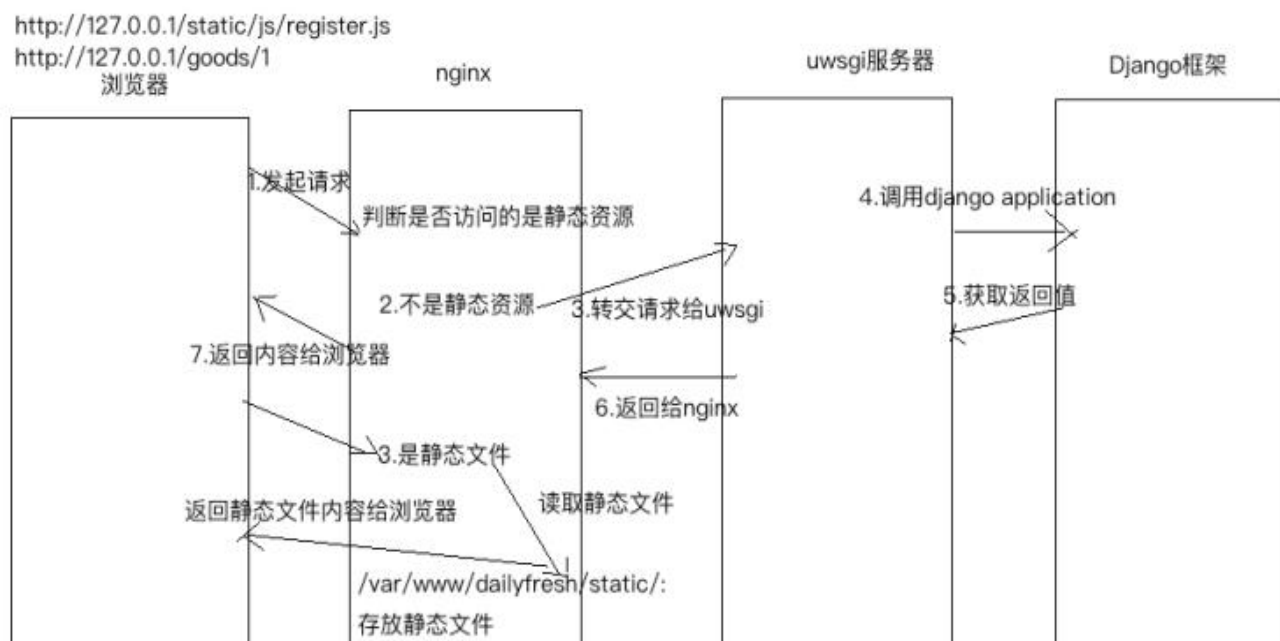


- 2.它的设计非常注重效率，最大支持 5 万个并发连接，但只占用很少的内存空间；
- 3.稳定性高，配置简洁；
- 4.强大的反向代理和负载均衡功能，平衡集群中各个服务器的负载压力应用。

## 7. 说说 nginx 和 uWSGI 服务器之间如何配合工作的？(2018-4-14-lxy)

首先浏览器发起 http 请求到 nginx 服务器，Nginx 根据接收到请求包，进行 url 分析，判断访问的资源类型，如果是静态资源，直接读取静态资源返回给浏览器，如果请求的是动态资源就转交给 uwsgi 服务器，uwsgi 服务器根据自身的 uwsgi 和 WSGI 协议，找到对应的 Django 框架，Django 框架下的应用进行逻辑处理后，将返回值发送到 uwsgi 服务器，然后 uwsgi 服务器再返回给 nginx，最后 nginx 将返回值返回给浏览器进行渲染显示给用户。

如果可以，画图讲解效果更佳，可以将下面的图画给面试官。



## 8. django 开发中数据库做过什么优化?(2018-4-14-lxy)

- 1.设计表时，尽量少使用外键，因为外键约束会影响插入和删除性能；

- 2.使用缓存，减少对数据库的访问；
- 3.在 orm 框架下设置表时，能用 varchar 确定字段长度时，就别用 text；
- 4.可以给搜索频率高的字段属性，在定义时创建索引；
- 5.Django orm 框架下的 Querysets 本来就有缓存的；
- 6.如果一个页面需要多次连接数据库，最好一次性取出所有需要的数据，减少对数据库的查询次数；
- 7.若页面只需要数据库里某一个两个字段时，可以用 QuerySet.values()；
- 8.在模板标签里使用 with 标签可以缓存 Qset 的查询结果。

## 9. 验证码过期时间怎么设置？(2018-4-14-lxy)

将验证码保存到数据库或 session，设置过期时间为 1 分钟，然后页面设置一个倒计时(一般是前端 js 实现 这个计时)的展示，一分钟过后再次点击获取新的信息。

## 10. Python 中三大框架各自的应用场景？(2018-4-14-lxy)

django：主要是用来搞快速开发的，他的亮点就是快速开发，节约成本，正常的并发量不过 10000，如果要想实现高并发的话，就要对 django 进行二次开发，比如把整个笨重的框架给拆掉，自己写 socket 实现 http 的通信，底层用纯 c，c++ 写提升效率，ORM 框架给干掉，自己编写封装与数据库交互的框架，因为啥呢，ORM 虽然面向对象来操作数据库，但是它的效率很低，使用外键来联系表与表之间的查询；

flask：轻量级，主要是用来写接口的一个框架，实现前后端分离，提升开发效率，Flask 本身相当于一个内核，其他几乎所有的功能都要用到扩展（邮件扩展 Flask-Mail，用户认证 Flask-Login），都需要用第三方的扩展来实现。比如可以用 Flask-extension 加入 ORM、窗体验证工具，文件上传、身份验证等。Flask 没有默认使用的数据库，你可以选择 MySQL，也可以用 NoSQL。

其 WSGI 工具箱采用 Werkzeug (路由模块)，模板引擎则使用 Jinja2。这两个也是 Flask 框架的核心。Python 最出名的框架要数 Django，此外还有 Flask、Tornado 等框架。虽然 Flask 不是最出名的框架，但是 Flask 应该算是最灵活的框架之一，这也是 Flask 受到广大开发者喜爱的原因。

Tornado：Tornado 是一种 Web 服务器软件的开源版本。Tornado 和现在的主流 Web 服务器框架（包括大多数 Python 的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。

得益于其非阻塞的方式和对 epoll 的运用，Tornado 每秒可以处理数以千计的连接，因此 Tornado 是实时 Web 服务的一个理想框架。

## 11.django 如何提升性能（高并发）？(2018-4-14-lxy)

对一个后端开发工程师来说，提升性能指标主要有两个一个是并发数，另一个是响应时间网站性能的优化一般包括 web 前端性能优化，应用服务器性能优化，存储服务器优化。

对前端的优化主要有：

1.减少 http 请求，减少数据库的访问量，比如使用雪碧图。

2.使用浏览器缓存，将一些常用的 css，js，logo 图标，这些静态资源缓存到本地浏览器，通过设置 http 头中的 cache-control 和 expires 的属性，可设定浏览器缓存，缓存时间可以自定义。

3 对 html，css，javascript 文件进行压缩，减少网络的通信量。

对我个人而言，我做的优化主要是以下三个方面：

1.合理的使用缓存技术，对一些常用到的动态数据，比如首页做一个缓存，或者某些常用的数据做个缓存，设置一定得过期时间，这样减少了对数据库的压力，提升网站性能。

2.使用 celery 消息队列，将耗时的操作扔到队列里，让 worker 去监听队列里的任务，实现异步操作，比如发邮件，发短信。

3.就是代码上的一些优化，补充：nginx 部署项目也是项目优化，可以配置合适的配置参数，提升效率，增加并发量。

4.如果太多考虑安全因素，服务器磁盘用固态硬盘读写，远远大于机械硬盘，这个技术现在没有普及，主要是固态硬盘技术上还不是完全成熟，相信以后会大量普及。

5.另外还可以搭建服务器集群，将并发访问请求，分散到多台服务器上处理。

6.最后就是运维工作人员的一些性能优化技术了。

## 12.什么是 restful api，谈谈你的理解?(2018-4-14-lxy)

REST:Representational State Transfer 的缩写，翻译：“具象状态传输”。一般解释为“表现层状态转换”。

REST 是设计风格而不是标准。是指客户端和服务器的交互形式。我们需要关注的重点是如何设计 REST 风格的网络接口。

REST 的特点：

1.具象的。一般指表现层，要表现的对象就是资源。比如，客户端访问服务器，获取的数据就是资源。比如文字、图片、音视频等。

2.表现：资源的表现形式。txt 格式、html 格式、json 格式、jpg 格式等。浏览器通过 URL 确定资源的位置，但是需要在 HTTP 请求头中，用 Accept 和 Content-Type 字段指定，这两个字段是对资源表现的描述。

3.状态转换：客户端和服务器的交互过程。在这个过程中，一定会有数据和状态的转化，这种转化叫做状态转换。其中，GET 表示获取资源，POST 表示新建资源，PUT 表示更新资源，DELETE 表示删除资源。HTTP 协议中最常用的就是这四种操作方式。

RESTful 架构：

- 1.每个 URL 代表一种资源；
- 2.客户端和服务端之间，传递这种资源的某种表现层；
- 3.客户端通过四个 http 动词，对服务器资源进行操作，实现表现层状态转换。

## 12.1 如何设计符合 RESTful 风格的 API

### 一、域名：

将 api 部署在专用域名下：

`http://api.example.com`

或者将 api 放在主域名下：

`http://www.example.com/api/`

### 二、版本：

将 API 的版本号放在 url 中。

`http://www.example.com/app/1.0/info`

`http://www.example.com/app/1.2/info`

### 三、路径：

路径表示 API 的具体网址。每个网址代表一种资源。资源作为网址，网址中不能有动词只能有名词，一般名词要与数据库的表名对应。而且名词要使用复数。

错误示例：

`http://www.example.com/getGoods`

`http://www.example.com/listOrders`

正确示例：

`#获取单个商品`

`http://www.example.com/app/goods/1`

`#获取所有商品`

`http://www.example.com/app/goods`

#### 四、使用标准的 HTTP 方法：

对于资源的具体操作类型，由 HTTP 动词表示。常用的 HTTP 动词有四个。

GET      SELECT ：从服务器获取资源。

POST     CREATE ：在服务器新建资源。

PUT      UPDATE ：在服务器更新资源。

DELETE   DELETE ：从服务器删除资源。

示例：

`#获取指定商品的信息`

`GET http://www.example.com/goods/ID`

`#新建商品的信息`

`POST http://www.example.com/goods`

`#更新指定商品的信息`

`PUT http://www.example.com/goods/ID`

`#删除指定商品的信息`

`DELETE http://www.example.com/goods/ID`

#### 五、过滤信息：

如果资源数据较多，服务器不能将所有数据一次全部返回给客户端。API 应该提供参数，过滤返回结果。 实例：

`#指定返回数据的数量`

```
http://www.example.com/goods?limit=10
```

#指定返回数据的开始位置

```
http://www.example.com/goods?offset=10
```

#指定第几页，以及每页数据的数量

```
http://www.example.com/goods?page=2&per_page=20
```

## 六、状态码：

服务器向用户返回的状态码和提示信息，常用的有：

200 OK ：服务器成功返回用户请求的数据

201 CREATED ：用户新建或修改数据成功。

202 Accepted ：表示请求已进入后台排队。

400 INVALID REQUEST ：用户发出的请求有错误。

401 Unauthorized ：用户没有权限。

403 Forbidden ：访问被禁止。

404 NOT FOUND ：请求针对的是不存在的记录。

406 Not Acceptable ：用户请求的的格式不正确。

500 INTERNAL SERVER ERROR ：服务器发生错误。

## 七、错误信息：

一般来说，服务器返回的错误信息，以键值对的形式返回。

```
{  
    error: 'Invalid API KEY'  
}
```

## 八、响应结果：



针对不同结果，服务器向客户端返回的结果应符合以下规范。

#返回商品列表

GET `http://www.example.com/goods`

#返回单个商品

GET `http://www.example.com/goods/cup`

#返回新生成的商品

POST `http://www.example.com/goods`

#返回一个空文档

DELETE `http://www.example.com/goods`

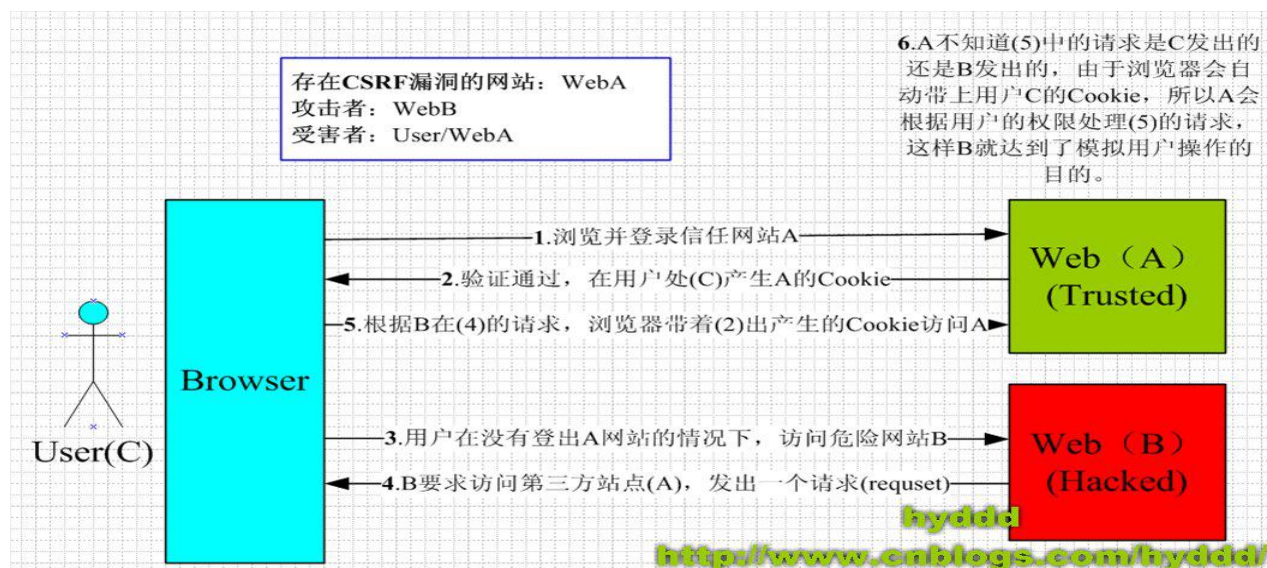
#### 九、使用链接关联相关的资源：

在返回响应结果时提供链接其他 API 的方法，使客户端很方便的获取相关联的信息。

#### 十、其他：

服务器返回的数据格式，应该尽量使用 JSON，避免使用 XML。

### 13. 什么 csrf 攻击原理？如何解决？(2018-4-14-lxy)





简单来说就是：你访问了信任网站 A，然后 A 会用保存你的个人信息并返回给你的浏览器一个 cookie，然后呢，在 cookie 的过期时间之内，你去访问了恶意网站 B，它给你返回一些恶意请求代码，要求你去访问网站 A，而你的浏览器在收到这个恶意请求之后，在你不知情的情况下，会带上保存在本地浏览器的 cookie 信息去访问网站 A，然后网站 A 误以为是用户本身的操作，导致来自恶意网站 C 的攻击代码会被执行：发邮件，发消息，修改你的密码，购物，转账，偷窥你的个人信息，导致私人信息泄漏和账户财产安全收到威胁

#### 14. 启动 Django 服务的方法？(2018-4-14-lxy)

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。

#### 15. 怎样测试 django 框架中的代码？(2018-4-14-lxy)

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：

1. python manage.py test：执行所有的测试用例
2. python manage.py test app\_name，执行该 app 的所有测试用例
3. 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



16.有过部署经验？用的什么技术？可以满足多少压力？（2018-4-14-lxy）

- 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）

17.Django 中哪里用到了线程？哪里用到了协程？哪里用到了进程？（2018-4-14-lxy）

- 1.Django 中耗时的任务用一个进程或者线程来执行，比如发邮件，使用 celery。
- 2.部署 django 项目的时候，配置文件中设置了进程和协程的相关配置。

18.django 关闭浏览器，怎样清除 cookies 和 session？（2018-4-14-lxy）

#### 设置 Cookie

```
1. def cookie_set(request):
2.     response = HttpResponse("<h1>设置 Cookie，请查看响应报文头</h1>")
3.     response.set_cookie('h1', 'hello django')
4.     return response
```

#### 读取 Cookie

```
1. def cookie_get(request):
```

```
2. response = HttpResponse("读取 Cookie，数据如下：<br>")
3. if request.COOKIE.has_key('h1'):
4.     response.write('<h1>' + request.COOKIE['h1'] + '</h1>')
5. return response
```

以键值对的格式写会话。

```
1. request.session['键']=值
```

根据键读取值。

```
1. request.session.get('键',默认值)
```

清除所有会话，在存储中删除值部分。

```
1. request.session.clear()
```

清除会话数据，在存储中删除会话的整条数据。

```
1. request.session.flush()
```

删除会话中的指定键及值，在存储中只删除某个键及对应的值。

```
1. del request.session['键']
```

设置会话的超时时间，如果没有指定过期时间则两个星期后过期。

如果 value 是一个整数，会话将在 value 秒没有活动后过期。

如果 value 为 0，那么用户会话的 Cookie 将在用户的浏览器关闭时过期。

如果 value 为 None，那么会话永不过期。

```
1. request.session.set_expiry(value)
```

Session 依赖于 Cookie，如果浏览器不能保存 cookie 那么 session 就失效了。因为它需要浏览器的 cookie 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。

## 19.代码优化从哪些方面考虑？有什么想法？（2018-4-14-lxy）

### 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。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)。

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。

## 20.Django 中间件是如何使用的? (2018-4-14-lxy)

中间件不用继承自任何类(可以继承 object),下面一个中间件大概的样子:

```
1. class CommonMiddleware(object):
2.     def process_request(self, request):
3.         return None
4.
5.     def process_response(self, request, response):
6.         return response
```

还有 process\_view , process\_exception 和 process\_template\_response 函数。

1)初始化:无需任何参数,服务器响应第一个请求的时候调用一次,用于确定是否启用当前中间件。

```
1. def __init__(self):
2.     pass
```

2)处理请求前:在每个请求上,request 对象产生之后,url 匹配之前调用,返回 None 或

HttpResponse 对象。

```
1. def process_request(self, request):
2.     pass
```

3)处理视图前:在每个请求上,url 匹配之后,视图函数调用之前调用,返回 None 或

HttpResponse 对象。

```
1. def process_view(self, request, view_func, *view_args, **view_kwargs):
2.     pass
```

4) 处理响应后：视图函数调用之后，所有响应返回浏览器之前被调用，在每个请求上调用，返回 `HttpResponse` 对象。

```
1. def process_response(self, request, response):  
2.     pass
```

5) 异常处理：当视图抛出异常时调用，在每个请求上调用，返回一个 `HttpResponse` 对象。

```
1. def process_exception(self, request, exception):  
2.     pass
```

## 21.有用过 Django REST framework 吗？(2018-4-14-lxy)

Django REST framework 是一个强大而灵活的 Web API 工具。使用 RESTframework 的理由有：

Web browsable API 对开发者有极大的好处

包括 OAuth1a 和 OAuth2 的认证策略

支持 ORM 和非 ORM 数据资源的序列化

全程自定义开发——如果不想使用更加强大的功能，可仅仅使用常规的 function-based views

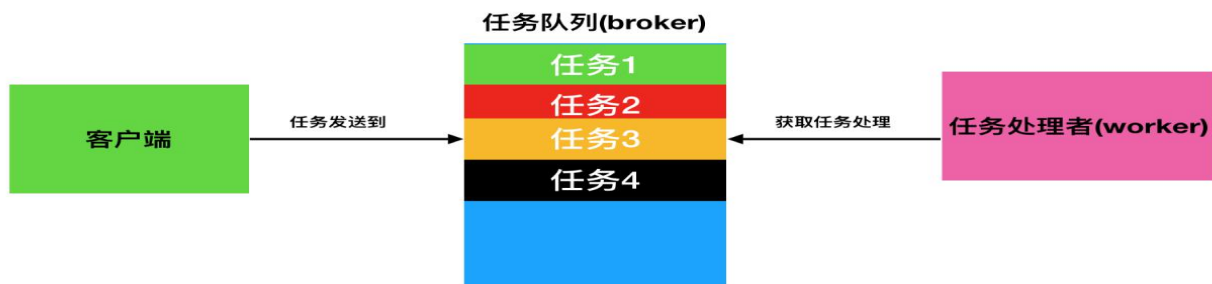
额外的文档和强大的社区支持

## 22.Celery 分布式任务队列？(2018-4-14-lxy)

情景：用户发起 request，并等待 response 返回。在本些 views 中，可能需要执行一段耗时的程序，那么用户就会等待很长时间，造成不好的用户体验，比如发送邮件、手机验证码等。

使用 celery 后，情况就不一样了。解决：将耗时的程序放到 celery 中执行。

将多个耗时的任务添加到队列 queue 中，也就是用 redis 实现 broker 中间人，然后用多个 worker 去监听队列里的任务去执行。



- 任务 task：就是一个 Python 函数。
- 队列 queue：将需要执行的任务加入到队列中。
- 工人 worker：在一个新进程中，负责执行队列中的任务。
- 代理人 broker：负责调度，在布置环境中使用 redis。

## 23.Jieba 分词 (2018-4-14-lxy)

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。

## 24.nginx 的正向代理与反向代理? (2018-4-14-lxy)

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。

## 25.简述 Django 下的（内建的）缓存机制？（2018-4-14-lxy）

一个动态网站的基本权衡点就是，它是动态的。每次用户请求页面，服务器会重新计算。从开销处理的角度来看，这比你读取一个现成的标准文件的代价要昂贵的多。

这就是需要缓存的地方。

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 HTTPheaders)关于网站哪些部分需要缓存和如何缓存。

设置缓存：

缓存系统需要一些设置才能使用。也就是说，你必须告诉他你要把数据缓存在哪里- 是数据库中，文件系统或者直接在内存中。这个决定很重要，因为它会影响你的缓存性能，是的，一些缓存类型要比其他的缓存类型更快速。

你的缓存配置是通过 setting 文件的 CACHES 配置来实现的。这里有 CACHES 所有可配置的变量值。



26.请简述浏览器是如何获取一枚网页的？（2018-4-14-lxy）

- 1.在用户输入目的 URL 后，浏览器先向 DNS 服务器发起域名解析请求；
- 2.在获取了对应的 IP 后向服务器发送请求数据包；
- 3.服务器接收到请求数据后查询服务器上对应的页面，并将找到的页面代码回复给客户端；
- 4.客户端接收到页面源代码后，检查页面代码中引用的其他资源，并再次向服务器请求该资源；
- 5.在资源接收完成后，客户端浏览器按照页面代码将页面渲染输出显示在显示器上；

27.对 cookie 与 session 的了解？他们能单独用吗？（2018-4-14-lxy）

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 SessionID，也就得不到 Session。

## 28.Django HTTP 请求的处理流程? (2018-4-14-lxy)

Django 和其他 Web 框架的 HTTP 处理的流程大致相同，Django 处理一个 Request 的过程是首先通过中间件，然后再通过默认的 URL 方式进行的。我们可以在 Middleware 这个地方把所有 Request 拦截住，用我们自己的方式完成处理以后直接返回 Response。

### 1. 加载配置

Django 的配置都在 “Project/settings.py” 中定义，可以是 Django 的配置，也可以是自定义的配置，并且都通过 `django.conf.settings` 访问，非常方便。

### 2. 启动

最核心动作的是通过 `django.core.management.commands.runfcgi` 的 Command 来启动，它运行 `django.core.servers.fastcgi` 中的 `runfastcgi`，`runfastcgi` 使用了 `flup` 的 `WSGIServer` 来启动 `fastcgi`。而 `WSGIServer` 中携带了 `django.core.handlers.wsgi` 的 `WSGIHandler` 类的一个实例，通过 `WSGIHandler` 来处理由 Web 服务器（比如 Apache，Lighttpd 等）传过来的请求，此时才是真正进入 Django 的世界。

### 3. 处理 Request

当有 HTTP 请求来时，`WSGIHandler` 就开始工作了，它从 `BaseHandler` 继承而来。`WSGIHandler` 为每个请求创建一个 `WSGIRequest` 实例，而 `WSGIRequest` 是从 `http.HttpRequest` 继承而来。接下来就开始创建 `Response` 了。

### 4. 创建 Response

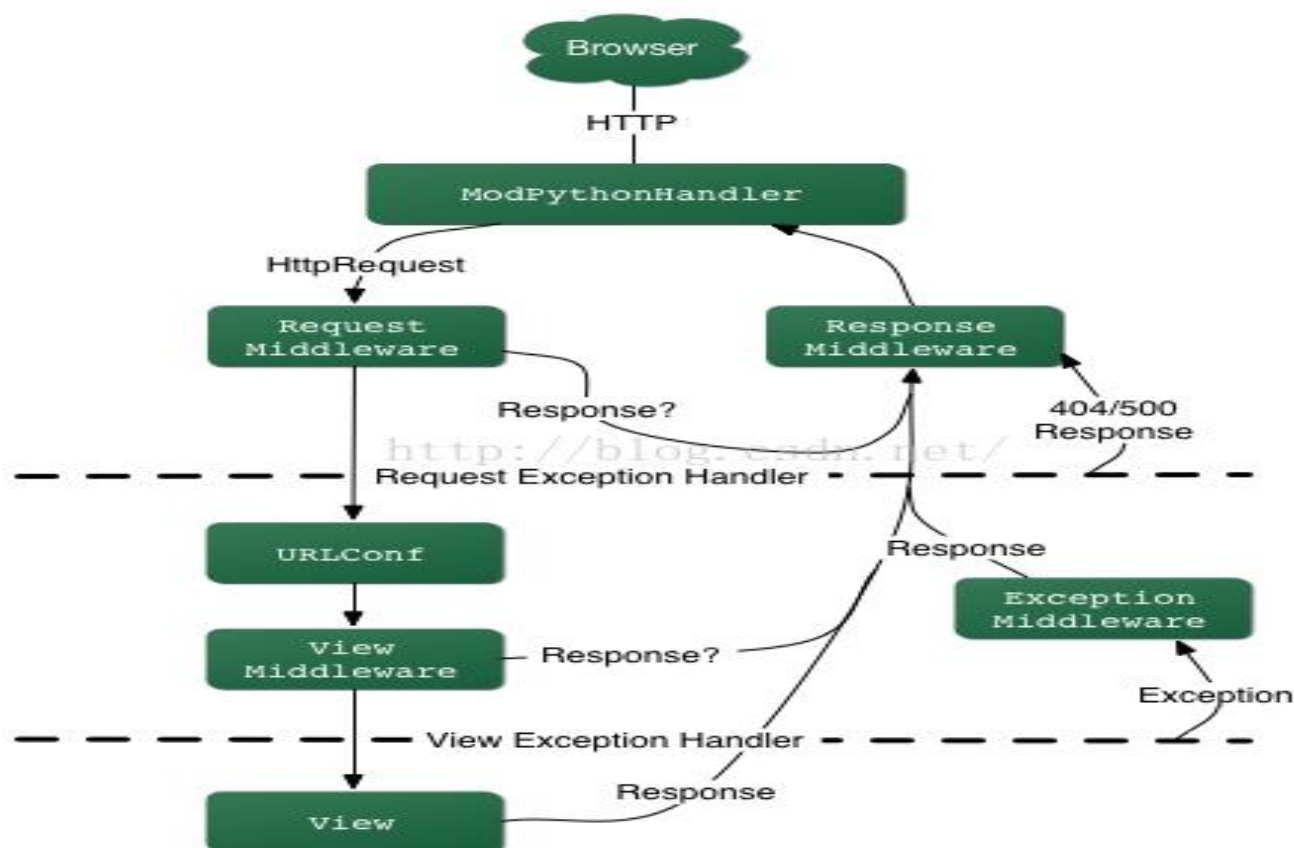
`BaseHandler` 的 `get_response` 方法就是根据 request 创建 response，而具体生成 response 的动作就是执行 `urls.py` 中对应的 view 函数了，这也是 Django 可以处理“友好 URL”的关键步骤，每个这样的函数都要返回一个 `Response` 实例。此时一般的做法是通过 loader 加载

template 并生成页面内容，其中重要的就是通过 ORM 技术从数据库中取出数据，并渲染到 Template 中，从而生成具体的页面了。

## 5. 处理 Response

Django 返回 Response 给 flup，flup 就取出 Response 的内容返回给 Web 服务器，由后者返回给浏览器。

总之，Django 在 fastcgi 中主要做了两件事：处理 Request 和创建 Response，而它们对应的核心就是“urls 分析”、“模板技术”和“ORM 技术”。



如图所示，一个 HTTP 请求，首先被转化成一个 HttpRequest 对象，然后该对象被传递给 Request 中间件处理，如果该中间件返回了 Response，则直接传递给 Response 中间件做收尾处理。否则的话 Request 中间件将访问 URL 配置，确定哪个 view 来处理，在确定了哪个 view 要执行，

但是还没有执行该 view 的时候，系统会把 request 传递给 view 中间件处理器进行处理，如果该中间件返回了 Response，那么该 Response 直接被传递给 Response 中间件进行后续处理，否则将执行确定的 view 函数处理并返回 Response，在这个过程中如果引发了异常并抛出，会被 Exception 中间件处理器进行处理。

## 29.Django 里 QuerySet 的 get 和 filter 方法的区别？（2018-4-14-lxy）

### 1) 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。

### 2) 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）。

### 3) 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

30.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 ? (2018-4-16-lxy)

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 ,那么之前的登录状态就没有了。

31.跨域请求问题 django 怎么解决的 ( 原理 ) (2018-4-16-lxy)

- 启用中间件
- post 请求
- 验证码
- 表单中添加 csrf\_token 标签

32.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? (2018-4-16-lxy)

- 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)

33.Django 重定向你是如何实现的 ? 用的什么状态码 ? (2018-4-16-lxy)

- 使用 HttpResponseRedirect
- redirect 和 reverse
- 状态码 : 302,301

### 34.生成迁移文件和执行迁移文件的命令是什么？(2018-4-16-lxy)

```
python manage.py makemigrations
```

```
python manage.py migrate
```

### 35.关系型数据库的关系包括哪些类型？(2018-4-16-lxy)

- 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。

### 36.查询集返回列表的过滤器有哪些？(2018-4-16-lxy)

- all()：返回所有的数据
- 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

### 37.判断查询集正是否有数据？(2018-4-16-lxy)

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。

### 38.Django 本身提供了 runserver，为什么不能用来部署？(2018-4-16-lxy)

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。

### 39.apache 和 nginx 的区别？(2018-4-16-lxy)

Nginx 相对 Apache 的优点：

- 轻量级，同样起 web 服务，比 apache 占用更少的内存及资源；

- 抗并发，nginx 处理请求是异步非阻塞的，支持更多的并发连接，而 apache 则是阻塞型的，在高并发下 nginx 能保持低资源低消耗高性能；

- 配置简洁；

- 高度模块化的设计，编写模块相对简单；

- 社区活跃。

Apache 相对 Nginx 的优点：

- rewrite，比 nginx 的 rewrite 强大；

- 模块超多，基本想到的都可以找到；

- 少 bug，nginx 的 bug 相对较多；

- 超稳定。



#### 40.varchar 与 char 的区别？(2018-4-16-lxy)

char 长度是固定的，不管你存储的数据是多少他都会都固定的长度。而 varchar 则处可变长度但他要在总长度上加 1 字符，这个用来存储位置。所以在处理速度上 char 要比 varchar 快速很多，但是对费存储空间，所以对存储不大，但在速度上有要求的可以使用 char 类型，反之可以用 varchar 类型。

#### 41.查询集两大特性？惰性执行？(2018-4-23-lxy)

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

#### 42.git 常用命令？(2018-4-23-lxy)

git clone 克隆指定仓库

git status 查看当前仓库状态

git diff 比较版本的区别

git log 查看 git 操作日志

git reset 回溯历史版本

git add 将文件添加到暂存区

git commit 将文件提交到服务器

git checkout 切换到指定分支

git rm 删除指定文件

### 43.电商网站库存问题(2018-5-1-lyf)

一般团购，秒杀，特价之类的活动，这样会使访问量激增，很多人抢购一个商品，作为活动商品，库存肯定是很有限的。控制库存问题，数据库的事务功能是控制库存超卖的有效方式。

1.在秒杀的情况下，肯定不能如此频率的去读写数据库，严重影响性能问题，必须使用缓存，将需要秒杀的商品放入缓存中，并使用锁来处理并发情况，先将商品数量增减（加锁、解析）后在进行其他方面的处理，处理失败再将数据递增（加锁、解析），否则表示交易成功。

2.这个肯定不能直接操作数据库的，会挂的。直接读库写库对数据库压力太大了，要用到缓存。

3.首先，多用户并发修改同一条记录时，肯定是后提交的用户将覆盖掉前者提交的结果了。这个直接可以使用加乐观锁的机制去解决高并发的的问题。

### 44.HttpRequest 和 HttpResponse 是什么?干嘛用的？（2018-5-2-xhq）

HttpRequest 是 django 接受用户发送多来的请求报文后，将报文封装到 HttpRequest 对象中去。

HttpResponse 返回的是一个应答的数据报文。render 内部已经封装好了 HttpResponse 类。

视图的第一个参数必须是 HttpRequest 对象，两点原因：表面上说，他是处理 web 请求的，所以必须是请求对象，根本上说，他是基于请求的一种 web 框架，所以，必须是请求对象。

因为 view 处理的是一个 request 对象，请求的所有属性我们都可以根据对象属性的查看方法来获取具体的信息：格式：request.属性

request.path 请求页面的路径，不包含域名

request.get\_full\_path 获取带参数的路径

request.method 页面的请求方式

request.GET GET 请求方式的数据

request.POST POST 请求方式的数据

request.COOKIES 获取 cookie

request.session 获取 session

request.FILES 上传图片（请求页面有 enctype="multipart/form-data" 属性时 FILES 才有数据。

? a=10 的键和值时怎么产生的，键是开发人员在编写代码时确定下来的，值时根据数据生成或者用户填写的，总之是不确定的。

403 错误：表示资源不可用，服务器理解客户的请求，但是拒绝处理它，通常由于服务器上文件和目录的权限设置导致的 web 访问错误。如何解决：1、把中间件注释。2、在表单内部添加{% csrf\_token %}

request.GET.get()取值时如果一键多值情况，get 是覆盖的方式获取的。getlist ( ) 可以获取多个值。

在一个有键无值的情况下，该键名 c 的值返回空。有键无值：c: getlist 返回的是列表，空列表

在无键无值也没有默认值的情况下，返回的是 None 无键无值：e:None

HttpResponse 常见属性：

content：表示返回的内容

charset: 表示 response 采用的编码字符集，默认是 utf-8

status\_code:返回的 HTTP 响应状态码 3XX 是对请求继续进一步处理，常见的是重定向。

常见方法：

init:创建 httpResponse 对象完成返回内容的初始化

set\_cookie：设置 Cookie 信息：格式：set\_cookies('key','value',max\_age=None,expires=None)

max\_age 是一个整数，表示指定秒数后过期，expires 指定过期时间，默认两个星期后过期。

write 向响应体中写数据

应答对象：

方式一：render(request,"index.html") 返回一个模板

render(request,"index.html", context) 返回一个携带动态数据的页面

方式二：`render_to_response("index.html")` 返回一个模板页面

方式三：`redirect("/")` 重定向

方式四：`HttpResponseRdirect("/")` 实现页面跳转功能

方式五：`HttpResponse ( "itcast1.0")`在返回到额页面中添加字符串内容

方式六：`HttpResponseJson()` 返回的页面中添加字符串内容。

`JsonResponse` 创建对象时候接收字典作为参数，返回的对象是一个 json 对象。

能接收 Json 格式数据的场景，都需要使用 view 的 `JsonResponse` 对象返回一个 json 格式数据

ajax 的使用场景，页面局部刷新功能。ajax 接收 Json 格式的数据。

在返回的应答报文中，可以看到 `JsonResponse` 应答的 `content-Type` 内容是 `application/json`

ajax 实现网页局部刷新功能：ajax 的 `get()`方法获取请求数据    ajax 的 `each()`方法遍历输出这些数据

## 45.什么是反向解析 ( 2018-5-2-xhq )

使用场景：模板中的超链接，视图中的重定向

使用：在定义 url 时为 include 定义 namespace 属性，为 url 定义 name 属性

在模板中使用 url 标签：`{% url 'namespace_value:name_value'%}`

在视图中使用 reverse 函数：`redirect(reverse('namespce_value:name_value' ))`

根据正则表达式动态生成地址，减轻后期维护成本。

注意反向解析传参数，主要是在我们的反向解析的规则后面天界了两个参数，两个参数之间使用空格隔

开：`<a href="{% url 'booktest:fan2' 2 3 %}">位置参数</a>`

## 46.Django 日志管理： ( 2018-5-2-xhq )

配置好之后：

```
import logging
```

```
logger=logging.getLogger(__name__) # 为 loggers 中定义的名称
```

```
logger.info("some info ...")
```

可用函数有：logger.debug()    logger.info()    logger.warning()    logger.error()

Django 文件管理：对于 Django 来说，项目中的 css，js，图片都属于静态文件，我们一般会将静态文件放到一个单独的目录中，以方便管理，在 html 页面调用时，也需要指定静态文件的路径。静态文件可以放在项目根目录下，也可以放在应用的目录下，由于这些静态文件在项目中是通用的，所以推荐放在项目的根目录下。

在生产中，只要和静态文件相关的，所有访问，基本上没有 Django 什么事，一般都是由 Nginx 软件代劳了，为什么？因为 Nginx 就是干这个的。

### 三 . Tornado

#### 1. Tornado 的核是什么？(2018-4-16-lxy)

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。

## 第六章 爬虫

### 一．常用库与模块

1. 试列出至少三种目前流行的大型数据库的名称:\_\_\_\_、\_\_\_\_、\_\_\_\_, 其中您最熟悉的是\_\_\_\_,从\_\_\_\_年开始使用 ( 2018-4-1-ydy )

( 考察对数据库的熟悉程度,同时考察你的工作年限注意和自己简历一致 )。Oracle , Mysql , SQLServer、MongoDB 根据自己情况 ( 推荐 Mysql 、 MongoDB )。

2. 列举您使用过的 Python 网络爬虫所用到的网络数据包?(2018-4-16-lxy)

requests、urllib、urllib2、httplib2。

3. 列举您使用过的 Python 网络爬虫所用到的解析数据包 ( 2018-4-1-ydy )

BeautifulSoup、pyquery、Xpath、lxml。

4. 爬取数据后使用哪个数据库存储数据的,为什么? ( 2018-4-1-ydy )

MongoDB 是使用比较多的数据库,这里以 MongoDB 为例,大家需要结合自己真实开发环境回答。

原因:1) 与关系型数据库相比,MongoDB 的优点如下。

①弱一致性(最终一致),更能保证用户的访问速度

举例来说,在传统的关系型数据库中,一个 COUNT 类型的操作会锁定数据集,这样可以保证得到“当前”情况下的较精确值。这在某些情况下,例如通过 ATM 查看账户信息的时候很重要,但对于 Wordnik 来说,数据是不断更新和增长的,这种“较精确”的保证几乎没有任何意义,反而

会产生很大的延迟。他们需要的是一个“大约”的数字以及更快的处理速度。

但某些情况下 MongoDB 会锁住数据库。如果此时正有数百个请求，则它们会堆积起来，造成许多问题。我们使用了下面的优化方式来避免锁定。

每次更新前，我们会先查询记录。查询操作会将对象放入内存，于是更新则会尽可能的迅速。在主/从部署方案中，从节点可以使用“-pretouch”参数运行，这也可以得到相同的效果。

使用多个 mongod 进程。我们根据访问模式将数据库拆分成多个进程。

②文档结构的存储方式，能够更便捷的获取数据。

对于一个层级式的数据结构来说，如果要将这样的数据使用扁平式的，表状的结构来保存数据，这无论是在查询还是获取数据时都十分困难。

③内置 GridFS，支持大容量的存储。

GridFS 是一个出色的分布式文件系统，可以支持海量的数据存储。内置了 GridFS 了 MongoDB，能够满足对大数据集的快速范围查询。

④内置 Sharding。

提供基于 Range 的 Auto Sharding 机制：一个 collection 可按照记录的范围，分成若干个段，切分到不同的 Shard 上。Shards 可以和复制结合，配合 Replica sets 能够实现 Sharding+fail-over，不同的 Shard 之间可以负载均衡。查询是对客户端是透明的。客户端执行查询，统计，MapReduce 等操作，这些会被 MongoDB 自动路由到后端的数据节点。这让我们关注于自己的业务，适当的时候可以无痛的升级。MongoDB 的 Sharding 设计能力较大可支持约 20 petabytes，足以支撑一般应用。

这可以保证 MongoDB 运行在便宜的 PC 服务器集群上。PC 集群扩充起来非常方便并且成本很低，避免了“sharding”操作的复杂性和成本。

⑤第三方支持丰富。(这是与其他的 NoSQL 相比，MongoDB 也具有的优势)

现在网络上的很多 NoSQL 开源数据库完全属于社区型的，没有官方支持，给使用者带来了很大的风险。而开源文档数据库 MongoDB 背后有商业公司 10gen 为其提供商业培训和支持。

而且 MongoDB 社区非常活跃，很多开发框架都迅速提供了对 MongoDB 的支持。不少知名大公司和网站也在生产环境中使用 MongoDB，越来越多的创新型企业转而使用 MongoDB 作为和 Django，RoR 来搭配的技术方案。

### ⑥性能优越

在使用场合下，千万级别的文档对象，近 10G 的数据，对有索引的 ID 的查询不会比 mysql 慢，而对非索引字段的查询，则是全面胜出。mysql 实际无法胜任大数据量下任意字段的查询，而 mongodb 的查询性能实在让我惊讶。写入性能同样很令人满意，同样写入百万级别的数据，mongodb 比我以前试用过的 couchdb 要快得多，基本 10 分钟以下可以解决。补上一句，观察过程中 mongodb 都远算不上是 CPU 杀手。

## 2)Mongoedb 与 redis 相比较

①mongodb 文件存储是 BSON 格式类似 JSON，或自定义的二进制格式。

mongodb 与 redis 性能都很依赖内存的大小，mongodb 有丰富的数据表达、索引；最类似于关系数据库，支持丰富的查询语言，redis 数据丰富，较少的 IO，这方面 mongodb 优势明显。

②mongodb 不支持事物，靠客户端自身保证，redis 支持事物，比较弱，仅能保证事物中的操作按顺序执行，这方面 redis 优于 mongodb。

③mongodb 对海量数据的访问效率提升，redis 较小数据量的性能及运算，这方面 mongodb 性能优于 redis。mongodb 有 mapreduce 功能，提供数据分析，redis 没有，这方面 mongodb 优于 redis。



## 5. 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？

(2018-4-16-lxy)

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，它包含了下载器，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，并发及分布式处理方面，不够灵活，不便调整与拓展。

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；

Scrapy 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。

## 6. 写爬虫是用多进程好？还是多线程好？为什么？(2018-4-16-lxy)

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. 常见的反爬虫和应对方法？(2018-4-16-lxy)

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。

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，例如同一个 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。

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。

## 8. 解析网页的解析器使用最多的是哪几个? (2018-4-16-lxy)

lxml , html5lib , html.parser,lxml-xml , 正则表达式。

## 9. 需要登录的网页，如何解决同时限制 ip , cookie,session ( 其中有一些是动态生成的 ) 在不使用动态爬取的情况下 ? (2018-4-16-lxy)

解决限制 IP 可以使用代理 IP 地址池、服务器；

不适用动态爬取的情况下可以使用反编译 JS 文件获取相应的文件 ,或者换用其他平台( 比如手机端 ) 看看是否可以获取相应的 json 文件。

## 10. 验证码的解决? (2018-4-16-lxy)

图形验证码：干扰、杂色不是特别多的图片可以使用开源库 Tesseract 进行识别，太过复杂的需要借助第三方打码平台。

点击和拖动滑块验证码可以借助 selenium、无图形界面浏览器 ( chromedriver 或者 phantomjs ) 和 pillow 包来模拟人的点击和滑动操作，pillow 可以根据色差识别需要滑动的位置。

## 11. 使用最多的数据库 ( Mysql , Mongodb , redis 等 ) , 对他们的理解 ? (2018-4-16-lxy)

MySQL 数据库：开源免费的关系型数据库，需要实现创建数据库、数据表和表的字段，表与表之间可以进行关联（一对多、多对多），是持久化存储。

Mongodb 数据库：是非关系型数据库，数据库的三元素是，数据库、集合、文档，可以进行持久化存储，也可作为内存数据库，存储数据不需要事先设定格式，数据以键值对的形式存储。

redis 数据库：非关系型数据库，使用前可以不用设置格式，以键值对的方式保存，文件格式相对自由，主要用与缓存数据库，也可以进行持久化存储。

## 12. 字符集和字符编码(2018-4-23-lyf)

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）如下所示：

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。

13. 写一个邮箱地址的正则表达式？(2018-4-23-lyf)

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

14. 编写过哪些爬虫中间件？(2018-4-23-lyf)

user-agent、代理池等。

15. “极验” 滑动验证码如何破解？(2018-4-23-lyf)

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。

## 16. 爬的那些内容数据量有多大，多久爬一次，爬下来的数据是怎么存储？

( 2018-4-20-xhq )

京东整站的数据大约在 1 亿左右，爬下来的数据存入数据库，mysql 数据库中如果有重复的 url 建议去重存入数据库，可以考虑引用外键。评分，评论如果做增量，Redis 中 url 去重，评分和评论建议建立一张新表用 id 做关联。

多久爬一次这个问题要根据公司的要求去处理，不一定是每天都爬。

Mongo 建立唯一索引键 ( id ) 可以做数据重复 前提是数据量不大 2 台电脑几百万的情况 数据库需要做分片 ( 数据库要设计合理 )。

例：租房的网站数据量每天大概是几十万条，每周固定爬取。

## 17. cookie 过期的处理问题？(2018-4-20-xhq)

因为 cookie 存在过期的现象，一个很好的处理方法就是做一个异常类，如果有异常的话 cookie 抛出异常类在执行程序。

## 18. 动态加载又对及时性要求很高怎么处理？(2018-4-20-xhq)

Selenium+Phantomjs

尽量不使用 sleep 而使用 WebDriverWait

关于 HTTP/HTTPS 的区别，分别应该在什么场合下。

## 19. HTTPS 有什么优点和缺点(2018-4-20-xhq)

优点：1、使用 HTTPS 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；

2、HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

3、HTTPS 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本

缺点：

1.HTTPS 协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用

2.HTTPS 协议还会影响缓存，增加数据开销和功耗，甚至已有安全措施也会受到影响也会因此而受到影响。

3.SSL 证书需要钱。功能越强大的证书费用越高。个人网站、小网站没有必要一般不会用。

4.HTTPS 连接服务器端资源占用高很多，握手阶段比较费时网站的相应速度有负面影响。

5.HTTPS 连接缓存不如 HTTP 高效。

## 20. HTTPS 是如何实现安全传输数据的。(2018-4-20-xhq)

HTTPS 其实就是在 HTTP 跟 TCP 中间加多了一层加密层 TLS/SSL。SSL 是个加密套件，负责对 HTTP 的数据进行加密。TLS 是 SSL 的升级版。现在提到 HTTPS，加密套件基本指的是 TLS。原先是应用层将数据直接给到 TCP 进行传输，现在改成应用层将数据给到 TLS/SSL，将数据加密后，再给到 TCP 进行传输。

## 21. TTL，MSL，RTT？(2018-4-20-xhq)

MSL：“报文最大生存时间”，他是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。



TTL : TTL 是 time to live 的缩写，中文可以译为“生存时间”，这个生存时间是由源主机设置初始值但不是存的具体时间，而是存储了一个 ip 数据报可以经过的最大路由数，每经过一个处理他的路由器此值就减 1，当此值为 0 则数据报将被丢弃，同时发送 ICMP 报文通知源主机。RFC 793 中规定 MSL 为 2 分钟，实际应用中常用的是 30 秒，1 分钟和 2 分钟等。TTL 与 MSL 是有关系的但不是简单的相等的关系，MSL 要大于等于 TTL。

RTT : RTT 是客户到服务器往返所花时间 ( round-trip time , 简称 RTT ) , TCP 含有动态估算 RTT 的算法。TCP 还持续估算一个给定连接的 RTT，这是因为 RTT 受网络传输拥塞程序的变化而变化。

## 22. 谈一谈你对 Selenium 和 PhantomJS 了解 ( 2018-4-23-xhq )

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。相比传统的 Chrome 或 Firefox 浏览器等，资源消耗会更少。

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 JavaScript、Cookie、headers，以及任何我们真实用户需要做的事情。

主程序退出后 ,selenium 不保证 phantomJS 也成功退出 ,最好手动关闭 phantomJS 进程。( 可能会导致多个 phantomJS 进程运行 , 占用内存 )。

WebDriverWait 虽然可能会减少延时 ,但是目前存在 bug( 各种报错 ) ,这种情况可以采用 sleep。  
phantomJS 爬数据比较慢 ,可以选择多线程。如果运行的时候发现有的可以运行 , 有的不能 , 可以尝试将 phantomJS 改成 Chrome。

### 23. 代理 IP 里的 “透明” “匿名” “高匿” 分别是指 ? (2018-4-23-xhq)

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。

设置代理有以下两个好处 :

- 1 , 让服务器以为不是同一个客户端在请求
- 2 , 防止我们的真实地址被泄露 , 防止被追究

### 24. requests 返回的 content 和 text 的区别 ?

a)response.text 返回的是 Unicode 型数据 ;

a)response.content 返回的是 bytes 类型 , 也就是二进制数据 ;

b)获取文本使用，`response.text`；

b)获取图片,文件，使用 `response.content`；

c)`response.text`

类型：`str`

解码类型：根据 HTTP 头部对响应的编码作出有根据的推测，推测的文本编码

如何修改编码方式：`response.encoding="gbk"`

c)`response.content`

类型：`bytes`

解码类型：没有指定

如何修改编码方式：`response.content.decode("utf8")`

## 25. robots 协议(2018-4-23-xhq)

Robots 协议：网站通过 Robots 协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取。

## 26. 为什么 requests 请求需要带上 header ? (2018-4-23-xhq)

原因是：模拟浏览器，欺骗服务器，获取和浏览器一致的内容

header 的形式：字典

`headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36`

`(KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}`

用法：`requests.get(url,headers=headers)`

## 27. dumps,loads 与 dump,load 的区别 ? (2018-4-23-xhq)

json.dumps()将 pyhton 的 dict 数据类型编码为 json 字符串 ;

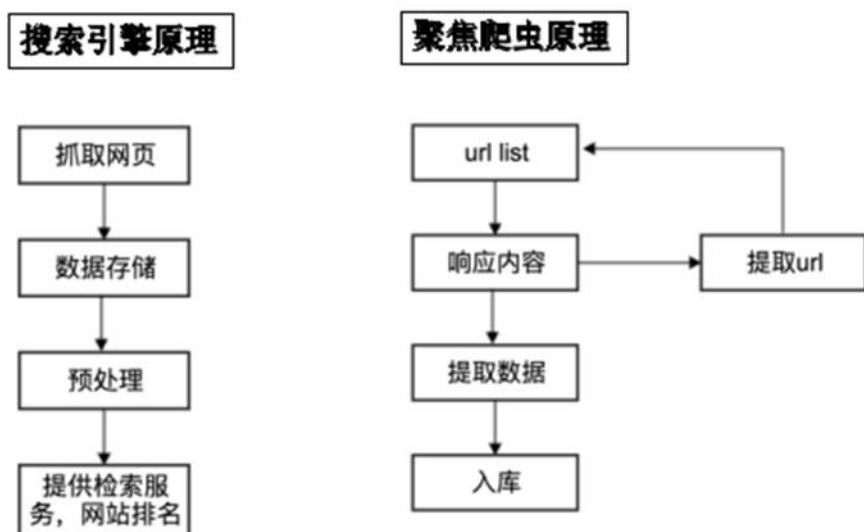
json.loads()将 json 字符串解码为 dict 的数据类型 ;

json.dump(x,y) x 是 json 对象, y 是文件对象 , 最终是将 json 对象写入到文件中 ;

json.load(y) 从文件对象 y 中读取 json 对象。

## 28. 通用爬虫 : 通常指搜索引擎的爬虫 ( 2018-4-23-xhq )

聚焦爬虫 : 针对特定网站的爬虫



通用搜索引擎的局限性 :

通用搜索引擎所返回的网页里 90%的内容无用。

图片、数据库、音频、视频多媒体的内容通用搜索引擎无能为力。不同用户搜索的目的不全相同，但是返回内容相同。

## 29. requests 使用小技巧(2018-4-23-xhq)

1、requests.util.dict\_from\_cookiejar 把 cookie 对象转化为字典

1.1. requests.get(url,cookies={})

2、设置请求不用 SSL 证书验证

```
response = requests.get("https://www.12306.cn/mormhweb/ ", verify=False)
```

3、设置超时

```
response = requests.get(url,timeout=10)
```

4、配合状态码判断是否请求成功

```
assert response.status_code == 200
```

## 30. 平常怎么使用代理的 ? (2018-4-23-xhq)

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 ) )

## 31. 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 ?

(2018-4-23-xhq)

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。

### 32. 怎么获取加密的数据？（2018-4-23-xhq）

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

### 33. 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？（2018-4-23-xhq）

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

### 34. 怎么监控爬虫的状态（2018-4-23-xhq）

1. 使用 python 的 STMP 包将爬虫的状态信息发送到指定的邮箱
2. Scrapyd、pyspider
3. 引入日志

## 二 . Scrapy

### 1. 描述下 scrapy 框架运行的机制？(2018-4-16-lxy)

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。

### 2. 谈谈你对 Scrapy 的理解？(2018-4-23-lyf)

scrapy 是一个为了爬取网站数据，提取结构性数据而编写的应用框架，我们只需要实现少量代码，就能够快速的抓取到数据内容。Scrapy 使用了 Twisted 异步网络框架来处理网络通讯，可以加快我们的下载速度，不用自己去实现异步框架，并且包含了各种中间件接口，可以灵活的完成各种需求。

scrapy 框架的工作流程：

1. 首先 Spiders (爬虫) 将需要发送请求的 url(requests) 经 ScrapyEngine (引擎) 交给 Scheduler (调度器)。
2. Scheduler (排序，入队) 处理后，经 ScrapyEngine，DownloaderMiddlewares(可选，主要有 User\_Agent，Proxy 代理) 交给 Downloader。
3. Downloader 向互联网发送请求，并接收下载响应 (response)。将响应 (response) 经 ScrapyEngine，SpiderMiddlewares(可选) 交给 Spiders。

4.Spiders 处理 response ,提取数据并将数据经 ScrapyEngine 交给 ItemPipeline 保存( 可以是本地,可以是数据库)。提取 url 重新经 ScrapyEngine 交给 Scheduler 进行下一个循环。直到无 Url 请求程序停止结束。

### 3. 怎么样让 scrapy 框架发送一个 post 请求 ( 具体写出来 ) (2018-4-23-lyf)

使用 FormRequest

```
1. class mySpider(scrapy.Spider):
2.     # start_urls = ["http://www.taobao.com/"]
3.     def start_requests(self):
4.         url = 'http://http://www.taobao.com//login'
5.         # FormRequest 是 Scrapy 发送 POST 请求的方法
6.         yield scrapy.FormRequest(
7.             url = url ,
8.             formdata = {"email" : "xxx" , "password" : "xxxxx"} ,
9.             callback = self.parse_page
10.        )
11.     def parse_page(self , response):
12.         # do something
```

### 4. 怎么判断网站是否更新 ? (2018-4-23-lyf)

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,则页面没有更新,否则网页就有更新。

### 5. 图片、视频爬取怎么绕过防盗连接,或者说怎么获取正确的链接地址? (2018-4-23-lyf)

自定义 Referer(建议自行 Google 相关知识)。



## 6. 你爬出来的数据量大概有多大？大概多长时间爬一次？(2018-4-23-lyf)

无标准答案，根据自己爬取网站回答即可（几百万，几千万，亿级）。

## 7. 增量爬取(2018-4-23-lyf)

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。

## 8. 爬虫向数据库存数据开始和结束都会发一条消息，是 scrapy 哪个模块实现的？ ( 2018-4-20-xhq )

Scrapy 使用信号来通知事情发生，因此答案是 signals 模块。

## 9. 爬取下来的数据如何去重，说一下具体的算法依据（2018-4-20-xhq）

1.通过 MD5 生成电子指纹来判断页面是否改变

2.nutch 去重。nutch 中 digest 是对采集的每一个网页内容的 32 位哈希值，如果两个网页内容完全一样，它们的 digest 值肯定会一样。

数据量不大时，可以直接放在内存里面进行去重，python 可以使用 set()进行去重。当去重数据需要持久化时可以使用 redis 的 set 数据结构。

当数据量再大一点时，可以用不同的加密算法先将长字符串压缩成 16/32/40 个字符，再使用上面两种方法去重。

当数据量达到亿（甚至十亿、百亿）数量级时，内存有限，必须用“位”来去重，才能够满足需求。Bloomfilter 就是将去重对象映射到几个内存“位”，通过几个位的 0/1 值来判断一个对象是否已经存在。

然而 Bloomfilter 运行在一台机器的内存上，不方便持久化（机器 down 掉就什么都没了），也不方便分布式爬虫的统一去重。如果可以在 Redis 上申请内存进行 Bloomfilter，以上两个问题就都能解决了。

simhash 最牛逼的一点就是将一个文档，最后转换成一个 64 位的字节，暂且称之为特征字，然后判断重复只需要判断他们的特征字的距离是不是  $< n$ （根据经验这个  $n$  一般取值为 3），就可以判断两个文档是否相似。

可见 scrapy\_redis 是利用 set 数据结构来去重的，去重的对象是 request 的 fingerprint（其实就是用 hashlib.sha1()对 request 对象的某些字段信息进行压缩）。其实 fp 就是 request 对象加密压缩后的一个字符串（40 个字符，0~f）。

## 10.Scrapy 的优缺点? ( 2018-4-23-xhq )

优点：

- 1 ) scrapy 是异步的
- 2 ) 采取可读性更强的 xpath 代替正则
- 3 ) 强大的统计和 log 系统
- 4 ) 同时在不同的 url 上爬行
- 5 ) 支持 shell 方式，方便独立调试
- 5 ) 写 middleware,方便写一些统一的过滤器
- 6 ) 通过管道的方式存入数据库

缺点：

- 1 ) 基于 python 的爬虫框架，扩展性比较差
- 2 ) 基于 twisted 框架，运行中的 exception 是不会干掉 reactor（反应器），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。

## 11.怎么设置深度爬取?(2018-4-23-xhq)

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。

### 三 . Scrapy-redis

#### 1. scrapy 和 scrapy-redis 有什么区别？为什么选择 redis 数据库？(2018-4-16-lxy)

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。

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. 分布式爬虫主要解决什么问题？(2018-4-16-lxy)

- 1.ip
- 2.带宽
- 3.cpu
- 4.io

#### 3. 什么是分布式存储？(2018-4-23-lyf)

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。

**容错**：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**：新增的服务器在集群中保障负载均衡？数据迁移过程中保障不影响现有的服务。

**事务与并发控制**：实现分布式事务。

**易用性**：设计对外接口，使得设计的系统易于使用。

#### 4. 你所知道的分布式爬虫方案有哪些？(2018-4-23-lyf)

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”。

5. 除了 scrapy-redis，有做过其他的分布式爬虫吗？(2018-4-23-lyf)

Celery、gearman 等，参考其他分布式爬虫策略。

6. 在爬取的时候遇到某些内容字段缺失怎么判断及处理？(2018-4-23-lyf)

判读字段缺失，做异常处理即可。

## 四．自定义框架

# 第七章 shell 与自动化运维

# 第八章 测试

## 一、 测试面试题

1. 禅道和 qc 的区别？(2018-4-23-zcz)

都是缺陷管理工具。

A.QC

作为缺陷管理工具，QC 在缺陷管理方面，做的相对完善。提 bug 页面：填写内容可以根据测试需求，不断修改添加新的字段；以我上一家公司为例，在提 bug 过程中，有以下几个必填项：

Bug 状态（new、fixed、closed 等）、发现人员、缺陷发现阶段(测试阶段、上现阶段等)、缺陷来源（测试人员给出的 bug 定位）、Bug 分类（功能、性能等问题）、测试阶段（单元测试、集成测试、

系统测试等)、归属需求、缺陷回归次数、优先级、分配给,这些必填项再加上 bug 标题和操作描述、上传附件,使很多疑问都变得清晰。

缺陷查看页面:可以根据自己需要选择要呈现的字段,相对人性化可操作,每个显示的字段都可以进行筛选,使研发人员很快能定位到属于自己的 bug,再根据 bug 状态、优先级进行筛选,使未完结的 bug 能有序并无遗漏地完成修改;页面还有注释功能,研发人员能写出针对本问题的各种感想,方便完善而又人性化。

### B.禅道(开源版)

禅道涉及面非常广,但是在缺陷管理这方面,与老牌的 QC 还是略逊一筹。提 bug 页面:页面是非常清晰整洁的 web 页面,但是需要填写的字段,并没有完全覆盖开发和测试人员的全部需求。页面字段:产品模块(对应 QC 中的项目)、所属项目(对应 QC 中的需求)、影响版本(bug 所属版本? )、当前指派(修改 bug 的人员)、bug 标题、重现步骤、相关需求(页面标注了这个字段,但是什么也没有显示,并且没有可填写的位置)、相关任务、类型/严重。

## 2. 编写测试计划的目的是(2018-4-23-zcz)

1. 使测试工作顺利进行
2. 使项目参与人员沟通更舒畅
3. 使测试工作更加系统化

## 3. 测试人员在软件开发过程中的任务是什么(2018-4-23-zcz)

1. 寻找 Bug ;
2. 避免软件开发过程中的缺陷 ;
3. 衡量软件的品质 ;



4. 关注用户的需求。
5. 总的目标是：确保软件的质量。

4. 您以往的工作中，一条软件缺陷（或者叫 Bug）记录都包含了哪些内容？如何提交高质量的软件缺陷（Bug）记录？(2018-4-23-zcz)

一条 Bug 记录最基本应包含：编号、Bug 所属模块、Bug 描述、Bug 级别、发现日期、发现人、修改日期、修改人、修改方法、回归结果等等；要有效的发现 Bug 需参考需求以及详细设计等前期文档设计出高效的测试用例，然后严格执行测试用例，对发现的问题要充分确认肯定，然后再向外发布如此才能提高提交 Bug 的质量。

5. 简述黑盒测试和白盒测试的优缺点(2018-4-23-zcz)

※ 黑盒测试的优点有：

- 1) 比较简单，不需要了解程序内部的代码及实现；
- 2) 与软件的内部实现无关；
- 3) 从用户角度出发，能很容易的知道用户会用到哪些功能，会遇到哪些问题；
- 4) 基于软件开发文档，所以也能知道软件实现了文档中的哪些功能；
- 5) 在做软件自动化测试时较为方便。

※ 黑盒测试的缺点有：

- 1) 不可能覆盖所有的代码，覆盖率较低，大概只能达到总代码量的 30%；
- 2) 自动化测试的复用性较低。

※ 白盒测试的优点有：

- 1) 帮助软件测试人员增大代码的覆盖率，提高代码的质量，发现代码中隐藏的问题。

※ 白盒测试的缺点有：

2) 程序运行会有很多不同的路径，不可能测试所有的运行路径；测试基于代码，只能测试开发人员做的对不对，而不能知道设计的正确与否，可能会漏掉一些功能需求；系统庞大时，测试开销会非常大。

6. 简述常用的 Bug 管理或者用例管理工具,并且描述其中一个工作流程。

(2018-4-23-zcz)

常用：testlink，QC，mantis，禅道，TAPD，JIRA。

TAPD：产品创建(需求，计划，模块)-->项目创建（PM 排期、任务分解）-->研发(编码、单元测试等)-->测试(测试计划，用例，执行，bug，报告等)。

7. 请列出你所知道的软件测试种类，至少 5 项。(2018-4-23-zcz)

单元测试，集成测试，系统测试，验收测试。

系统测试包含：功能测试，性能测试，压力测试，兼容性测试，健壮性测试，冒烟测试，文档测试。

8. Alpha 测试与 Beta 测试的区别是什么？(2018-4-23-zcz)

Alpha 主要是模拟用户的操作和用户的环境。

Beta 主要验证测试，准备进入发布阶段，Beta 测试是一种验收测试。

9. 举例说明什么是 Bug？一个 bug report 应包含什么关键字？(2018-4-23-zcz)

比如聊天中，点击发送按钮后，无法发送消息。

标题，模块，严重程度，bug 类型，版本号，可否重现，描述，附件，日志等等。

## 第九章 数据库

### 一 . Mysql

#### 1. Python 中操作 Mysql 步骤 ( 2018-3-31-sxd)

#### 使用Python DB API访问数据库流程



代码实现：

```
1. #首先安装包-pymysql sudo pip install pymysql
2. #之后在程序中调用 from pymysql import *
3. ''' connection 对象 用于建立与数据库的连接 创建对象：调用 connect()方法 '''
4. conn = connect(参数列表)
5. ''' 参数列表：
6. host:连接 MySQL 主机，如果是本机则为" localhost "
7. port:连接 MySQL 主机端口，默认 3306
8. database:数据库名称
```

```
9. user : 连接的用户名
10. password : 连接的密码
11. charset:通信采用的编码方式，推荐采用 utf8
12. ""
13. "" connection 对象方法
14. close() 关闭连接
15. commit() 提交
16. rollback() 回滚
17. cursor() 返回
18. cursor 对象，用于执行 sql 语句
19. 例如 : select,insert,update,delete ""
20. cs1 = conn.cursor()
21. "" cursor 对象方法
22. close() 关闭
23. execute(operation[,parameters])执行语句，返回受影响的行数，主要用于执行 insert、update、delete 语句，
24. 也可以执行 create、alter、drop 等语句
25. fetchone()执行查询语句时，获取查询结果集的第一个行数据，返回一个元组
26. fetchall()执行查询时，获取结果集的所有行，一行构成一个元组，再将这些元组装入一个元组返回 ""
27. ""
28. cursor 对象属性
29. rowcount 只读属性，表示最近一次
30. execute()执行后受影响的行数
31. connection 获得当前连接对象 ""
32. #例子
33. #创建 Connection 连接
34. conn = connect(host='localhost', port=3306, user='root', password='mysql', database='python1',
35. charset='utf8')
36. #获得 Cursor 对象 cs = conn.cursor()
37. # 更新 #
38. sql = 'update students set name="刘邦" where id=6'
39. #删除
40. # sql = 'delete from students where id=6'
41. #执行 select 语句，并返回受影响的行数：查询一条学生数据
42. sql = 'select id,name from students where id = 7'
43. # sql = 'SELECT id,name FROM students WHERE id = 7' count=cs.execute(sql)
44. #打印受影响的行数

45. print count
```

## 2. SQL 的 select 语句完整的执行顺序 ( 2018-3-31-sxd )

SQL Select 语句完整的执行顺序：

- 1、from 子句组装来自不同数据源的数据；
- 2、where 子句基于指定的条件对记录行进行筛选；
- 3、group by 子句将数据划分为多个分组；
- 4、使用聚集函数进行计算；
- 5、使用 having 子句筛选分组；
- 6、计算所有的表达式；
- 7、select 的字段；
- 8、使用 order by 对结果集进行排序。

SQL 语言不同于其他编程语言的最明显特征是处理代码的顺序。在大多数据库语言中，代码按编码顺序被处理。但在 SQL 语句中，第一个被处理的子句式 FROM，而不是第一出现的 SELECT。

SQL 查询处理的步骤序号：

- (1) FROM <left\_table>
- (2) <join\_type> JOIN <right\_table>
- (3) ON <join\_condition>
- (4) WHERE <where\_condition>
- (5) GROUP BY <group\_by\_list>
- (6) WITH {CUBE | ROLLUP}
- (7) HAVING <having\_condition>
- (8) SELECT
- (9) DISTINCT
- (9) ORDER BY <order\_by\_list>
- (10) <TOP\_specification> <select\_list>

以上每个步骤都会产生一个虚拟表，该虚拟表被用作下一个步骤的输入。这些虚拟表对调用者(客户端应用程序或者外部查询)不可用。只有最后一步生成的表才会给调用者。如果没有在查询中指定某一个子句，将跳过相应的步骤。

逻辑查询处理阶段简介：

- 1、 FROM：对 FROM 子句中的前两个表执行笛卡尔积(交叉联接)，生成虚拟表 VT1。
- 2、 ON：对 VT1 应用 ON 筛选器，只有那些使为真才被插入到 TV2。
- 3、 OUTER JOIN:如果指定了 OUTER JOIN(相对于 CROSS JOIN 或 INNER JOIN)，保留表中未找到匹配的行将作为外部行添加到 VT2，生成 TV3。如果 FROM 子句包含两个以上的表，则对上一个联接生成的结果表和下一个表重复执行步骤 1 到步骤 3，直到处理完所有的表位置。
- 4、 WHERE：对 TV3 应用 WHERE 筛选器，只有使为 true 的行才插入 TV4。
- 5、 GROUP BY：按 GROUP BY 子句中的列列表对 TV4 中的行进行分组，生成 TV5。
- 6、 CUTE|ROLLUP：把超组插入 VT5，生成 VT6。
- 7、 HAVING：对 VT6 应用 HAVING 筛选器，只有使为 true 的组插入到 VT7。
- 8、 SELECT：处理 SELECT 列表，产生 VT8。
- 9、 DISTINCT：将重复的行从 VT8 中删除，产品 VT9。
- 10、 ORDER BY 将 VT9 中的行按 ORDER BY 子句中的列列表顺序 生成一个游标(VC10)。
- 11、 TOP：从 VC10 的开始处选择指定数量或比例的行，生成表 TV11，并返回给调用者。

where 子句中的条件书写顺序

### 3. 说一下 Mysql 数据库存储的原理？(2018-4-16-lxy)

存储过程是一个可编程的函数，它在数据库中创建并保存。它可以有 SQL 语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。它允许控制数据的访问方式。存储过程通常有以下优点：

- 1、存储过程能实现较快的执行速度
- 2、存储过程允许标准组件是编程。
- 3、存储过程可以用流程控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。
- 4、存储过程可被作为一种安全机制来充分利用。
- 5、存储过程能够减少网络流量

### 4. 事务的特性？(2018-4-16-lxy)

- 1、原子性(Atomicity)：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不执行。
- 2、一致性(Consistency)：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致。
- 3、隔离性(Isolation)：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。
- 4、持久性(Durability)：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障

## 5. 数据库索引 ? (2018-4-16-lxy)

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\_TREE。B\_TREE 索引加速了数据访问，因为存储引擎不会再去扫描整张表得到需要的数据；相反，它从根节点开始，根节点保存了子节点的指针，存储引擎会根据指针快速寻找数据。

## 6. 数据库怎么优化查询效率 ? (2018-4-16-lxy)

1、储存引擎选择：如果数据表需要事务处理，应该考虑使用 InnoDB，因为它完全符合 ACID 特性。

如果不需要事务处理，使用默认存储引擎 MyISAM 是比较明智的

2、分表分库，主从。

3、对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引

4、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

5、应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫描

6、应尽量避免在 where 子句中使用 or 来连接条件，如果一个字段有索引，一个字段没有索引，将导致引擎放弃使用索引而进行全表扫描

7、Update 语句，如果只更改 1、2 个字段，不要 Update 全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志

8、对于多张大数据量（这里几百条就算大了）的表 JOIN，要先分页再 JOIN，否则逻辑读会很高，性能很差。



## 7. Mysql 集群的优缺点 ? (2018-4-16-lxy)

优点：

- 99.999%的高可用性
- 快速的自动失效切换
- 灵活的分布式体系结构，没有单点故障
- 高吞吐量和低延迟
- 可扩展性强，支持在线扩容

缺点：

- 存在很多限制，比如：不支持外键
- 部署、管理、配置很复杂
- 占用磁盘空间大、内存大
- 备份和恢复不方便
- 重启的时候，数据节点将数据 load 到内存需要很长的时间

## 8. 你用的 Mysql 是哪个引擎，各引擎之间有什么区别 ? (2018-4-16-lxy)

主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：

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；

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；

InnoDB 支持外键，MyISAM 不支持；

MyISAM 是默认引擎，InnoDB 需要指定；

InnoDB 不支持 FULLTEXT 类型的索引；

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；

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；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；

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%'`

## 9. 数据库的优化？(2018-4-16-lxy)

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,RAID5)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\*\*。

## 10. Mysql 数据库如何分区、分表？(2018-4-16-lxy)

分表可以通过三种方式：Mysql 集群、自定义规则和 merge 存储引擎。

分区有四类：

RANGE 分区：基于属于一个给定连续区间的列值，把多行分配给分区。

LIST 分区：类似于按 RANGE 分区，区别在于 LIST 分区是基于列值匹配一个离散值集合中的某个值来进行选择。

HASH 分区：基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含 MySQL 中有效的、产生非负整数值的任何表达式。

KEY 分区：类似于按 HASH 分区，区别在于 KEY 分区只支持计算一列或多列，且 MySQL 服务器提供其自身的哈希函数。必须有一列或多列包含整数值。

## 11. 如何对查询命令进行优化？(2018-4-16-lxy)

a. 应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索。

b. 应尽量避免在 where 子句中对字段进行 null 值判断，避免使用!=或<>操作符，避免使用 or 连接条件，或在 where 子句中使用参数、对字段进行表达式或函数操作，否则会导致权标扫描

c. 不要在 where 子句中的 “=” 左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

d. 使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用。

e. 很多时候可考虑用 exists 代替 in。

f. 尽量使用数字型字段。

g. 尽可能的使用 varchar/nvarchar 代替 char/nchar。

- h. 任何地方都不要使用 `select from t` ,用具体的字段列表代替 `""` ,不要返回用不到的任何字段。
- i. 尽量使用表变量来代替临时表。
- j. 避免频繁创建和删除临时表，以减少系统表资源的消耗。
- k. 尽量避免使用游标，因为游标的效率较差。
- l. 在所有的存储过程和触发器的开始处设置 `SET NOCOUNT ON` ,在结束时设置 `SET NOCOUNT OFF`。
- m. 尽量避免大事务操作，提高系统并发能力。
- n. 尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

## 12. Sql 注入是如何产生的，如何防止？(2018-4-16-lxy)

程序开发过程中不注意规范书写 sql 语句和对特殊字符进行过滤，导致客户端可以通过全局变量 POST 和 GET 提交一些 sql 语句正常执行。产生 Sql 注入。下面是防止办法：

- a. 过滤掉一些常见的数据库操作关键字，或者通过系统函数来进行过滤。
- b. 在 PHP 配置文件中将 `Register_globals=off`; 设置为关闭状态
- c. SQL 语句书写的时候尽量不要省略小引号(tab 键上面那个)和单引号
- d. 提高数据库命名技巧，对于一些重要的字段根据程序的特点命名，取不易被猜到的
- e. 对于常用的方法加以封装，避免直接暴露 SQL 语句
- f. 开启 PHP 安全模式：`Safe_mode=on`;
- g. 打开 `magic_quotes_gpc` 来防止 SQL 注入
- h. 控制错误信息：关闭错误提示信息，将错误信息写到系统日志。
- i. 使用 `mysqli` 或 `pdo` 预处理。

### 13. NoSQL 和关系数据库的区别？(2018-4-16-lxy)

a. 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。

b. 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。

c. SQL 中如果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。而在 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。

d. 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。

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

f. 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。

g. 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。

## 14. Mysql 数据库中怎么实现分页？(2018-4-23-zcz)

`select * from table limit (start-1)*limit,limit;` 其中 start 是页码，limit 是每页显示的条数。

## 15. sql 语句怎么看效率？(2018-4-23-zcz)

SQLServer2005-->新建一个查询-->输入语句 `SELECT * FROM Person.Contact`

执行(F5)-->Ctrl+L。

## 16. 优化数据库？提高数据库的性能？(2018-4-23-zcz)

### 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

例如 float 和 int、char 和 varchar、binary 和 varbinary 是不兼容的。

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
```

在这条语句中,如 salary 字段是 money 型的,则优化器很难对其进行优化,因为 60000 是个整型数。我们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.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。

## 17. 提取数据库中倒数 10 条数据？(2018-4-23-zcz)

```
select top (10) * from table1 order by id desc.
```

## 18. 数据库负载均衡(2018-4-23-zcz)

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、 实现多数据库数据同步



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。这是一个集群所必需的，因为，如果数据不实时、不同步，那么用户从一台服务器读出的数据，就有别于从另一台服务器读出的数据，这是不能允许的。所以必须实现数据库的数据同步。这样，在查询的时候就可以有多个资源，实现均衡。比较常用的方法是 Moebius for SQL Server 集群，Moebius for SQL Server 集群采用将核心程序驻留在每个机器的数据库中的办法，这个核心程序称为 Moebius for SQL Server 中间件，主要作用是监测数据库内数据的变化并将变化的数据同步到其他数据库中。数据同步完成后客户端才会得到响应，同步过程是并发完成的，所以同步到多个数据库和同步到一个数据库的时间基本相等；另外同步的过程是在事务的环境下完成的，保证了多份数据在任何时刻数据的一致性。正因为 Moebius 中间件宿主在数据库中的创新，让中间件不但能知道数据的变化，而且知道引起数据变化的 SQL 语句，根据 SQL 语句的类型智能的采取不同的数据同步的策略以保证数据同步成本的最小化。

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 :

a) 不能够按照 Web 服务器的处理能力分配负载。

b) 负载均衡器(控制端)故障 ,会导致整个数据库系统瘫痪。

## 19. Mysql 数据库的操作?(2018-5-1-lxy)

修改表-修改字段 , 重命名版 :

alter table 表名 change 原名 新名 类型及约束 ;

alter table students change birthday birth datetime not null;

修改表-修改字段 , 不重名版本 :

alter table 表名 modify 列名 类型和约束 ;

alter table students modify birth date not null

全列插入 : insert into 表名 values(...)

insert into students values(0,"郭靖", 1,"内蒙","2017-6");

部分插入 : 值的顺序与给出的列顺序对应 :

insert into students(name, birthday) values("黄蓉","2017-8");

修改 : update 表名 set 列 1=值 1 , 列 2=值 2。。 where

update students set gender=0, homwtown="古墓" , where id = 5;

备份 : mysqldump -uroot -p 数据库名 > python.sql,

恢复 : mysql -uroot -p 数据库名 < python.sql

## 20. 数据库的设计 ? (2018-5-1-lxy)

第一范式：数据库表的每一列都是不可分割的原子数据项，即列不可拆分。

第二范式：建立在第一范式的基础上，要求数据库表中的每个实例或记录必须是可以唯一被区分的，即唯一标识。

第三范式：建立在第二范式的基础上，任何非主属性不依赖与其他非主属性，即引用主键。

## 21. 存储过程和函数的区别?(2018-5-1-lxy)

相同点：存储过程和函数都是为了可重复的执行操作数据库的 sql 语句的集合。

1) 存储过程和函数都是一次编译，就会被缓存起来，下次使用就直接命中已经编译好的 sql 语句，不需要重复使用。减少网络交互，减少网络访问流量。

不同点：标识符不同，函数的标识符是 function，存储过程是 procedure。

1) 函数中有返回值，且必须有返回值，而过程没有返回值，但是可以通过设置参数类型 (in,out) 来实现多个参数或者返回值。

2) 存储函数使用 select 调用，存储过程需要使用 call 调用。

3) select 语句可以在存储过程中调用，但是除了 select..into 之外的 select 语句都不能在函数中使用。

4) 通过 in out 参数，过程相关函数更加灵活，可以返回多个结果。

## 22. Mysql 日志 (2018-5-1-lyf)

错误日志：记录启动，运行或者停止 mysql 时出现的问题；

通用日志：记录建立的客户端连接和执行的语句；

二进制日志：记录所有更改数据的语句；

慢查询日志：记录所有执行时间超过 long\_query\_time 秒的查询或者不适用索引的查询)

通过使用--slow\_query\_log[={0|1}]选项来启用慢查询日志，所有执行时间超多 long\_query\_time 的语句都会被记录。

## 二 . MongoDB

### 1. 数据库的一些基本操作命令（列举一些常用命令即可）？(2018-4-23-zcz)

MySQL 的常见命令如下：

1. create database name; 创建数据库
2. use databasename; 选择数据库
3. drop database name 直接删除数据库，不提醒
4. show tables; 显示表
5. describe tablename; 表的详细描述
6. select 中加上 distinct 去除重复字段
7. mysqladmin drop databasename 删除数据库前，有提示。
8. 显示当前 mysql 版本和当前日期
9. select version(),current\_date;

MongoDB 的常见命令如下：

1. db.help(); Help 查看命令提示
2. use yourDB; 切换/创建数据库
3. show dbs; 查询所有数据库
4. db.dropDatabase(); 删除当前使用数据库
5. db.getName(); 查看当前使用的数据库

6. db.version(); 当前 db 版本
7. db.addUser("name"); 添加用户
8. db.addUser("userName", "pwd123", true);
9. show users; 显示当前所有用户
10. db.removeUser("userName"); 删除用户
11. db.yourColl.count(); 查询当前集合的数据条数

更多命令参考博客：<https://blog.csdn.net/wangpeng047/article/details/7705588>



2. Python 中调用 mongo 数据库的包叫什么？(2018-4-23-zcz)

Pymongo。

3. MongoDB (2018-4-23-zcz)

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：

- a) 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。
- b) 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。
- c) 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。
- d) 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。
- e) 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。
- f) 重要数据：mysql，一般数据：mongodb，临时数据：memcache
- g) 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。
- h) 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:
- 1) 弱一致性（最终一致），更能保证用户的访问速度
  - 2) 文档结构的存储方式，能够更便捷的获取数
  - 3) 内置 GridFS，高效存储二进制大对象（比如照片和视频）
  - 4) 支持复制集、主备、互为主备、自动分片等特性
  - 5) 动态查询
  - 6) 全索引支持,扩展到内部对象和内嵌数组

- 缺点:
- 1) 不支持事务
  - 2) MongoDB 占用空间过大,维护工具不够成熟

#### 4. MongoDB 成为优秀的 NoSQL 数据库的原因是什么? (2018-4-23-zcz)

以下特点使得 MongoDB 成为优秀的 NoSQL 数据库：

- 1) 面向文件的
- 2) 高性能
- 3) 高可用性
- 4) 易扩展性
- 5) 丰富的查询语言

## 5. 分析器在 MongoDB 中的作用是什么? (2018-4-23-zcz)

MongoDB 中包括了一个可以显示数据库中每个操作性能特点的数据库分析器。通过这个分析器你可以找到比预期慢的查询(或写操作);利用这一信息,比如,可以确定是否需要添加索引。

## 6. 怎么查看 MongoDB 正在使用的链接? (2018-4-23-zcz)

```
db._adminCommand("connPoolStats");
```

## 7. MySQL 与 MongoDB 本质之间最基本的差别是什么(2018-4-23-zcz)

差别在多方面,例如:数据的表示、查询、关系、事务、模式的设计和定义、速度和性能。

MongoDB 是由 C++ 语言编写的,是一个基于分布式文件存储的开源数据库系统。在高负载的情况下,添加更多的节点,可以保证服务器性能。

MongoDB 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档,数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档,数组及文档数组。

MongoDB 是一个面向文档的数据库,目前由 10gen 开发并维护,它的功能丰富齐全,所以完全可以替代 MySQL。

与 MySQL 等关系型数据库相比,MongoDB 的优点如下:

- ①弱一致性,更能保证用户的访问速度。
- ②文档结构的存储方式,能够更便捷的获取数据。
- ③内置 GridFS,支持大容量的存储。
- ④内置 Sharding。
- ⑤第三方支持丰富。(这是与其他的 NoSQL 相比,MongoDB 也具有的优势)



## ⑥性能优越：

MongoDB 本身它还算比较年轻的一个产品，所以它的问题，就是成熟度肯定没有传统 MySQL 那么成熟稳定。所以在使用的时候，

第一，尽量使用稳定版，不要在线上使用开发版，这是一个大原则；

另外一点，备份很重要，MongoDB 如果出现一些异常情况，备份一定是要能跟上。除了通过传统的复制的方式来做备份，离线备份也还是要有，不管你是用什么方式，都要有一个完整的离线备份。往往最后出现了特殊情况，它能帮助到你；

另外，MongoDB 性能的一个关键点就是索引，索引是不是能有比较好的使用效率，索引是不是能够放在内存中，这样能够提升随机读写的性能。如果你的索引不能完全放在内存中，一旦出现随机读写比较高的时候，它就会频繁地进行磁盘交换，这个时候，MongoDB 的性能就会急剧下降，会出现波动。

另外，MongoDB 还有一个最大的缺点，就是它占用的空间很大，因为它属于典型空间换时间原则的类型。那么它的磁盘空间比普通数据库会浪费一些，而且到目前为止它还没有实现在线压缩功能，在 MongoDB 中频繁的进行数据增删改时，如果记录变了，例如数据大小发生了变化，这时候容易产生一些数据碎片，出现碎片引发的结果，一个是索引会出现性能问题，

另外一个就是在一定的时间后，所占空间会莫名其妙地增大，所以要定期把数据库做修复，定期重新做索引，这样会提升 MongoDB 的稳定性和效率。在最新的版本里，它已经在实现在线压缩，估计应该在 2.0 版左右，应该能够实现在线压缩，可以在后台执行现在 repair DataBase 的一些操作。如果那样，就解决了目前困扰我们的大问题。

## 8. 使用 MongoDB 的优点(2018-4-23-zcz)

### 1. 面向文件

2. 高性能
  3. 高可用
  4. 易扩展
  5. 可分片
  6. 对数据存储友好
9. 关于 MongoDB 更多知识请扫描二维码查看(2018-4-23-zcz)



### 三 . Redis

1. Redis 中 list 底层实现有哪几种？有什么区别？(2018-4-16-lxy)

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。

更多详细内容请见：<https://blog.csdn.net/caishenfans/article/details/44784131>



## 2. 怎样解决数据库高并发的问题？(2018-4-23-zcz)

解决数据库高并发的常见方案：

### 1) 缓存式的 Web 应用程序架构：

在 Web 层和 DB(数据库)层之间加一层 cache 层，主要目的：减少数据库读取负担，提高数据读取速度。cache 存取的媒介是内存，可以考虑采用分布式的 cache 层，这样更容易破除内存容量的限制，同时增加了灵活性。

### 2) 增加 Redis 缓存数据库：

参考博客：[https://www.cnblogs.com/Leo\\_wl/p/5791327.html](https://www.cnblogs.com/Leo_wl/p/5791327.html)



### 3) 增加数据库索引

### 4) 页面静态化：

效率最高、消耗最小的就是纯静态化的 html 页面，所以我们尽可能使我们的网站上的页面采用静态页面来实现，这个最简单的方法其实也是最有效的方法。用户可以直接获取页面，不用像 MVC 结构走那么多流程，比较适用于页面信息大量被前台程序调用，但是更新频率很小的情况。

### 5) 使用存储过程：

处理一次请求需要多次访问数据库的操作，可以把操作整合到储存过程，这样只要一次数据库访问就可以了。

### 6) MySQL 主从读写分离：

当数据库的写压力增加，cache 层（如 Memcached）只能缓解数据库的读取压力。读写集中在一个数据库上让数据库不堪重负。使用主从复制技术（master-slave 模式）来达到读写分离，以提高读写性能和读库的可扩展性。读写分离就是只在主服务器上写，只在从服务器上读，基本原理是让主数据库处理事务性查询，而从数据库处理 select 查询，数据库复制被用于把事务性查询（增删改）导致的改变更新同步到集群中的从数据库。

MySQL 读写分离提升系统性能：

- 1、主从只负责各自的读和写，极大程度缓解 X 锁和 S 锁争用。
- 2、slave 可以配置 MyISAM 引擎，提升查询性能以及节约系统开销。
- 3、master 直接写是并发的，slave 通过主库发送来的 binlog 恢复数据是异步的。
- 4、slave 可以单独设置一些参数来提升其读的性能。
- 5、增加冗余，提高可用性。

实现主从分离可以使用 MySQL 中间件如：Atlas

## 7) 分表分库：

在 cache 层的高速缓存，MySQL 的主从复制，读写分离的基础上，这时 MySQL 主库的写压力开始出现瓶颈，而数据量的持续猛增，由于 MyISAM 使用表锁，在高并发下会出现严重的锁问题，大量的高并发 MySQL 应用开始使用 InnoDB 引擎代替 MyISAM。采用 Master-Slave 复制模式的 MySQL 架构，只能对数据库的读进行扩展，而对数据的写操作还是集中在 Master 上。这时需要对数据库的吞吐能力进一步地扩展，以满足高并发访问与海量数据存储的需求。

对于访问极为频繁且数据量巨大的单表来说，首先要做的是减少单表的记录条数，以便减少数据查询所需的时间，提高数据库的吞吐，这就是所谓的**分表【水平拆分】**。在分表之前，首先需要选择适当的分表策略（尽量避免分出来的多表关联查询），使得数据能够较为均衡地分布到多张表中，并且不影响正常的查询。

分表能够解决单表数据量过大带来的查询效率下降的问题，但是却无法给数据库的并发处理能力带来质的提升。面对高并发的读写访问，当数据库 master 服务器无法承载写操作压力时，不管如何扩展 Slave 服务器都是没有意义的，对数据库进行拆分，从而提高数据库写入能力，即**分库【垂直拆分】**。

## 8) 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。

负载均衡将是大型网站解决高负荷访问和大量并发请求采用的终极解决办法。

### 3. Redis 集群实现？(2018-4-23-zcz)

需要结合真实案例去分析，这里给大家推荐一个不错的博客。

<https://blog.csdn.net/yfkiss/article/details/38944179>



### 4. Redis 数据库，内容是以何种结构存放在 Redis 中的？(2018-4-23-zcz)

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

### 5. Redis 的并发竞争问题怎么解决？(2018-4-23-zcz)

方案一：可以使用独占锁的方式，类似操作系统的 mutex 机制，不过实现相对复杂，成本较高。

[https://blog.csdn.net/black\\_ox/article/details/48972085](https://blog.csdn.net/black_ox/article/details/48972085)



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）

如何用乐观锁方式进行解决？

本质上是假设不会进行冲突，使用 redis 的命令 watch 进行构造条件

## 6. MySQL 和 Redis 高可用性体现在哪些方面？(2018-4-23-zcz)

- a) MySQL Replication 是 MySQL 官方提供的主从同步方案，用于将一个 MySQL 实例的数据，同步到另一个实例中。Replication 为保证 数据安全做了重要的保证，也是现在运用最广的 MySQL 容灾方案。Replication 用两个或以上的实例搭建了 MySQL 主从复制集群，提供 单点写入，多点读取的服务，实现了读的 scale out。
- b) Sentinel 是 Redis 官方为集群提供的高可用解决方案。在实际 项目中可以使用 sentinel 去做 Redis 自动故障转移，减少人工介入的工作量。另外 sentinel 也给客户端提供了监控消息的通知，这样客户端就可根据消息类型去判断服务器的状态，去做对应的适配操作。
- c)下面是 Sentinel 主要功能列表：

Monitoring：Sentinel 持续检查集群中的 master、slave 状态，判断是否存活。



Notification : 在发现某个 Redis 实例死的情况下，Sentinel 能通过 API 通知系统管理员或其他程序脚本。

Automatic failover : 如果一个 master 挂掉后，sentinel 立马启动故障转移，把某个 slave 提升为 master。其他的 slave 重新配置指向新 master。

Configuration provider : 对于客户端来说 sentinel 通知是有效可信赖的。客户端会连接 sentinel 去请求当前 master 的地址，一旦发生故障 sentinel 会提供新地址给客户端。

## 7. Redis 和 MongoDB 的优缺点(2018-4-23-zcz)

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：

- a) 读写性能优异
- b) 支持数据持久化，支持 AOF 和 RDB 两种持久化方式
- c) 支持主从复制，主机自动将数据同步到从机，可以进行读写分离。
- d) 数据结构丰富：数据结构丰富：支持 string、hash、set、sortedset、list 等数据结构。

缺点：

- e) 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。
- f) 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。



- g) 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。
- h) 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。

#### MongoDB 优点:

- a) 弱一致性(最终一致),更能保证用户的访问速度
- b) 文档结构的存储方式,能够更便捷的获取数
- c) 内置 GridFS,高效存储二进制大对象(比如照片和视频)
- d) 支持复制集、主备、互为主备、自动分片等特性
- e) 动态查询
- f) 全索引支持,扩展到内部对象和内嵌数组

#### 缺点:

- a) 不支持事务
- b) MongoDB 占用空间过大
- c) 维护工具不够成熟

### 8. Redis 基本类型、相关方法(2018-4-23-zcz)

Redis 支持五种数据类型: string(字符串)、hash(哈希)、list(列表)、set(集合)及 zset(sorted

set: 有序集合)。

## 一、 String

1.String 是 Redis 最为常用的一种数据类型，String 的数据结构为 key/value 类型，String 可以包含任何数据。

2.常用命令: set,get,decr,incr,mget 等。

## 二、 Hash

1.Hash 类型可以看成是一个 key/value 都是 String 的 Map 容器。

2.常用命令：hget,hset,hgetall 等。

## 三、 List

1.List 用于存储一个有序的字符串列表，常用的操作是向队列两端添加元素或者获得列表的某一段。

2.常用命令：lpush,rpush,lpop,rpop,lrange 等

## 四、 Set

1.Set 可以理解为一组无序的字符集合，Set 中相同的元素是不会重复出现的，相同的元素只保留一个。

2.常用命令：sadd,spop,smembers,sunion 等。

## 五、 Sorted Set (有序集合)

1.有序集合是在集合的基础上为每一个元素关联一个分数，Redis 通过分数为集合中的成员进行排序。

2.常用命令：zadd,zrange,zrem,zcard 等。

## 9. Redis 的事务？(2018-4-23-zcz)

一、 Redis 事务允许一组命令在单一步骤中执行。事务有两个属性，说明如下：

- a) 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
- b) Redis 事务是原子的。原子意味着要么所有的命令都执行，要么都不执行；

二、 一个事务从开始到执行会经历以下三个阶段：

- a)开始事务
- b)命令入队
- c)执行事务

## 10. Redis 的使用场景有哪些？(2018-4-23-zcz)

- 1.取最新 N 个数据的操作
- 2.排行榜应用,取 TOP N 操作
- 3.需要精准设定过期时间的应用
- 4.计数器应用
- 5.uniq 操作,获取某段时间所有数据排重值
- 6.Pub/Sub 构建实时消息系统
- 7.构建队列系统
- 8.缓存

## 11. Redis 默认端口，默认过期时间，Value 最多可以容纳的数据 长度？

(2018-4-23-zcz)

- 1.默认端口：6379

2.默认过期时间：可以说永不过期，一般情况下，当配置中开启了超出最大内存限制就写磁盘的话，那么没有设置过期时间的 key 可能会被写到磁盘上。假如没设置，那么 REDIS 将使用 LRU 机制，将内存中的老数据删除，并写入新数据。

3.Value 最多可以容纳的数据长度是：512M。

## 12. Redis 有多少个库？(2018-4-23-zcz)

Redis 一个实例下有 16 个。

# 第十章 人工智能

## 一. 数据挖掘

1. 1G 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。（2018-4-23-xhq）

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、据每个词的引用次数进行排序。冒泡、快排等等都可以。

2. 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。（2018-4-23-xhq）

用 trie 树统计每个词出现的次数，时间复杂度是  $O(n * le)$  ( $le$  表示单词的平均长度)。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  $O(n * \lg 10)$ 。所以总的时间复杂度，是  $O(n * le)$  与  $O(n * \lg 10)$  中较大的哪一个。

详情扫描二维码了解。

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



3. 怎么在海量数据中找出重复次数最多的一个？（2018-4-23-xhq）

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。

参考博客一：

<https://blog.csdn.net/u010601183/article/details/56481868>



参考博客二：

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



4. 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  $2^{32}/8=512\text{M}$  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。

## 二．机器学习

## 三．深度学习

# 第十一章 数据结构与算法

### 1. 算法的特征？(2018-4-16-lxy)

- 1) 有穷性：一个算法必须保证执行有限步骤之后结束；
- 2) 确切性：算法的每一步骤必须有确切的定义；
- 3) 输入：一个算法有 0 个或多个输入，以刻画运算对象的初始情况，所谓 0 个输入是指算法本身给出了初始条件；
- 4) 输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；
- 5) 可行性：算法原则上能够精确地运行，而且人们用笔和纸做有限次数运算后即可完成。

## 2. 冒泡排序的思想？(2018-4-16-lxy)

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  $O(n^2)$ ，空间复杂度为  $O(1)$

```
1. def bubble_sort(alist):
2.     for j in range(len(alist)-1, 0, -1):
3.         # j 表示每次遍历需要比较的次数，是逐渐减小的
4.         for i in range(j):
5.             if alist[i] > alist[i+1]:
6.                 alist[i], alist[i+1] = alist[i+1], alist[i]
7.
8. li = [54, 26, 93, 17, 77, 31, 44, 55, 20]
9. bubble_sort(li)
10. print(li)
```

## 3. 快速排序的思想？(2018-4-16-lxy)

快排的基本思想：通过一趟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。

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  $O(n^2)$ ，最好的时间复杂度是  $O(n \log n)$ ，空间复杂度为  $O(\log n)$

```
13. def quick_sort(alist, start, end):
14.     """快速排序"""
15.     # 递归的退出条件
16.     if start >= end:
17.         return
18.     # 设定起始元素为要寻找位置的基准元素
19.     mid = alist[start]
20.     # low 为序列左边的由左向右移动的游标
```



```
21. low = start
22. # high 为序列右边的由右向左移动的游标
23. high = end
24. while low < high:
25.     # 如果 low 与 high 未重合，high 指向的元素不比基准元素小，则 high 向左移动
26.     while low < high and alist[high] >= mid:
27.         high -= 1
28.     # 将 high 指向的元素放到 low 的位置上
29.     alist[low] = alist[high]
30.     # 如果 low 与 high 未重合，low 指向的元素比基准元素小，则 low 向右移动
31.     while low < high and alist[low] < mid:
32.         low += 1
33.     # 将 low 指向的元素放到 high 的位置上
34.     alist[high] = alist[low]
35. # 退出循环后，low 与 high 重合，此时所指位置为基准元素的正确位置
36. # 将基准元素放到该位置
37. alist[low] = mid
38. # 对基准元素左边的子序列进行快速排序
39. quick_sort(alist, start, low-1)
40. # 对基准元素右边的子序列进行快速排序
41. quick_sort(alist, low+1, end)
42. alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
43. quick_sort(alist, 0, len(alist)-1)
44. print(alist)
```

#### 4. 如何判断单向链表中是否有环？(2018-4-16-lxy)

首先遍历链表，寻找是否有相同地址，借此判断链表中是否有环。如果程序进入死循环，则需要一块空间来存储指针，遍历新指针时将其和储存的旧指针比对，若有相同指针，则该链表有环，否则将这个新指针存下来后继续往下读取，直到遇见 NULL，这说明这个链表无环。

#### 5. 基础的数据结构有哪些？(2018-4-23-lyf)

基本的算法有：排序算法(冒泡排序，插入排序，快速排序，归并排序)，查找(二分查找)，搜索((DFS) 深度优先搜索，(BFS) 广度优先搜索)，(Dijkstra 算法)，动态规划算法，分类(朴素贝叶斯分类算法等)。

评价算法的好坏一般有两种: 时间复杂度和空间复杂度。

时间复杂度: 同样的输入规模(问题规模)花费多少时间。

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。

稳定性: 不会因为输入的不同而导致不稳定的情况发生。

算法的思路是否简单: 越简单越容易实现的越好。

## 6. 基本的算法有哪些，怎么评价一个算法的好坏？(2018-4-23-lyf)

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。

时间复杂度: 同样的输入规模(问题规模)话费多少时间。

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。

稳定性: 不会引文输入的不同而导致不稳定的情况发生。

算法的思路是否简单: 越简单越容易实现的越好。

## 7. 哪种数据结构可以实现递归？(2018-4-23-lyf)

栈可以实现，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，只有栈先进后出的特性才能实现。

## 8. 你知道哪些排序算法（一般是通过问题考算法）(2018-4-23-lyf)

冒泡，选择，快排，归并。

## 9. 斐波那契数列(2018-4-23-lyf)

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。

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

## 10. 二叉树如何求两个叶节点的最近公共祖先？(2018-4-23-lyf)

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数大，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：

```
1. class TreeNode(object):
2.     def __init__(self, left=None, right=None, data=None):
3.         self.data = data
4.         self.left = left
5.         self.right = right
6.     def getCommonAncestor(root, node1, node2):
7.         while root:
8.             if root.data > node1.data and root.data > node2.data:
```

```
9.         root = root.left
10.        elif root.data < node1.data and root.data < node2.data:
11.            root = root.right
12.        else:
13.            return root
14.    return None
```

## 11. 两个字符串，如何求公共字符串？(2018-4-23-lyf)

```
1. def getLCString(str1, str2):
2.     maxlen = len(str1) if len(str1) < len(str2) else len(str2)
3.     example = str1 if len(str1) < len(str2) else str2
4.     other = str2 if str1 == example else str1
5.     for i in range(maxlen):
6.         for j in range(maxlen, 0, -1):
7.             if other.find(example[i:j]) != -1:
8.                 return example[i:j]
```

## 12. 找出二叉树中最远结点的距离？(2018-4-23-lyf)

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。

## 13. 写一个二叉树(2018-4-23-lyf)

```
1. class TreeNode(object):
2.     def __init__(self, left=None, right=None, data=None):
3.         self.data = data
4.         self.left = left
5.         self.right = right
6.     def preorder(root): #前序遍历
7.         if root is None:
8.             return
9.         else:
10.            print root.data
```

```
11.         preorder(root.left)
12.         preorder(root.right)
13.
14.     def inorder(root): #中序遍历
15.         if root is None:
16.             return
17.         else:
18.             inorder(root.left)
19.             print root.data
20.             inorder(root.right)
21.
22.     def postorder(root): # 后序遍历
23.         if root is None:
24.             return
25.         postorder(root.left)
26.         postorder(root.right)
27.         print root.data
28.
```

## 14. 写一个霍夫曼数(2018-4-23-lyf)

```
1. class TreeNode(object):
2.     def __init__(self, left=None, right=None, data=None):
3.         self.data = data
4.         self.left = left
5.         self.right = right
6. # 根据数组 a 中 n 个权值建立一棵哈夫曼树，返回树根指针
7. def CreateHuffman(a):
8.     a.sort() # 对列表进行排序(从小到大)
9.     n = len(a)
10.    b = [TreeNode(data=a[i]) for i in range(n)]
11.    for i in range(n): # 进行 n-1 次循环建立哈夫曼树
12.        [print(temp, end=' ') for temp in b]
13.        #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
14.        k1 = -1
15.        k2 = 0
16.        for j in range(n): # 让 k1 初始指向森林中第一棵树，k2 指向第二棵
17.            if b[j]:
18.                if k1 == -1:
19.                    k1 = j
20.                    continue
21.                k2 = j
22.                break
```

```
23.         print('k1', k1, 'k2', k2)
24.         # print(b[k1].data, b[k2].data)
25.         for j in range(k2, n): # 从当前森林中求出最小权值树和次最小
26.             if b[j]:
27.                 if b[j].data < b[k1].data:
28.                     k2 = k1
29.                     k1 = j
30.             elif b[k2]:
31.                 if b[j].data < b[k2].data:
32.                     k2 = j
33. # 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
34.     if b[k2]:
35.         q = TreeNode()
36.         q.data = b[k1].data + b[k2].data
37.         q.left = b[k1]
38.         q.right = b[k2]
39.         b[k1] = q # 将指向新树的指针赋给 b 指针数组中 k1 位置
40.         b[k2] = None # k2 位置为空
41.         return q # 返回整个哈夫曼树的树根指针
```

## 15. 写一个二分查找(2018-4-23-lyf)

```
1. def binary_search(a, n, key):
2.     m = 0; l = 0; r = n - 1 # 闭区间[0, n - 1]
3.     while (l < r):
4.         m = l + ((r - l) >> 1) # 向下取整
5.         if (a[m] < key): l = m + 1;
6.         else: r = m;
7.
8.     if (a[r] == key): return r;
9.     return -1
```

## 16. set 用 in 时间复杂度是多少，为什么？(2018-4-23-lyf)

O(1)，因为 set 是键值相同的一个数据结构，键做了 hash 处理。

## 17. 深度优先遍历和广度优先遍历的区别？(2018-4-23-lyf)

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) 深度优先搜索算法：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。

18. 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 ;

(2018-4-23-lyf)

```
1. def quick_sort(lista , start , stop):
2.     if start >= stop:
3.         return
4.     low = start
5.     high = stop
6.     mid = lista[start]
7.     while low < high:
8.         while low < high and lista[high] >= mid:
9.             high -= 1
10.        lista[low] = lista[high]
11.        while low < high and mid > lista[low]:
12.            low += 1
13.        lista[high] = lista[low]
14.    lista[low] = mid
15.    quick_sort(lista , start , low-1)
16.    quick_sort(lista , low+1 , stop)
17.    return lista
18. lista = [0 , 1000]
19. print(quick_sort(lista , 0 , len(lista)-1))
```

19. 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？

(2018-4-23-lyf)

```
1. Import stack
2. Class queue(stack):
3.     def __init__(self):
4.         def __init__(self):
5.             stack1 = stack() # 进来的元素都放在里面
6.             stack2 = stack() # 元素都从这里出去
7. def push(self , element):
8.     self.stack1.push(element)
9. def pop(self):
10.    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
11.        while not self.stack1.empty():
12.            self.stack2.push(self.stack1.top())
13.        self.stack1.pop()
```



```
14.         ret = self.stack2.top() # 有元素后就可以弹出了
15.         self.stack2.pop()
16.         return ret
17. def top(self):
18.     if (self.stack2.empty()):
19.         while not self.stack1.empty():
20.             self.stack2.push(self.stack1.top())
21.             self.stack1.pop()
22.     return self.stack2.top()
```

20. 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(2018-4-23-lyf)

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

时间复杂度  $O(n^2)$  空间复杂度  $O(n)$ 。

21. 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）(2018-4-23-lyf)

```
1. class ListNode:
2.     def __init__(self, x):
3.         self.val=x
4.         self.next=None
5.
6. def nonrecurse(head): #循环的方法反转链表
7.     if head is None or head.next is None:
8.         return head
9.     pre=None
10.    cur=head
11.    h=head
12.    while cur:
13.        h=cur
```

```
14.         tmp=cur.next
15.         cur.next=pre
16.         pre=cur
17.         cur=tmp
18.     return h
19.
20. class ListNode:
21.     def __init__(self, x):
22.         self.val=x
23.         self.next=None
24. def recurse(head, newhead): #递归，head 为原链表的头结点，newhead 为
25. 反转后链表的头结点
26.     if head is None:
27.         return
28.     if head.next is None:
29.         newhead=head
30.     else :
31.         newhead=recurse(head.next, newhead)
32.         head.next.next=head
33.         head.next=None
34.     return newhead
```

22. 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）(2018-4-23-lyf)

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

## 23. 桶排序 ( 最快最简单的排序 ) (2018-4-23-lyf)

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。

```
1. def bucketSort(nums):
2.     # 选择一个最大的数
3.     max_num = max(nums)
4.     # 创建一个元素全是 0 的列表 , 当做桶
5.     bucket = [0]*(max_num+1)
6.     # 把所有元素放入桶中 , 即把对应元素个数加一
7.     for i in nums:
8.         bucket[i] += 1
9.     # 存储排序好的元素
10.    sort_nums = []
11.    # 取出桶中的元素
12.    for j in range(len(bucket)):
13.        if bucket[j] != 0:
14.            for y in range(bucket[j]):
15.                sort_nums.append(j)
16.    return sort_nums
17. nums = [5, 6, 3, 2, 1, 65, 2, 0, 8, 0]
18. 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

## 24. 青蛙跳台阶问题(2018-4-23-lyf)

一只青蛙要跳上 n 层高的台阶 , 一次能跳一级 , 也可以跳两级 , 请问这只青蛙有多少种跳上这个 n 层高台阶的方法 ?

思路分析 :

这个问题有三种方法来解决 , 并在下面给出三处方法的 python 实现。

## 方法 1:递归

设青蛙跳上  $n$  级台阶有  $f(n)$  种方法，把这  $n$  种方法分为两大类，第一种最后一次跳了一级台阶，这类方法共有  $f(n-1)$  种，第二种最后一次跳了两级台阶，这种方法共有  $f(n-2)$  种，则得出递推公式  $f(n)=f(n-1)+f(n-2)$ ，显然， $f(1)=1$ ， $f(2)=2$ ，递推公式如下：

\* 这种方法虽然代码简单，但效率低，会超出时间上限\*

代码实现如下：

```
1. class Solution:
2.     # @param {integer} n
3.     # @return {integer}
4.     def climbStairs(self, n):
5.         if n==1:
6.             return 1
7.         elif n==2:
8.             return 2
9.         else:
10.            return self.climbStairs(n-1)+self.climbStairs(n-2)
```

## 方法 2: 用循环来代替递归

这种方法的原理仍然基于上面的公式，但是用循环代替了递归，比上面的代码效率上有较大的提升，可以 AC。

代码实现如下：

```
1. class Solution:
2.     # @param {integer} n
3.     # @return {integer}
4.     def climbStairs(self, n):
5.         if n==1 or n==2:
6.             return n
7.         a=1;b=2;c=3
8.         for i in range(3, n+1):
9.             c=a+b;a=b;b=c
10.            return c
```

## 方法三：建立简单数学模型，利用组合数公式

设青蛙跳上这  $n$  级台阶一共跳了  $z$  次，其中有  $x$  次是一次跳了两级， $y$  次是一次跳了一级，则有  $z=x+y$ ， $2x+y=n$ ，对一个固定的  $x$ ，利用组合可求出跳上这  $n$  级台阶的方法共有种方法又因为  $x$  在区间  $[0, n/2]$  内，所以我们只需要遍历这个区间内所有的整数，求出每个  $x$  对应的组合数累加到最后的的结果即可。

```
1. python 代码实现如下：
2. class Solution:
3.     # @param {integer} n
4.     # @return {integer}
5.     def climbStairs(self, n):
6.         def fact(n):
7.             result=1
8.             for i in range(1, n+1):
9.                 result*=i
10.            return result
11.        total=0
12.        for i in range(n/2+1):
13.            total+=fact(i+n-2*i)/fact(i)/fact(n-2*i)
14.        return total
```

## 25.删除排序数组中的重复数字 Remove Duplicates from Sorted Array(2018-4-23-lyf)

给定一个排序数组，在原数组中删除重复出现的数字，使得每个元素只出现一次，并且返回 新的数组的长度。 不要使用额外的数组空间，必须在原地没有额外空间的条件下完成。

样例：

给出数组  $A = [1,1,2]$ ，你的函数应该返回长度 2，此时  $A = [1,2]$ 。

```
1. class Solution:
2.     # @param a list of integers
3.     # @return an integer
4.     def removeDuplicates(self, A):
5.         if len(A) == 0:
6.             return 0
7.         if len(A) == 1: # to make it clear
8.             return 1     # (without this also works well)
```

```
9.
10.     index = 0
11.     for i in A[1:]:
12.         if i != A[index]:
13.             index += 1
14.             A[index] = i
15.
16.     return index + 1
```

分析：遍历，跳过重复元素。时间复杂度  $O(n)$ 。

思考：其他条件不变，如果可以允许出现两次重复将如何处理？

样例：

给出数组 `nums = [1,1,1,2,2,3]`。

你的函数应当返回长度 `length = 5`，且前 5 个元素分别为 1, 1, 2, 2 and 3。

```
1. class Solution:
2.     # @param a list of integers
3.     # @return an integer
4.     def removeDuplicates(self, A):
5.         if None == A:
6.             return 0
7.         len_A = len(A)
8.         if len_A <= 1:
9.             return len_A
10.
11.         m = 0
12.         n = 1
13.         count = 1
14.         while n < len_A:
15.             if A[m] != A[n]:
16.                 count = 1
17.                 m += 1
18.                 if m != n:
19.                     A[m] = A[n]
20.             elif count >= 2:
21.                 count += 1
22.             else:
23.                 m += 1
24.                 count += 1
25.                 if m != n:
26.                     A[m] = A[n]
```

```
27.         n += 1
28.         A = A[0:m+1]    # A must be modified
29.         return m + 1
```

## 26. 两数之和 Two Sum(2018-4-23-lyf)

给一个整数数组，找到两个数使得他们的和等于一个给定的数 target。

你需要实现的函数 twoSum 需要返回这两个数的下标，并且第一个下标小于第二个下标。注意这里下标的范围是 1 到 n，不是以 0 开头。

样例：

给出 numbers = [2, 7, 11, 15], target = 9, 返回 [1, 2].

分析：

给定一个数列（注意不一定有序），和一个指定的数值 target。从这个数列中找出两个数相加刚好等于 target，要求给出这两个数的下标（注意数列下标是从 1 而不是从 0 开始）。

首先将数列排序。由于最后要得求的是两个数的原有下标，而不是两个数本身，因此要用一个新的对象 Item 来封装原有数列元素，并记录该元素的原有下标。排序是针对数列元素本身数值的，分别用一个 index1 指针和一个 index2 指针指向排序后的数列的首位 如果指向的两个数相加的和等于 target，则搜索结束；如果和小于 target，则由于 index2 此时指向的已经是数组中的最大数了，因此只能令 index1 向右移动一次；如果和大于 target，则由于此时 index1 已经指向数组中的最小数了，因此只能令 index2 向左移动一次。用一个循环重复上述过程，直到和等于 target 宣告搜索成功，或者 index1 >= index2 宣告搜索失败。

```
1. class Solution:
2.
3.     class Item:
4.         def __init__(self, value, index):
5.             self.value = value
6.             self.index = index
7.
```

```
8.      # @return a tuple, (index1, index2)
9.      def twoSum(self, num, target):
10.         len_num = len(num)
11.         if 0 == len_num:
12.             return (-1, -1)
13.
14.         items = [Solution.Item(value, 0) for value in num]
15.         for i in range(0, len_num):
16.             items[i].index = i + 1
17.         items.sort(lambda x, y: cmp(x.value, y.value))
18.         index1 = 0
19.         index2 = len_num - 1
20.         is_find = False
21.         while index1 < index2:
22.             total = items[index1].value + items[index2].value
23.             if total < target:
24.                 index1 += 1
25.             elif total > target:
26.                 index2 -= 1
27.             else:
28.                 is_find = True
29.                 break
30.         (index1, index2) = (index1, index2) if items[index1].index <= items[index2].index else (index2, index1)
31.         return (items[index1].index, items[index2].index) if is_find else (-1, -1)
```

空间复杂度  $O(n)$ ，因为构造了一个新的 Item 序列。时间复杂度方面，如果假设 Python 的 sort 算法是用的快速排序的话，那排序的时间复杂度为  $O(n \cdot \log n)$ ，搜索过程的时间复杂度为  $O(n)$ ，因此总的时间复杂度为  $O(n \cdot \log n)$ 。

注意给的数列也许长度为 0，这样无论 target 是多少都是搜索失败。而且题目中给的 example 明显是在误导人，不仔细看误以为数列原本就有序。此外，数列下标是从 1 而不是从 0 开始的。

但是这种算法的前提要先进行一次排序，看起来总是有点不舒服，是不是有更好的算法呢？

具体扫下面二维码查看：





## 27.删除元素 Remove Element(2018-4-23-lyf)

给定一个数组和一个值，在原地删除与值相同的数字，返回新数组的长度。元素的顺序可以改变，并且对新的数组不会有影响。

样例：

给出一个数组 `[0,4,4,0,0,2,4,4]`，和值 `4`

返回 `4` 并且 `4` 个元素的新数组为`[0,0,0,2]`

分析：

题目已经暗示可以将需要删除的元素移至数组末尾，因此用两个下标 `m` 和 `n`，`m` 用来从左至右遍历数组寻找该元素，`n` 从右至左记录可以用来交换的尾部位置。

```
1. def removeElement(A,elem):
2.     if None == A:
3.         return A
4.     len_A = len(A)
5.     m = 0
6.     n = len_A-1
7.     while m<=n:
8.         if elem == A[m]:
9.             if elem!=A[n]:
```

```
10.         A[m],A[n] = A[n],A[m]
11.         m+=1
12.         n-=1
13.     else:
14.         n-=1
15.     else:
16.         m+=1
17.     for i in range(n+1):
18.         A.pop()
19.     return A
```

## 28.加一 Plus One(2018-4-23-lyf)

给一个由包含一串数字的列表组成的非负整数加上一。

注意点：列表前面的数字表示高位，注意最高位也可能进位。

例子：

输入: [1, 2, 3, 4, 9]

输出: [1, 2, 3, 5, 0]

解题思路：

用一个 int 数组代表一个数值，最左边是最高位，现要求加上 1，同样地返回一个 int 数组代表计算结果。非常简单的模拟加法，注意进位问题，如果是 99，999 这种，最后加上 1 后还要在数组最左边插入一个元素 1。

```
1. class Solution:
2.     # @param digits, a list of integer digits
3.     # @return a list of integer digits
4.     def plusOne(self, digits):
5.         len_s = len(digits)
6.         carry = 1
7.         for i in range(len_s - 1, -1, -1):
8.             total = digits[i] + carry
9.             digit = int(total % 10)
10.            carry = int(total / 10)
11.            digits[i] = digit
```

```
12.     if 1 == carry:
13.         digits.insert(0, 1)
14.     return digits
```

## 29. 用两个队列如何实现一个栈，用两个栈如何实现一个队列？(2018-4-23-lyf)

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。

## 30. 爬楼梯 Climbing Stairs(2018-4-23-lyf)

假设你正在爬楼梯，需要  $n$  步你才能到达顶部。但每次你只能爬一步或者两步，你能有多少种不同的方法爬到楼顶部？

样例：

比如  $n=3$ ， $1+1+1=1+2=2+1=3$ ，共有 3 中不同的方法

返回 3

解题思路：

如果按照从右至左的逆序递归求解，其实就相当于搜索算法了，会造成子搜索过程的重复计算。搜索算法一般都可以用动态规划来替代，因此这里就用 1D 动态规划。

然后可以发现， $f(x)$  的求解只依赖于  $f(x-1)$  和  $f(x-2)$ ，因此可以将空间复杂度缩小到  $\text{int}[3]$ 。于是你就会发现，这其实就是一个斐波拉契数列问题。

```
1. class Solution:
2.     # @param n, an integer
3.     # @return an integer
4.     def climbStairs(self, n):
5.         if n <= 1:
6.             return 1
7.         arr = [1, 1, 0]      # look here, arr[0] = 1, arr[1] = 2
8.         for i in range(2, n + 1):
9.             arr[2] = arr[0] + arr[1]
10.            arr[0], arr[1] = arr[1], arr[2]
11.            return arr[2]
```

### 31. 落单的数 Single Number(2018-4-23-lyf)

给出  $2*n + 1$  个的数字，除其中一个数字之外其他每个数字均出现两次，找到这个数字。

样例：

给出  $[1,2,2,1,3,4,3]$ ，返回 4。

解题思路：

异或操作，知道的人立马能做出来，不知道的人想破脑袋也想不出这个方法。当然用 hashmap/map 之类的把所有元素插一遍也能找出这个只出现过一次的元素，但是想必面试官不会很开心。位操作还是有很多技巧的，还需要继续深入学习。

```
1. class Solution:
2.     # @param A, a list of integer
3.     # @return an integer
```

```
4. def singleNumber(self, A):
5.     len_A = len(A)
6.     if 0 == len_A:
7.         return 0
8.     elif 1 == len_A:
9.         return A[0]
10.    else:
11.        result = A[0]
12.        for i in range(1, len_A):
13.            result ^= A[i]
14.        return result
```

### 32. 搜索旋转排序数组 Search in Rotated Sorted Array(2018-4-23-lyf)

假设有一个排序的按未知的旋转轴旋转的数组(比如，0 1 2 4 5 6 7 可能成为 4 5 6 7 0 1 2)。给定一个目标值进行搜索，如果在数组中找到目标值返回数组中的索引位置，否则返回-1。

你可以假设数组中不存在重复的元素。

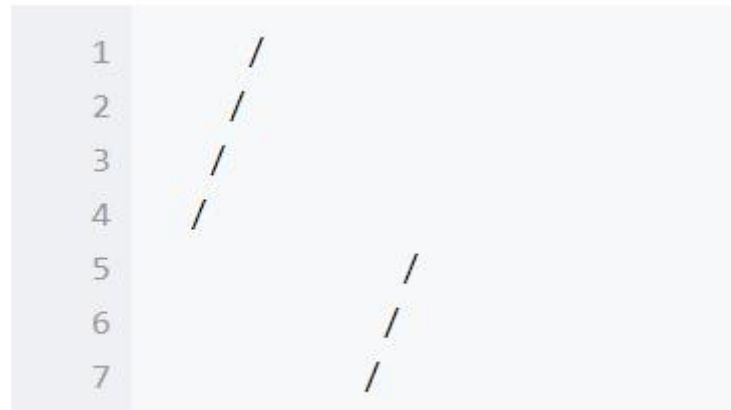
样例：

给出[4, 5, 1, 2, 3]和 target=1，返回 2

给出[4, 5, 1, 2, 3]和 target=0，返回 -1

解题思路：

采用了二分搜索，不过旋转后的数组要讨论的情况增多了。其实旋转后的数组的大小关系一般如下图所示：先通过中点与左顶点的关系来分类讨论中点落在了哪一部分，如果在左半边，则继续讨论目标数在中点的左边还是右边；如果在右半边，同样讨论目标数的位置。同时需要注意特殊情况只剩下两个数时，例如[3,1]，这时求出的中点也是 3，如果中点不匹配，应考虑 1。这种情况不好与上面的情况合并，单独列出。



```

1. class Solution:
2.
3.     def doSearch(self, A, left, right, target):
4.         len_A = right - left + 1
5.         if 0 == len_A:
6.             return -1
7.         elif 1 == len_A:    ###
8.             return left if target == A[left] else -1
9.         if A[left] < A[right]:
10.            while left <= right:
11.                mid = (left + right) / 2
12.                if A[mid] < target:
13.                    left = mid + 1
14.                elif A[mid] > target:
15.                    right = mid - 1
16.                else:
17.                    return mid
18.            else:
19.                mid = (left + right) / 2
20.                left_result = self.doSearch(A, left, mid, target)
21.                if -1 != left_result:
22.                    return left_result
23.            else:
24.                right_result = self.doSearch(A, mid + 1, right, target)
25.                if -1 != right_result:
26.                    return right_result
27.            return -1
28.
29.     # @param A, a list of integers
30.     # @param target, an integer to be searched
31.     # @return an integer
32.     def search(self, A, target):

```

```
33.         len_A = len(A)
34.         return self.doSearch(A, 0, len(A) - 1, target)
```

思考：

其他条件不变，假如有重复元素又将如何？是否会影响运行时间复杂度？如何影响？为何会影响？

写出一个函数判断给定的目标值是否出现在数组中。

样例：

给出[3,4,4,5,7,0,1,2]和 target=4，返回 true。

```
1. class Solution(object):
2.     def search(self, nums, target):
3.         """
4.         :type nums: List[int]
5.         :type target: int
6.         :rtype: int
7.         """
8.         left = 0
9.         right = len(nums) - 1
10.        while left <= right:
11.            mid = left + (right - left) // 2
12.            if nums[mid] == target:
13.                return True
14.            if nums[mid] > target:
15.                if nums[left] <= target or nums[mid] < nums[left]:
16.                    right = mid - 1
17.                else:
18.                    left = mid + 1
19.            else:
20.                if nums[left] > target or nums[mid] >= nums[left]:
21.                    left = mid + 1
22.                else:
23.                    right = mid - 1
24.        return False
25.
26.
27. if __name__ == "__main__":
28.     assert Solution().search([4, 5, 5, 6, 7, 0, 1, 2], 4) == True
29.     assert Solution().search([4, 5, 6, 7, 7, 7, 7, 0, 1, 2], 7) == True
```

### 33.三数之和 3Sum(2018-4-23-lyf)

给出一个有  $n$  个整数的数组  $S$ ，在  $S$  中找到三个整数  $a, b, c$ ，找到所有使得  $a + b + c = 0$  的三元组。

注意事项：在三元组  $(a, b, c)$ ，要求  $a \leq b \leq c$ ，结果不能包含重复的三元组。

样例：

如  $S = \{-1, 0, 1, 2, -1, -4\}$ ，你需要返回的三元组集合的是：

$(-1, 0, 1)$ 、 $(-1, -1, 2)$

解题思路：

求一个列表中所有和为零的二元组的一种思路是先把列表排序，再用两个指针从两头向中间移动。如果前后两个数的和小于 0，则左指针右移；如果和大于 0，则右指针左移。求三元组时可以参考这种做法，第一个数  $a$  确定后，可以理解为求列表中和为  $-a$  的二元组。由于不要考虑重复的元组，遇到重复的数可以直接跳过。

```
1. class Solution(object):
2.     def threeSum(self, nums):
3.         """
4.         :type nums: List[int]
5.         :rtype: List[List[int]]
6.         """
7.         # Sorted array can save a lot of time
8.         nums.sort()
9.         result = []
10.        i = 0
11.        while i < len(nums) - 2:
12.            j = i + 1
13.            k = len(nums) - 1
14.            while j < k:
15.                l = [nums[i], nums[j], nums[k]]
16.                if sum(l) == 0:
17.                    result.append(l)
18.                    j += 1
19.                    k -= 1
```



```
20.             # Ignore repeat numbers
21.             while j < k and nums[j] == nums[j - 1]:
22.                 j += 1
23.             while j < k and nums[k] == nums[k + 1]:
24.                 k -= 1
25.             elif sum(l) > 0:
26.                 k -= 1
27.             else:
28.                 j += 1
29.             i += 1
30.             # Ignore repeat numbers
31.             while i < len(nums) - 2 and nums[i] == nums[i - 1]:
32.                 i += 1
33.             return result
34.
35. if __name__ == "__main__":
36.     assert Solution().threeSum([-1, 0, 1, 2, -1, -4]) == [[-1, -1, 2], [-1, 0, 1]]
```

### 34. 四数之和 4Sum(2018-4-23-lyf)

给一个包含  $n$  个数的整数数组  $S$ ，在  $S$  中找到所有使得和为给定整数  $target$  的四元组  $(a, b, c, d)$ 。

注意事项：

四元组  $(a, b, c, d)$  中，需要满足  $a \leq b \leq c \leq d$ 。

答案中不可以包含重复的四元组。

样例：

例如，对于给定的整数数组  $S=[1, 0, -1, 0, -2, 2]$  和  $target=0$ 。满足要求的四元组集合为：

$(-1, 0, 0, 1)$

$(-2, -1, 1, 2)$

$(-2, 0, 0, 2)$

解题思路：

想参照之前的一题 3Sum 的解法来解决，结果超时了。换个思路，空间换时间。既然有四个数，那就把前两个数之和和后两个数之和分别当做一个数。现在要求的就是在一个列表中哪两个数的和为特定值。可以先遍历一遍列表，存值和它的下标的键值对，第二遍遍历列表寻找（目标值-当前值）是否在之前的键值对中，如果在就是符合的一组解。

```
1. class Solution(object):
2.     def fourSum(self, nums, target):
3.         """
4.         :type nums: List[int]
5.         :type target: int
6.         :rtype: List[List[int]]
7.         """
8.         if len(nums) < 4:
9.             return []
10.        result = set()
11.        sumsIndexes = {}
12.        # Get all two elements' sum and indexes map
13.        for i in range(len(nums)):
14.            for j in range(i + 1, len(nums)):
15.                if nums[i] + nums[j] in sumsIndexes:
16.                    sumsIndexes[nums[i] + nums[j]].append((i, j))
17.                else:
18.                    sumsIndexes[nums[i] + nums[j]] = [(i, j)]
19.
20.        for i in range(len(nums)):
21.            for j in range(i + 1, len(nums)):
22.                sumNeeded = target - (nums[i] + nums[j])
23.                if sumNeeded in sumsIndexes:
24.                    for index in sumsIndexes[sumNeeded]:
25.                        # Ignore repeating results
26.                        if index[0] > j:
27.                            result.add(tuple(sorted([nums[i], nums[j], nums[index[0]], nums[index[1]]])))
28.        # Format result with list[list] pattern
29.        result = [list(l) for l in result]
30.        return result
31. if __name__ == "__main__":
32.     assert Solution().fourSum([1, 0, -1, 0, -2, 2], 0) == [[-1, 0, 0, 1], [-2, 0, 0, 2], [-2, -1, 1, 2]]
```

### 35. 下一个排列 Next Permutation(2018-4-23-lyf)

给定一个整数数组来表示排列，找出其之后的一个排列。

注意事项：排列中可能包含重复的整数。

样例：

给出排列[1,3,2,3]，其下一个排列是[1,3,3,2]

给出排列[4,3,2,1]，其下一个排列是[1,2,3,4]

解题思路：

通过一个例子来说明，原数组为[1,7,3,4,1]。我们要找到比 173421 大一点的数，那就要优先考虑将后面的数字变换顺序，而 421 从后往前是升序的（也就是这三个数字能组成的最大的数），变换了反而会变小，所以要先找到降序的点。可以看出 3 是第一个降序的点，要想整个数变大，3 就要变大，从后往前找到第一个比 3 大的数 4，将 3 和 4 交换位置得到 174321，既然原来 3 所在位置的数字变大了，那整个数肯定变大了，而它之后的数是最大的（从后往前是升序的），应转换成最小的，直接翻转。

```
1. class Solution(object):
2.     def nextPermutation(self, nums):
3.         """
4.         :type nums: List[int]
5.         :rtype: void Do not return anything, modify nums in-place instead.
6.         """
7.         length = len(nums)
8.
9.         targetIndex = 0
10.        changeIndex = 0
11.        for i in range(length - 1, 0, -1):
12.            if nums[i] > nums[i - 1]:
13.                targetIndex = i - 1
14.                break
15.        for i in range(length - 1, -1, -1):
16.            if nums[i] > nums[targetIndex]:
17.                changeIndex = i
18.                break
19.        nums[targetIndex], nums[changeIndex] = nums[changeIndex], nums[targetIndex]
```

```
20.         if targetIndex == changeIndex == 0:
21.             nums.reverse()
22.         else:
23.             nums[targetIndex + 1:] = reversed(nums[targetIndex + 1:])
```

### 36.第 K 个排列 Permutation Sequence(2018-4-23-lyf)

给定  $n$  和  $k$ ，求  $123..n$  组成的排列中的第  $k$  个排列。

注意事项： $1 \leq n \leq 9$ 。

样例：

对于  $n = 3$ ，所有的排列如下：

```
123
132
213
231
312
321
```

解题思路：

因为  $n$  个不同的数字可以组成  $n!$  个序列，那么首位确定的序列都有  $(n-1)!$  种不同的可能性，而且这些序列都根据首位的大小进行了分组， $1...$  是最小的  $(n-1)!$  个， $2...$  是  $(n-1)!+1$  到  $2(n-1)!$  个，那么现在只需要计算  $k$  中有几个  $(n-1)!$  就可以确定首位的数字，同样可以通过这样的方法来确定第二位、第三位.....

此外，由于列表下标从 0 开始，所以  $k$  要减去 1。

```
1. class Solution(object):
2.     def getPermutation(self, n, k):
3.         """
4.         :type n: int
5.         :type k: int
6.         :rtype: str
7.         """
```

```
8.         k -= 1
9.         factorial = 1
10.        for i in range(1, n):
11.            factorial *= i
12.
13.        result = []
14.        array = list(range(1, n + 1))
15.        for i in range(n - 1, 0, -1):
16.            index = k // factorial
17.            result.append(str(array[index]))
18.            array = array[:index] + array[index + 1:]
19.            k %= factorial
20.            factorial //= i
21.        result.append(str(array[0]))
22.        return "".join(result)
23.
24.
25. if __name__ == "__main__":
26.     assert Solution().getPermutation(3, 3) == "213"
27.     assert Solution().getPermutation(9, 324) == "123685974"
```

### 37.判断数独是否合法 Valid Sudoku(2018-4-23-lyf)

请判定一个数独是否有效。该数独可能只填充了部分数字，其中缺少的数字用 . 表示。

注意事项：

一个合法的数独（仅部分填充）并不一定是可解的。我们仅需使填充的空格有效即可。

说明：什么是 数独？（详细解释可以扫下面二维码连接）。



样例：

The following partially filled sudoku is valid

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

解题思路：

用三个列表表示横向、纵向和小正方形的情况。特别需要注意的是小正方形内的元素的表示方法：

`board[i/3*3+j/3][i%3*3+j%3]`。

```
1. class Solution(object):
2.     def isValidSudoku(self, board):
3.         """
```

```
4.         :type board: List[List[str]]
5.         :rtype: bool
6.         """
7.         point = "."
8.         for i in range(9):
9.             row = []
10.            column = []
11.            square = []
12.            for j in range(9):
13.                element = board[i][j]
14.                if element != point:
15.                    if element in row:
16.                        return False
17.                else:
18.                    row.append(element)
19.
20.                element = board[j][i]
21.                if element != point:
22.                    if element in column:
23.                        return False
24.                else:
25.                    column.append(element)
26.
27.                element = board[i // 3 * 3 + j // 3][i % 3 * 3 + j % 3]
28.                if element != point:
29.                    if element in square:
30.                        return False
31.                else:
32.                    square.append(element)
33.        return True
```

### 38. 链表求和 Add Two Numbers(2018-4-23-lyf)

你有两个用链表代表的整数，其中每个节点包含一个数字。数字存储按照在原来整数中相反的顺序，使得第一个数字位于链表的开头。写出一个函数将两个整数相加，用链表形式返回和。

样例：

输入: (2 -> 4 -> 3) + (5 -> 6 -> 4)，返回 7 -> 0 -> 8。

解题思路：

把两个链表当成是相同长度的，短的那个想象成后面都是 0；如果进位的话，在初始化下一个节点的时候将其赋值为 1 即可，所以在计算当前节点的值时要加上自己本来的值。

```
1. class ListNode(object):
2.     def __init__(self, x):
3.         self.val = x
4.         self.next = None
5.
6.     # Define this to check if it works well
7.     def myPrint(self):
8.         print(self.val)
9.         if self.next:
10.             self.next.myPrint()
11.
12.
13. class Solution(object):
14.     def addTwoNumbers(self, l1, l2):
15.         """
16.         :type l1: ListNode
17.         :type l2: ListNode
18.         :rtype: ListNode
19.         """
20.         result = ListNode(0);
21.         cur = result;
22.         while l1 or l2:
23.             cur.val += self.addTwoNodes(l1, l2)
24.             if cur.val >= 10:
25.                 cur.val -= 10
26.                 cur.next = ListNode(1)
27.             else:
28.                 # Check if there is need to make the next node
29.                 if l1 and l1.next or l2 and l2.next:
30.                     cur.next = ListNode(0)
31.             cur = cur.next
32.             if l1:
33.                 l1 = l1.next
34.             if l2:
35.                 l2 = l2.next
36.         return result
37.
38.     def addTwoNodes(self, n1, n2):
39.         if not n1 and not n2:
40.             # This cannot happen, ignore it
```



```
41.         None
42.         if not n1:
43.             return n2.val
44.         if not n2:
45.             return n1.val
46.         return n1.val + n2.val
47.
48.
49. if __name__ == "__main__":
50.     list = ListNode(9)
51.     list.next = ListNode(8)
52.     print(Solution().addTwoNumbers(list, ListNode(1)).myPrint())
```

### 39. 链表划分 Partition List(2018-4-23-lyf)

给定一个单链表和数值  $x$ ，划分链表使得所有小于  $x$  的节点排在大于等于  $x$  的节点之前。你应该保留两部分内链表节点原有的相对顺序。

解题思路：

看成有一串珠子，有红和蓝两种颜色，现在要把红色和蓝色分别集中到一起。可以遍历每个珠子，如果是蓝色就串在一条线上，红色的串在另一条线上，最后把两条线连起来就可以了。注意，在比较大的那串数中，最后的指针要置为 `None`，因为那是排序后的最后一个节点。

```
1. # Definition for singly-linked list.
2. class ListNode(object):
3.     def __init__(self, x):
4.         self.val = x
5.         self.next = None
6.
7.     def to_list(self):
8.         return [self.val] + self.next.to_list() if self.next else [self.val]
9.
10.
11. class Solution(object):
12.     def partition(self, head, x):
13.         """
14.         :type head: ListNode
15.         :type x: int
16.         :rtype: ListNode
```

```
17.     """
18.     dummy = ListNode(-1)
19.     dummy.next = head
20.     small_dummy = ListNode(-1)
21.     large_dummy = ListNode(-1)
22.
23.     prev = dummy
24.     small_prev = small_dummy
25.     large_prev = large_dummy
26.     while prev.next:
27.         curr = prev.next
28.         if curr.val < x:
29.             small_prev.next = curr
30.             small_prev = small_prev.next
31.         else:
32.             large_prev.next = curr
33.             large_prev = large_prev.next
34.         prev = prev.next
35.     large_prev.next = None
36.     small_prev.next = large_dummy.next
37.     return small_dummy.next
38.
39.
40. if __name__ == "__main__":
41.     n1 = ListNode(1)
42.     n2 = ListNode(4)
43.     n3 = ListNode(3)
44.     n4 = ListNode(2)
45.     n5 = ListNode(5)
46.     n6 = ListNode(2)
47.     n1.next = n2
48.     n2.next = n3
49.     n3.next = n4
50.     n4.next = n5
51.     n5.next = n6
52.     r = Solution().partition(n1, 3)
53.     assert r.to_list() == [1, 2, 2, 4, 3, 5]
```

#### 40. 如何翻转一个单链表 ? (2018-4-23-lyf)

```
1. #!/usr/bin/env python
2. #coding = utf-8
3. class Node:
```

```
4.     def __init__(self, data=None, next = None):
5.         self.data = data
6.         self.next = next
7.
8.     def rev(link):
9.         pre = link
10.        cur = link.next
11.        pre.next = None
12.        while cur:
13.            temp = cur.next
14.            cur.next = pre
15.            pre = cur
16.            cur = temp
17.        return pre
18.
19. if __name__ == '__main__':
20.     link = Node(1, Node(2, Node(3, Node(4, Node(5, Node(6, Node(7, Node(8, Node(9))))))))
21.     root = rev(link)
22.     while root:
23.         print(root.data)
24.         root = root.next
```

## 41. 旋转链表(2018-4-23-lyf)

给定一个链表，旋转链表，使得每个节点向右移动  $k$  个位置，其中  $k$  是一个非负数。

样例：

给出链表 1->2->3->4->5->null 和  $k=2$  返回 4->5->1->2->3->null

```
1. class Solution:
2.     # @param head, a ListNode
3.     # @param k, an integer
4.     # @return a ListNode
5.     def rotateRight(self, head, k):
6.         if None == head or None == head.next:
7.             return head
8.         cur_head = head
9.         cur = head
10.        while k > 0:
11.            cur = cur.next
12.            if None == cur:
13.                return head
```

```
14.         k -= 1
15.         cur_tail = cur
16.         cur_head = cur.next
17.         if None == cur_head:
18.             return head
19.         cur = cur_head
20.         while None != cur.next:
21.             cur = cur.next
22.         cur.next = head
23.         cur_tail.next = None
24.     return cur_head
```

## 42. 两两交换链表中的节点(2018-4-23-lyf)

例子：

输入: head = 1->2->3->4 输出: 2->1->4->3

解题思路：

比较常见的链表操作。下面看一下典型情况，如要交换链表中 A->B->C->D 中的 B 和 C 需要做如下操作：

将 A 指向 C；

将 B 指向 D；

将 C 指向 B；

在头节点之前加一个假节点就可以使所有的交换都符合上面的情况。

```
1. # Definition for singly-linked list.
2. class ListNode(object):
3.     def __init__(self, x):
4.         self.val = x
5.         self.next = None
6.
7.
8. class Solution(object):
9.     def swapPairs(self, head):
10.         """
11.         :type head: ListNode
```

```

12.         :rtype: ListNode
13.         """
14.         prev = ListNode(-1)
15.         prev.next = head
16.         temp = prev
17.         while temp.next and temp.next.next:
18.             node1 = temp.next
19.             node2 = temp.next.next
20.             temp.next = node2
21.             node1.next = node2.next
22.             node2.next = node1
23.             temp = temp.next.next
24.         return prev.next

```

### 43. 重排链表(2018-4-23-lyf)

给定一个单链表  $L: L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$ 。

重新排列后为： $L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

必须在不改变节点值的情况下进行原地操作。

例子:

输入: {1,2,3,4} , 输出: {1,4,2,3}

解题思路：

由于是一个单向链表，从尾部不断取元素比较困难，所以我们要将链表反转，又因为是从两头取元素，所以我们只需要反转后半段链表。我们先通过快慢指针来获得链表的中间节点，并将链表截断。接着翻转后半段链表，最后依次从两个链表中提取元素进行连接。需要注意的是，在截断时注意哪个链表比较长，合并的时候不要遗漏元素。

```

1. """
2. Definition of ListNode
3. class ListNode(object):
4.
5.     def __init__(self, val, next=None):
6.         self.val = val
7.         self.next = next

```

```
8. """
9. class Solution:
10.     """
11.     @param head: The first node of the linked list.
12.     @return: nothing
13.     """
14.     def reorderList(self, head):
15.         if head is None or head.next is None:
16.             return head
17.         slow, fast = head, head.next
18.         # 找到中间节点
19.         while fast and fast.next:
20.             fast = fast.next.next
21.             slow = slow.next
22.         # 以中间节点为界，将链表断开，分成前后两部分
23.         # 断开的目的是方便链表翻转操作
24.         cur = slow.next
25.         slow.next = None
26.         # 单链表逆置
27.         while cur:
28.             temp = cur
29.             cur = cur.next
30.             temp.next = slow.next
31.             slow.next = temp
32.         # 再次断开成两个链表，合并
33.         second = slow.next
34.         slow.next = None
35.         pre = head
36.         while second:
37.             temp = second
38.             second = second.next
39.             temp.next = pre.next
40.             pre.next = temp
41.             pre = temp.next
42.         return head
```

## 第十二章 企业真题实战

### 一、360 面试题

1. 请拿出 B 表中的 accd , (A 表中和 B 表中一样的数据) ? (2018-4-16-lxy)

```
Error 1146 (42S02): Table 'acc' doesn't exist
mysql> select * from A;
+----+-----+
| id | name |
+----+-----+
| 1  | a    |
| 2  | b    |
| 3  | c    |
| 4  | c    |
| 5  | d    |
+----+-----+
```

```
mysql> select * from B;
+----+-----+
| id | name |
+----+-----+
| 1  | a    |
| 2  | c    |
| 3  | d    |
| 4  | e    |
| 5  | d    |
+----+-----+
```

1. select \* from B inner join on B.name = A.name

2. a = "abbbccc" , 用正则匹配为 abccc,不管有多少 b , 就出现一次 ?

(2018-4-16-lxy)

1. 思路 : 不管有多少个 b 替换成一个  
2. re.sub(r'b+', 'b', a)

3. xpath 使用的什么库 ? (2018-4-16-lxy)

1. lxml

#### 4. py2 和 py3 的区别 ? (2018-4-16-lxy)

详情查看本文第二章第一部分第 6.5 节内容。

#### 5. Redis 里面 list 内容的长度 ? (2018-4-16-lxy)

```
1. len key_name
```

#### 6. 多线程交互，访问数据，如果访问到了就不访问了，怎么避免重读？

(2018-4-16-lxy)

创建一个已访问数据列表，用于存储已经访问过的数据，并加上互斥锁，在多线程访问数据的时候先查看数据是否已经在已访问的列表中，若已存在就直接跳过。

#### 7. Mysql 怎么限制 IP 访问 ? (2018-4-16-lxy)

grant all privileges on . to '数据库中用户名' @' ip 地址' identified by '数据库密码' ;

#### 8. 带参数的装饰器? (2018-4-16-lxy)

带定长参数的装饰器。

```
1. def new_func(func):
2.     def wrappedfun(username,passwd):
3.         if username == 'root' and passwd == '123456789':
4.             print('通过认证！')
5.             print('开始执行附加功能')
6.             return func()
7.         else:
8.             print('用户名或密码错误')
9.             return
10.    return wrappedfun
11.
12. @new_func
13. def origin():
```



```
14.     print('开始执行函数')
15.     orign('root','123456789')
```

带不定长参数的装饰器。

```
1. def new_func(func):
2.     def wrappedfun(*parts):
3.         if parts:
4.             counts = len(parts)
5.             print('本系统包含 ', end='')
6.             for part in parts:
7.                 print(part, ' ', end='')
8.             print('等', counts, '部分')
9.             return func()
10.        else:
11.            print('用户名或密码错误')
12.            return func()
13.
14.    return wrappedfun
15.
16. @new_func
17. def orign():
18.     print('开始执行函数')
19.     orign('硬件', '软件', '用户数据')
```

同时带不定长、关键字参数的装饰器。

```
1. def new_func(func):
2.     def wrappedfun(*args,**kwargs):
3.         if args:
4.             counts = len(args)
5.             print('本系统包含 ',end='')
6.             for arg in args:
7.                 print(arg,' ',end='')
8.             print('等',counts,'部分')
9.             if kwargs:
10.                for k in kwargs:
11.                    v= kwargs[k]
12.                    print(k,'为：',v)
13.            return func()
14.        else:
15.            if kwargs:
16.                for kwarg in kwargs:
17.                    print(kwarg)
18.                    k,v = kwarg
```

```
19.             print(k, '为 : ', v)
20.             return func()
21.     return wrappedfun
22.
23. @new_func
24. def orign():
25.     print('开始执行函数')
26.
27. orign('硬件', '软件', '用户数据', 总用户数=5, 系统版本='CentOS 7.4')
```

## 二、妙计旅行面试题

### 1. Python 主要的内置数据类型有哪些？(2018-4-16-lxy)

Python 主要的内置数据类型有：str，int，float，tuple，list，dict，set。

### 2. print(dir('a'))输出的是什么？(2018-4-16-lxy)

会打印出字符型的所有的内置方法。

```
1. ['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'_formatter_field_name_split', '_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

### 3. 给定两个 list，A 和 B，找出相同元素和不同元素？(2018-4-16-lxy)

A、B 中相同元素：print(set(A)&set(B))

A、B 中不同元素：print(set(A)^set(B))

### 4. 请反转字符串？(2018-4-16-lxy)

```
1. new_str = old_str[::-1]
```

## 5. 交换变量 a,b 的值 ? (2018-4-16-lxy)

```
1. a,b = b,a
```

## 6. 用 select 语句输出每个城市中心距离市中心大于 20km 酒店数 ? (2018-4-16-lxy)

```
1. select count ( hotel ) i from hotel_table where distance >20 group by city
```

## 7. 给定一个有序列表，请输出要插入值 k 所在的索引位置 ? (2018-4-16-lxy)

```
1. def index(list, key)
2.     if key < list[0]:
3.         position = 0
4.     elif key > list[-1]:
5.         position = len(list)-1
6.     else:
7.         for index in range(list):
8.             if key>list[index] and list[index]>key:
9.                 position = index
```

## 8. 正则表达式贪婪与非贪婪模式的区别 ? (2018-4-16-lxy)

在形式上非贪婪模式有一个 “?” 作为该部分的结束标志。

在功能上贪婪模式是尽可能多的匹配当前正则表达式，可能会包含好几个满足正则表达式的字符串，非贪婪模式，在满足所有正则表达式的情况下尽可能少的匹配当前正则表达式。

## 9. 写出开头匹配字母和下划线，末尾是数字的正则表达式 ? (2018-4-16-lxy)

```
1. ^[A-Za-z]_.*\d$
```

## 10. 请说明 HTTP 状态码的用途 ,请说明常见的状态码机器意义 ? (2018-4-16-lxy)

通过状态码告诉客户端服务器的执行状态，以判断下一步该执行什么操作。

常见的状态机器码有：

100-199：表示服务器成功接收部分请求，要求客户端继续提交其余请求才能完成整个处理过程。

200-299：表示服务器成功接收请求并已完成处理过程，常用 200（OK 请求成功）。

300-399：为完成请求，客户需要进一步细化请求。302（所有请求页面已经临时转移到新的 url），304、307（使用缓存资源）。

400-499：客户端请求有错误，常用 404（服务器无法找到被请求页面），403（服务器拒绝访问，权限不够）。

500-599：服务器端出现错误，常用 500（请求未完成，服务器遇到不可预知的情况）。

11. 当输入 `http://www.itheima.com` 时，返回页面的过程中发生了什么？

(2018-4-16-lxy)

- 1)浏览器向 DNS 服务器发送 itheima.com 域名解析请求；
- 2)DNS 服务器返回解析后的 ip 给客户端浏览器，浏览器向该 ip 发送页面请求；
- 3)DNS 服务器接收到请求后，查询该页面，并将页面发送给客户端浏览器；
- 4)客户端浏览器接收到页面后，解析页面中的引用，并再次向服务器发送引用资源请求；
- 5)服务器接收到资源请求后，查找并返回资源给客户端；
- 6)客户端浏览器接收到资源后，渲染，输出页面展现给用户。

12. 有一个多层嵌套列表 `A=[1,2,[3.4["434",[...]]]]` 请写一段代码遍历 A 中的每一个元素并打印出来。(2018-5-2-xhq)

思路：就是有几个嵌套链表就用几个 for 循环进行迭代,然后对最后一个结果进行打印。

```
1. a= ["a","b",["1","3"],["4","haha"]]
2. for b in a:
3.     for c in b:
4.         for d in c:
5.             print(d)
```

13. 关系型数据库中，表和表之间有左连接，内连接，外连接，分别解释下他们的含义和区别？(2018-5-2-xhq)

内连接查询：查询的结果为两个表匹配到的数据。

右接查询：查询的结果为两个表匹配到的数据，右表特有的数据，对于左表中不存在的数据使用 null 填充。



左连接查询：查询的结果为两个表匹配到的数据，左表特有的数据，对于右表中不存在的数据使用 null 填充。



## 14. 如何定时启动你的爬虫项目：( 2018-5-2-xhq )

### 1. 最简单的方法：直接使用 Timer 类

```
1. import time
2. import os
3.
4. while True:
5.     os.system("scrapy crawl News")
6.     time.sleep(86400) #每隔一天运行一次 24*60*60=86400s
7.
```

### 2. 使用 sched

```
1. import sched
2. #初始化 sched 模块的 scheduler 类
3. #第一个参数是一个可以返回时间戳的函数，第二个参数可以在定时未到达之前阻塞。
4. schedule = sched.scheduler ( time.time, time.sleep )
5.
6. #被周期性调度触发的函数
7. def func():
8.     os.system("scrapy crawl News")
9. def perform1(inc):
10.     schedule.enter(inc,0,perform1,(inc,))
11.     func() # 需要周期执行的函数
12. def mymain():
13.     schedule.enter(0,0,perform1,(86400,))
14.
15. if __name__ == "__main__":
16.     mymain()
17.     schedule.run() # 开始运行，直到计划时间队列变成空为止
```

### 3. 使用 Crontab

参考博客：<https://blog.csdn.net/gaoshanliushui131/article/details/72721704>



### 15. 什么是 scrapy-redis 中的指纹,是如何去重的？（2018-5-2-xhq）

指纹：通过 sha1 加密，把请求体，请求方式，请求 url 放在一起。然后进行 16 进制的转义字符串生成指纹。生成一个字符串，放到数据库中作为唯一标示。

去重：url 中按照 url 去重：1.按照 url 去重，有一个列表，发送请求之前从数据表中看一下这个 url 有没有请求过，请求过了就不用看了 2，内容判断，从数据库中查数据的表示，如果请求过了就在不在请求了。

### 16. 代码优化从哪些方面考虑？有什么想法？（2018-5-2-xhq）

- 1.优化算法时间复杂度。
- 2.减少冗余数据。
- 3.合理使用 copy 与 deepcopy。
- 4.使用 dict 或 set 查找元素。

- 5.合理使用生成器 ( generator ) 和 yield。
- 6.优化循环。
- 7.优化包含多个判断表达式的顺序。
- 8.使用 join 合并迭代器中的字符串。
- 9.选择合适的格式化字符方式。
- 10 不借助中间变量交换两个变量的值。
- 11.使用 if is。
- 12.使用级联比较  $x < y < z$ 。
- 13.while 1 比 while True 更快。
- 14.使用\*\*而不是 pow。
- 15.使用 cProfile, cStringIO 和 cPickle 等用 c 实现相同功能 ( 分别对应 profile, StringIO, pickle ) 的包。
- 16.使用最佳的反序列化方式。
- 17.使用 C 扩展(Extension)。
- 18.并行编程。
- 19.终级大杀器：PyPy。



20.使用性能分析工具。

以上为简单思想，详情扫描二维码了解。



## 17. Django 项目的优化 ( web 通用 ) ( 2018-5-2-xhq )

### 1. 优化数据库查询

1.1 一次提供所有数据

1.2 仅提供相关的数据

### 2. 代码优化

2.1 简化代码

2.2 更新或替代第三方软件包

2.3 重构代码

更多详情参考博客：<http://python.jobbole.com/88971/>



### 三、智慧星光面试题

这家公司主要做舆情分析，问题主要集中在页面解析和 Selenium+Phantom JS 解析复杂页面，ajax 页面请求等，图片识别，机器学习。

1. 定义 A=( "a" , "b" , "c" , "d" ),执行 delA[2]后的结果为：

D(2018-5-1-lxy)

A: ( "a" , "c" , "d" )

B: ( "a" , "b" , "c" )

C: ( "a" , "b" , "d" )

D: 异常

2. String = "{1},{0}" ; string = string.format( "Hello" , "Python" ),请问将string 打印出来为 ( C ) (2018-5-1-lxy)

A: Hello Python

B: {1},{0}

C:Python,Hello

D:Hello,Hello

3. 定义 A=[1,2,3,4],使用列表生成式[i\*i for i in A]生成列表为 : B(2018-5-1-lxy)

A: [1, 2, 3, 4]

B: [1, 4, 9, 16]

C: [1, 4, 6, 8]

D: [1, 4, 9, 12]

4. 请对 Python 数据结构 Tuple,List,Dict 进行操作(2018-5-1-lxy)

1. 如何让元祖内部可变 ( 叙述或简单定义 ) ?

元祖变成列表，比如：

```
1. A = (1,2,3,4)
```

```
2. A = list(A)
```

2.如何将 L1 = [1,2,3,4],L2 = [6,7,8,9];使用列表内置函数变成 L1=[1,2,3,4,5,6,7,8,9]?

```
L1.extend(L2)。
```

3.如何将字典 D={ 'Adam' : 95, 'Lisa' : 85, 'Bart' : 59}中的值 'Adam' 删除?

```
del D['Adam']。
```

4.请按照如下格式 K : V 打印出字典?

```
1. for k,v in D.items():  
2.     print(k,":",v)
```

5. 请用 Python 内置函数处理以下问题 ? (2018-5-1-lxy)

1.请判断一个字符串是否以 er 结尾 ?

使用 endswith 函数 , 比如 :

```
1. Str1 = "nihaoer"  
2. print(Str1.endswith("er"))
```

2.请将 "#teacher#" 两侧的#去掉

```
1. str = "#tea#"  
2. b = str.replace("#","").strip()
```

3.请使用 map 函数将[1,2,3,4]处理成[1,0,1,0]

```
1. def f(x):  
2.     if x%2 == 0:  
3.         return 0  
4.     else:
```

```
5.         return 1
6. b = map(f,[1,2,3,4])
7. print(list(b))
```

4.请使用 filter 函数将[1,2,3,4]处理成[2,4]?

```
1. def f(x):
2.     if x%2 == 0:
3.         return x
4.
5. b = filter(f,[1,2,3,4])
6. print(list(b))
```

6. 请使用 reduce 函数计算 100 的阶乘?

reduce()函数在库 functools 里，如果要使用它，要从这个库里导入。reduce 函数与 map 函数有不一样地方，map 操作是并行操作，reduce 函数是把多个参数合并的操作，也就是从多个条件简化的结果，在计算机的算法里，大多数情况下，就是为了简单化。比如识别图像是否是一只猫，那么就是从众多的像素里提炼出来一个判断：是或否。可能是几百万个像素，就只出来一个结果。在 google 大规模集群里，就是利用这个思想，把前面并行处理的操作叫做 map，并行处理之后的结果，就需要简化，归类，把这个简化和归类的过程就叫做 reduce。由于 reduce 只能在一台主机上操作，并不能分布式地处理，但是 reduce 处理的是 map 结果，那么意味着这些结果已经非常简单，数据量大大减小，处理起来就非常快。因此可以把 mapreduce 过程叫做分析归纳的过程。

```
1. from functools import reduce
2.
3. sum=reduce(lambda x,y:x*y,range(1,101))
4. print(sum)
```

7. 现在需要从一个简单的登陆网站获取信息，请使用 Python 写出简要的登陆函数的具体实现？（登录信息只包含用户名，密码）(2018-5-1-lxy)

```
1. session = requests.session()
2. response = session.get(url,headers)
```

8. 正则表达式操作(2018-5-1-lxy)

1. 匹配手机号

分析：

（1）手机号位数为 11 位；

（2）开头为 1，第二位为 3 或 4 或 5 或 7 或 8；

表达式为：`/^[1][3,4,5,7,8][0-9]{9}$/`。

2. 请匹配出变量 `A = 'json({"Adam":95,"Lisa":85,"Bart":59})'` 中的 json 字符串。

```
1. A = 'json({"Adam":95,"Lisa":85,"Bart":59})'
2. b = re.search(r'json.*?({.*?}).*',A,re.S)
3. print(b.group(1))
```

3.怎么过滤评论中的表情？

```
1. co = re.compile(u'[\uD800-\uDBFF][\uDC00-\uDFFF]')
2. co.sub( " ",text)
```

## 四、壹讯面试题

1. Python 中 pass 语句的作用是什么？(2018-5-1-lxy)

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。

## 2. 尽可能写出多的 str 方法 ? (2018-5-1-lxy)

方法	描述
<u><code>string.capitalize()</code></u>	把字符串的第一个字符大写
<u><code>string.center(width)</code></u>	返回一个原字符串居中,并使用空格填充至长度 <code>width</code> 的新字符串
<u><code>string.count(str, beg=0, end=len(string))</code></u>	返回 <code>str</code> 在 <code>string</code> 里面出现的次数,如果 <code>beg</code> 或者 <code>end</code> 指定则返回指定范围内 <code>str</code> 出现的次数
<u><code>string.decode(encoding='UTF-8', errors='strict')</code></u>	以 <code>encoding</code> 指定的编码格式解码 <code>string</code> ,如果出错默认报一个 <code>ValueError</code> 的异常,除非 <code>errors</code> 指定的是 <code>'ignore'</code> 或者 <code>'replace'</code>
<u><code>string.encode(encoding='UTF-8', errors='strict')</code></u>	以 <code>encoding</code> 指定的编码格式编码 <code>string</code> ,如果出错默认报一个 <code>ValueError</code> 的异常,除非 <code>errors</code> 指定的是 <code>'ignore'</code> 或者 <code>'replace'</code>
<u><code>string.endswith(obj, beg=0, end=len(string))</code></u>	检查字符串是否以 <code>obj</code> 结束,如果 <code>beg</code> 或者 <code>end</code> 指定则检查指定的范围内是否以 <code>obj</code> 结束,如果是,返回 <code>True</code> ,否则返回 <code>False</code> .
<u><code>string.expandtabs(tabsize=8)</code></u>	把字符串 <code>string</code> 中的 <code>tab</code> 符号转为空格, <code>tab</code> 符号默认的空格数是 8。

<u>string.find(str, beg=0, end=len(string))</u>	检测 str 是否包含在 string 中，如果 beg 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1
<u>string.format()</u>	格式化字符串
<u>string.index(str, beg=0, end=len(string))</u>	跟 find()方法一样，只不过如果 str 不在 string 中会报一个异常.
<u>string.isalnum()</u>	如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True,否则返回 False
<u>string.isalpha()</u>	如果 string 至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False
<u>string.isdecimal()</u>	如果 string 只包含十进制数字则返回 True 否则返回 False.
<u>string.isdigit()</u>	如果 string 只包含数字则返回 True 否则返回 False.
<u>string.islower()</u>	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
<u>string.isnumeric()</u>	如果 string 中只包含数字字符，则返回 True，否则返回 False
<u>string.isspace()</u>	如果 string 中只包含空格，则返回 True，否则返回 False.



<u>string.istitle()</u>	如果 string 是标题化的(见 title())则返回 True , 否则返回 False
<u>string.isupper()</u>	如果 string 中包含至少一个区分大小写的字符 ,并且所有这些(区分大小写的)字符都是大写 , 则返回 True , 否则返回 False
<u>string.join(seq)</u>	以 string 作为分隔符 , 将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
<u>string.ljust(width)</u>	返回一个原字符串左对齐,并使用空格填充至长度 width 的新字符串
<u>string.lower()</u>	转换 string 中所有大写字符为小写.
<u>string.lstrip()</u>	截掉 string 左边的空格
<u>string.maketrans(intab, outtab)</u>	maketrans() 方法用于创建字符映射的转换表,对于接受两个参数的最简单的调用方式,第一个参数是字符串,表示需要转换的字符,第二个参数也是字符串表示转换的目标。
<u>max(str)</u>	返回字符串 str 中最大的字母。
<u>min(str)</u>	返回字符串 str 中最小的字母。
<u>string.partition(str)</u>	有点像 find()和 split()的结合体,从 str 出现的第一个位置起,把字符串 string 分成一个 3 元素的元组 (string_pre_str,str,string_post_str),如果 string 中不包含 str

	则 <code>string_pre_str == string</code> .
<code>string.replace(str1, str2, num=string.count(str1))</code>	把 <code>string</code> 中的 <code>str1</code> 替换成 <code>str2</code> ,如果 <code>num</code> 指定,则替换不超过 <code>num</code> 次.
<code>string.rfind(str, beg=0,end=len(string))</code>	类似于 <code>find()</code> 函数,不过是从右边开始查找.
<code>string.rindex( str, beg=0,end=len(string))</code>	类似于 <code>index()</code> ,不过是从右边开始.
<code>string.rjust(width)</code>	返回一个原字符串右对齐,并使用空格填充至长度 <code>width</code> 的新字符串
<code>string.rpartition(str)</code>	类似于 <code>partition()</code> 函数,不过是从右边开始查找.
<code>string.rstrip()</code>	删除 <code>string</code> 字符串末尾的空格.
<code>string.split(str="", num=string.count(str))</code>	以 <code>str</code> 为分隔符切片 <code>string</code> ,如果 <code>num</code> 有指定值,则仅分隔 <code>num</code> 个子字符串
<code>string.splitlines([keepends])</code>	按照行('\r', '\r\n', '\n')分隔,返回一个包含各行作为元素的列表,如果参数 <code>keepends</code> 为 <code>False</code> ,不包含换行符,如果为 <code>True</code> ,则保留换行符。
<code>string.startswith(obj, beg=0,end=len(string))</code>	检查字符串是否是以 <code>obj</code> 开头,是则返回 <code>True</code> ,否则返回 <code>False</code> 。如果 <code>beg</code> 和 <code>end</code> 指定值,则在指定范围内检查.

<u>string.strip([obj])</u>	在 string 上执行 lstrip()和 rstrip()
<u>string.swapcase()</u>	翻转 string 中的大小写
<u>string.title()</u>	返回"标题化"的 string,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())
<u>string.translate(str, del="")</u>	根据 str 给出的表(包含 256 个字符)转换 string 的字符，要过滤掉的字符放到 del 参数中
<u>string.upper()</u>	转换 string 中的小写字母为大写
<u>string.zfill(width)</u>	返回长度为 width 的字符串，原字符串 string 右对齐，前面填充 0
<u>string.isdecimal()</u>	isdecimal()方法检查字符串是否只包含十进制字符。这种方法只存在于 unicode 对象。

### 3. 生成一个斐波那契数列？(2018-5-1-lxy)

```
1. # [] 列表实现
2. def fibonacci(num):
3.     fibs = [0, 1]
4.     for i in range(num - 2):
5.         fibs.append(fibs[-2] + fibs[-1]) # 倒数第二个+倒数第一个数的结果，追加到列表
6.     print(fibs)
7. # yield 实现
8. def fab_demo4(max):
9.     a,n,b = 0,0,1
10.    while n < max:
11.        yield b
12.        #print b
13.        a,b = b,a+b
```

```
14.         n+=1
15. print(next(fab_demo4(5)))
16. for i in fab_demo4(8):
17.     print(i)
```

#### 4. 说明一下 os.path 和 sys.path 分别代表什么？(2018-5-1-lxy)

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。

#### 5. 什么是 lambda 函数？有什么好处？(2018-5-1-lxy)

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数。

1、lambda 函数比较轻便，即用即仍，很适合需要完成一项功能，但是此功能只在此一处使用，连名字都很随意的情况下；

2、匿名函数，一般用来给 filter，map 这样的函数式编程服务；

3、作为回调函数，传递给某些应用，比如消息处理。

### 五、H3C 面试题

#### 1. 下列哪个语句在 Python 中是非法的?(B)(2018-5-1-lxy)

A、x=y=z=1

B、x=(y=z+1)

C、x,y=y,x

D、x +=y

2. 关于 Python 内存管理,下列说法错误的是(B)(2018-5-1-lxy)

- A、变量不必事先声明 B、变量无须先创建和赋值而直接使用
- C、变量无须指定类型 D、可以使用 del 释放资源

3. 下面哪个不是 Python 合法的标识符(b)(2018-5-1-lxy)

- A、int32
- B、40XL
- C、self
- D、name

# 第一个字符必须是字母或是下划线

4. 下列哪种说法是错误的 ( A ) (2018-5-1-lxy)

- A、除字典类型外,所有标准对象均可以用于布尔测试
- B、空字符串的布尔值是 False
- C、空列表对象的布尔值是 Fale
- D、值为 0 的任何数字对象的布尔值是 False

5. 下列表达式的值为 True 的是 ( C ) (2018-5-1-lxy)

- A、 $5+4j>2-3j$
- B、 $3>2>2$
- C、 $(3,2)<('a', 'b')$
- D、 $'abc' > 'xyz'$

备注：在 Python3 中 整数和字符不可以使用运算符做比较，在 Python2 中可以。

6. Python 不支持的数据类型有(A)(2018-5-1-lxy)

A、 char B、 int C、 float D、 list

7. 关于 Python 中的复数,下列说法错误的是(C)(2018-5-1-lxy)

A、表示复数的语法是 real+ image j      B、实部和虚部都是浮点数  
C、虚部必须后缀 j,且必须是小写      D、方法 conjugate 返回复数的共轭

## 六、通联数据 (2018-5-2-lyf)

1. 说一下你对多线程的看法？

答案详见第三章→七.系统编程。

2. 多线程和多线程有什么区别？

答案详见第三章→七.系统编程。

3. 进程间的数据共享和线程间数据共享？

进程间数据共享：

多进程中，每个进程都是独立的，各自持有一份数据，无法共享。本篇文章介绍三种用于进程

数据共享的方法

Queue：

```
1. from multiprocessing import queues
2. import multiprocessing
3.
```

```
4. def func(i, q):
5.     q.put(i)
6.     print("--->", i, q.qsize())
7.
8.
9. q = queues.Queue(9, ctx=multiprocessing)
10. for i in range(5):
11.     p = multiprocessing.Process(target=func, args=(i, q,))
12.     p.start()
13. p.join()
```

Queue 是多进程安全的队列，可以使用 Queue 实现多进程之间的数据传递。put 方法用以插入数据到队列中，put 方法还有两个可选参数：blocked 和 timeout。如果 blocked 为 True（默认值），并且 timeout 为正值，该方法会阻塞 timeout 指定的时间，直到该队列有剩余的空间。如果超时，会抛出 Queue.Full 异常。如果 blocked 为 False，但 Queue 已满，会立即抛出 Queue.Full 异常。

get 方法可以从队列读取并且删除一个元素。同样，get 方法有两个可选参数：blocked 和 timeout。如果 blocked 为 True（默认值），并且 timeout 为正值，那么在等待时间内没有取到任何元素，会抛出 Queue.Empty 异常。如果 blocked 为 False，有两种情况存在，如果 Queue 有一个值可用，则立即返回该值，否则，如果队列为空，则立即抛出 Queue.Empty 异常。

```
1. import multiprocessing
2.
3. def func(i, q):
4.     q.put(i)
5.     print("--->", i, q.qsize())
6.
7.
8. q = multiprocessing.Queue()
9. for i in range(5):
10.     p = multiprocessing.Process(target=func, args=(i, q,))
11.     p.start()
12. p.join()
```

#### 4. Redis 数据库结构有那些？

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

#### 5. 你一般用 Redis 来做什么？

Redis 用到的地方很多，如我们最熟悉的分布式爬虫，Set 去重等，具体可以扫下面二维码查看。



#### 6. MongoDB 中存入了 100 万条数据，如何提高查询速度？

索引在很多数据库中是提高性能的标志优化手段，所以在大数据量的情况下索引可以提高数据的查询速度，如果没有索引 MongoDB 会扫描全部数据，才能获取满足条件的内容，在关系数据库中可以使用强制索引方式查询数据库，确保更准确快速的查询到满足条件的数据。

语法：

1、ensureIndex() 基本语法 1 创建升序索引 -1 创建降序索引

2、mongodb 默认所以字段 \_id,创建文档，会自动创建，此索引不能删除由 mongodb 自己维护

相关参数：



1、unique 创建唯一索引，默认 false ，true 必须唯一索引，否则报错

实例：

1、创建升序索引

```
db.user.ensureIndex({age:1});
```

```
db.user.find({age:{$gte:20}});
```

## 7. 如何提高并发性能？

我们常规处理并发的解决方案：

1.动态页面静态化。

2.制作数据库散列表，即分库分表。

3.增加缓存。

4.增加镜像。

5.部署集群。

6.负载均衡。

7.异步读取，异步编程。

8.创建线程池和自定义连接池，将数据持久化。

9.把一件事，拆成若干件小事，启用线程，为每个线程分配一定的事做，多个线程同时进行把该事件搞定再合并。

详细了解可以扫下面二维码：



8. 说一下你对 Django 框架的认识？

详细请看第五章第二节 Django。

9. Flask 框架了解吗，说下 Flask 和 Django 的区别？

Flask 了解详细请看第五章第一节 Flask。

区别：

( 1 ) Flask

- Flask 确实很“轻”，不愧是 Micro Framework，从 Django 转向 Flask 的开发者一定会如此感慨，除非二者均为深入使用过；
- Flask 自由、灵活，可扩展性强，第三方库的选择面广，开发时可以结合自己最喜欢用的轮子，也能结合最流行最强大的 Python 库；
- 入门简单，即便没有多少 web 开发经验，也能很快做出网站；

- 非常适用于小型网站；
- 非常适用于开发 web 服务的 API；
- 开发大型网站无压力，但代码架构需要自己设计，开发成本取决于开发者的能力和经验；
- 各方面性能均等于或优于 Django；
- Django 自带的或第三方的好评如潮的功能，Flask 上总会找到与之类似第三方库；
- Flask 灵活开发，Python 高手基本都会喜欢 Flask，但对 Django 却可能褒贬不一；
- Flask 与关系型数据库的配合使用不弱于 Django，而其与 NoSQL 数据库的配合远远优于 Django；
- Flask 比 Django 更加 Pythonic，与 Python 的 philosophy 更加吻合。

## ( 2 ) Django

- Django 太重了，除了 web 框架，自带 ORM 和模板引擎，灵活和自由度不够高；
- Django 能开发小应用，但总会有“杀鸡焉用牛刀”的感觉；
- Django 的自带 ORM 非常优秀，综合评价略高于 SQLAlchemy；
- Django 自带的模板引擎简单好用，但其强大程度和综合评价略低于 Jinja；
- Django 自带 ORM 也使 Django 与关系型数据库耦合度过高，如果想使用 MongoDB 等 NoSQL 数据，需要选取合适的第三方库，且总感觉 Django+SQL 才是天生一对的搭配，Django+NoSQL 砍掉了 Django 的半壁江山；
- Django 目前支持 Jinja 等非官方模板引擎；
- Django 自带的数据库管理 app 好评如潮；

- Django 非常适合企业级网站的开发：快速、靠谱、稳定；
- Django 成熟、稳定、完善，但相比于 Flask，Django 的整体生态相对封闭；
- Django 是 Python web 框架的先驱，用户多，第三方库最丰富，最好的 Python 库，如果不能直接用到 Django 中，也一定能找到与之对应的移植；
- Django 上手也比较容易，开发文档详细、完善，相关资料丰富。

## 10. 有一个无序数组，如何获取第 K 大的数，说下思路，实现后的时间复杂度？

根据自己熟悉的算法说下思路，如：快排、二分法。

具体实现方法和思路请扫下面的二维码：

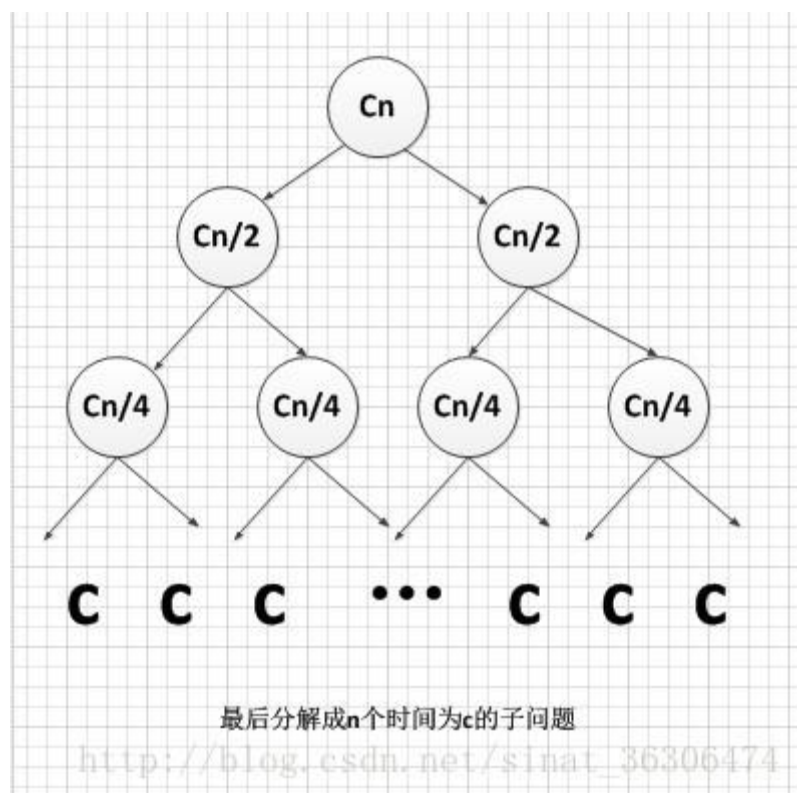


## 11. 归并排序的时间复杂度？

合并排序是比较复杂的排序，特别是对于不了解分治法基本思想的同学来说可能难以理解。总时间 = 分解时间 + 解决问题时间 + 合并时间。分解时间就是把一个待排序序列分解成两序列，时间为一常数，

时间复杂度  $O(1)$ 。解决问题时间是两个递归式，把一个规模为  $n$  的问题分成两个规模分别为  $n/2$  的子问题，时间为  $2T(n/2)$ 。合并时间复杂度为  $O(n)$ 。总时间  $T(n) = 2T(n/2) + O(n)$ 。这个递归式可以用递归树来解，其解是  $O(n \log n)$ 。此外在最坏、最佳、平均情况下归并排序时间复杂度均为  $O(n \log n)$ 。从合并过程中可以看出合并排序稳定。

用递归树的方法解递归式  $T(n) = 2T(n/2) + O(n)$ ：假设解决最后的子问题用时为常数  $c$ ，则对于  $n$  个待排序记录来说整个问题的规模为  $cn$ 。



从这个递归树可以看出，第一层时间代价为  $cn$ ，第二层时间代价为  $cn/2 + cn/2 = cn$ .....每一层代价都是  $cn$ ，总共有  $\log n + 1$  层。所以总的时间代价为  $cn * (\log n + 1)$ 。时间复杂度是  $O(n \log n)$ 。

## 七、北京号外科技爬虫面试题（2018-5-2 lyf）

### 1. 单引号、双引号、三引号的区别？

这几个符号都是可以表示字符串的，如果是表示一行，则用单引号或者双引号表示，它们的区别是如果内容里有"符号，并且你用双引号表示的话则需要转义字符，而单引号则不需要。

三单引号和三双引号也是表示字符串，并且可以表示多行，遵循的是所见即所得的原则。

另外，三双引号和三单引号可以作为多行注释来用，单行注释用#号。

### 2. 如何在一个 function 里面设置一个全局变量？

Global 声明。

### 3. 描述 yield 使用场景？

生成器。

当有多个返回值时，用 return 全部一起返回了，需要单个逐一返回时可以用 yield。

详细了解可以扫下面的二维码：



## 4. 生成 1~10 之间的整数？

```
for i in range(1,11)
```

生成器：(i for i in range(1,10))

## 5. Python 如何生成缩略图？

```
1. import os
2. import glob
3. from PIL import Image
4.
5. def thumbnail_pic(path):
6.     a=glob.glob(r'./*.jpg')
7.     for x in a:
8.         name=os.path.join(path,x)
9.         im=Image.open(name)
10.        im.thumbnail((80,80))
11.        print(im.format,im.size,im.mode)
12.        im.save(name,'JPEG')
13.    print('Done!')
14.
15. if __name__=='__main__':
16.    path='.'
17.    thumbnail_pic(path )
```

详细请扫下面的二维码：



## 6. 列出比较熟悉的爬虫框架，并简要说明？

- (1) Scrapy 框架：很强大的爬虫框架，可以满足简单的页面爬取（比如可以明确获知 url pattern 的情况）。用这个框架可以轻松爬下来如亚马逊商品信息之类的数据。但是对于稍微复杂一点的页面，如 weibo 的页面信息，这个框架就满足不了需求了。
- (2) Crawley: 高速爬取对应网站的内容，支持关系和非关系数据库，数据可以导出为 JSON、XML 等
- (3) Portia: 可视化爬取网页内容
- (4) newspaper: 提取新闻、文章以及内容分析
- (5) python-goose: java 写的文章提取工具
- (6) BeautifulSoup: 名气大，整合了一些常用爬虫需求。缺点：不能加载 JS。
- (7) mechanize: 优点：可以加载 JS。缺点：文档严重缺失。不过通过官方的 example 以及人肉尝试的方法，还是勉强能用的。
- (8) selenium: 这是一个调用浏览器的 driver，通过这个库你可以直接调用浏览器完成某些操作，比如输入验证码。
- (9) cola: 一个分布式爬虫框架。项目整体设计有点糟，模块间耦合度较高。

## 7. 列举常见的反爬技术，并给出应对方案？

### 1. Headers：

从用户的 headers 进行反爬是最常见的反爬虫策略。Headers（上一讲中已经提及）是一种区分浏览器行为和机器行为中最简单的方法，还有一些网站会对 Referer（上级链接）进行检测（机器行为不太可能通过链接跳转实现）从而实现爬虫。



相应的解决措施：通过审查元素或者开发者工具获取相应的 headers 然后把相应的 headers 传输给 python 的 requests，这样就能很好地绕过。

## 2. IP 限制

一些网站会根据你的 IP 地址访问的频率，次数进行反爬。也就是说如果你用单一的 IP 地址访问频率过高，那么服务器会在短时间内禁止这个 IP 访问。

解决措施：构造自己的 IP 代理池，然后每次访问时随机选择代理（但一些 IP 地址不是非常稳定，需要经常检查更新）。

## 3. UA 限制

UA 是用户访问网站时候的浏览器标识，其反爬机制与 ip 限制类似。

解决措施：构造自己的 UA 池，每次 python 做 requests 访问时随机挂上 UA 标识，更好地模拟浏览器行为。当然如果反爬对时间还有限制的话，可以在 requests 设置 timeout（最好是随机休眠，这样会更安全稳定，time.sleep()）。

## 4. 验证码反爬虫或者模拟登陆

验证码：这个办法也是相当古老并且相当的有效果，如果一个爬虫要解释一个验证码中的内容，这在以前通过简单的图像识别是可以完成的，但是就现在来讲，验证码的干扰线，噪点都很多，甚至还出现了人类都难以认识的验证码（~~~~~~~）。

相应的解决措施：验证码识别的基本方法：截图，二值化、中值滤波去噪、分割、紧缩重排（让高矮统一）、字库特征匹配识别。（python 的 PIL 库或者其他）

模拟登陆（例如知乎等）：用好 python requests 中的 session（下面几行代码实现了最简单的 163 邮箱的登陆，其实原理是类似的~~）。

```
1. import requests
2. s = requests.session()
3. login_data={"account": " ", "password": " "}
```

```
4. res=s.post("http://mail.163.com/",login_data)
```

## 5.Ajax 动态加载

网页的不希望被爬虫拿到的数据使用 Ajax 动态加载，这样就为爬虫造成了绝大的麻烦，如果一个爬虫不具备 js 引擎，或者具备 js 引擎，但是没有处理 js 返回的方案，或者是具备了 js 引擎，但是没办法让站点显示启用脚本设置。基于这些情况，ajax 动态加载反制爬虫还是相当有效的。

Ajax 动态加载的工作原理是：从网页的 url 加载网页的源代码之后，会在浏览器里执行 JavaScript 程序。这些程序会加载出更多的内容，并把这些内容传输到网页中。这就是为什么有些网页直接爬它的 URL 时却没有数据的原因。

处理方法：若使用审查元素分析“请求”对应的链接(方法：右键→审查元素→Network→清空，点击“加载更多”，出现对应的 GET 链接寻找 Type 为 text/html 的，点击查看 get 参数或者复制 Request URL)，循环过程。如果“请求”之前有页面，依据上一步的网址进行分析推导第 1 页。以此类推，抓取抓 Ajax 地址的数据。对返回的 json 使用 requests 中的 json 进行解析，使用 eval() 转成字典处理(上一讲中的 fiddler 可以格式化输出 json 数据。

## 6.cookie 限制

一次打开网页会生成一个随机 cookie，如果再次打开网页这个 cookie 不存在，那么再次设置，第三次打开仍然不存在，这就非常有可能是爬虫在工作了。

解决措施：在 headers 挂上相应的 cookie 或者根据其方法进行构造（例如从中选取几个字母进行构造）。如果过于复杂，可以考虑使用 selenium 模块（可以完全模拟浏览器行为）。

## 8. 网络协议 http 和 https 区别？

HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务端请求和应答的标准（TCP），用于从 WWW 服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

HTTPS：是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版，即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。

HTTPS 协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

详细讲解请扫下面二维码查看：



## 9. 什么是 cookie，session 有什么区别？

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，会比较占用你服务器的性能，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、可以考虑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。

## 10.Mysql 中 myisam 与 innodb 的区别？

### 1、 存储结构

MyISAM：每个 MyISAM 在磁盘上存储成三个文件。第一个文件的名称以表的名称开始，扩展名指出文件类型。.frm 文件存储表定义。数据文件的扩展名为.MYD (MYData)。索引文件的扩展名是.MYI (MYIndex)。

InnoDB：所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB 表的大小只受限于操作系统文件的大小，一般为 2GB。

### 2、 存储空间

MyISAM：可被压缩，存储空间较小。支持三种不同的存储格式：静态表(默认，但是注意数据末尾不能有空格，会被去掉)、动态表、压缩表。

InnoDB：需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引。

### 3、 事务支持

MyISAM：强调的是性能，每次查询具有原子性,其执行速度比 InnoDB 类型更快，但是不提供事务支持。

InnoDB：提供事务支持事务，外部键等高级数据库功能。具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。

### 4、 CURD 操作

MyISAM：如果执行大量的 SELECT，MyISAM 是更好的选择。(因为没有支持行级锁)，在增删的时候需要锁定整个表格，效率会低一些。相关的是 innodb 支持行级锁，删除插入的时候只需要锁定改行就行，效率较高

InnoDB：如果你的数据执行大量的 INSERT 或 UPDATE，出于性能方面的考虑，应该使用 InnoDB 表。DELETE 从性能上 InnoDB 更优，但 DELETE FROM table 时，InnoDB 不会重新建立表，而是一行一行的删除，在 innodb 上如果要清空保存有大量数据的表，最好使用 truncate table 这个命令。

## 5、 外键

MyISAM：不支持

InnoDB：支持

想详细了解可扫下面二维码：



## 八、首信 Python 研发面试题 ( 2018-5-2 lyf )

1. Python 中 list、tuple、dict、set 有什么区别，主要应用在什么样的场景？并用 for 语句遍历？

区别：

- 1、list、tuple 是有序列表；dict、set 是无序列表；
- 2、list 元素可变、tuple 元素不可变；
- 3、dict 和 set 的 key 值不可变，唯一性；
- 4、set 只有 key 没有 value；
- 5、set 的用途：去重、并集、交集等；
- 6、list、tuple：+、\*、索引、切片、检查成员等；
- 7、dict 查询效率高，但是消耗内存多；list、tuple 查询效率低、但是消耗内存少

应用场景：

list：简单的数据集合,可以使用索引;

tuple：把一些数据当做一个整体去使用,不能修改;

dict：使用键值和值进行关联的数据;

set：数据只出现一次,只关心数据是否出现, 不关心其位置;

列表遍历：

```
1. >>> a_list = [1, 2, 3, 4, 5]
2. >>> for num in a_list:
3. ...     print(num,end=' ')
4. ....
5. 1 2 3 4 5
```

元组遍历：

```
1. >>> a_turple = (1, 2, 3, 4, 5)
```

```
2. >>> for num in a_turple:
3. ...     print(num,end=" ")
4. 1 2 3 4 5
```

遍历字典：

```
1. dict = { "name" : " xiaoming" , " sex" : " man" }
2. >>> for key in dict.key():
3. >>> for value in dict.value()
4. >>> for item in dict.items()
5. ...     print key
6. ...     print value
7. ...     print item
```

Set 遍历：

```
1. >>> s = set(['Adam', 'Lisa', 'Bart'])
2. >>> for name in s:
3. ...     print name
4. ...
5. Lisa
6. Adam
7. Bart
```

## 2. Python 中静态函数、类函数、成员函数的区别？并写一个示例？

定义：

静态函数(@staticmethod)：即静态方法,主要处理与这个类的逻辑关联, 如验证数据；

类函数(@classmethod)：即类方法, 更关注于从类中调用方法, 而不是在实例中调用方法, 如构造重载；

成员函数：实例的方法, 只能通过实例进行调用。

详细请看扫下面二维码：



3. 用 Python 语言写一个函数，输入一个字符串，返回倒序结果？

```
1. def test ()
2.     strA = raw_input("请输入需要翻转的字符串：")
3.     order = []
4.     for i in strA:
5.         order.append(i)
6.     order.reverse()    #将列表反转
7.     print "".join(order)    #将 list 转换成字符串
8.     test()
```

4. 介绍一下 Python 的异常处理机制和自己开发过程中的体会？

1. 默认异常处理器

代码如下：

```
1. s = 'Hello girl!'
2. print s[100]
3. print 'continue'
```

如果我们没有对异常进行任何预防,那么在程序执行的过程中发生异常,就会中断程序,调用 python 默认的异常处理器,并在终端输出异常信息。这种情况下,第 3 行代码不会执行。

2. try...except



代码如下:

```
1. s = 'Hello girl!'
2. try:
3.     print s[100]
4. except IndexError:
5.     print 'error...'
6.     print 'continue'
```

程序执行到第 2 句时发现 try 语句,进入 try 语句块执行,发生异常,回到 try 语句层,寻找后面是否有 except 语句。找到 except 语句后,会调用这个自定义的异常处理器。except 将异常处理完毕后,程序继续往下执行。这种情况下,最后两个 print 语句都会执行。

except 后面也可以为空,表示捕获任何类型的异常。

### 3. try...finally

代码如下:

```
1. s = 'Hello girl!'
2. try:
3.     print s[100]
4. finally:
5.     print 'error...'
6.     print 'continue'
```

finally 语句表示,无论异常发生与否,finally 中的语句都要执行。但是,由于没有 except 处理器,finally 执行完毕后程序便中断。这种情况下,倒第 2 个 print 会执行,到第 1 个不会执行。如果 try 语句中没有异常,三个 print 都会执行。

### 4. assert

代码如下:

```
1. assert False, 'error...'
2. print 'continue'
```

这个语句,先判断 assert 后面紧跟的语句是 True 还是 False,如果是 True 则继续执行 print,如果是 False 则中断程序,调用默认的异常处理器,同时输出 assert 语句逗号后面的提示信息。本例情况下,程序中断,提示 error,后面的 print 不执行。

## 5. with...as

代码如下:

```
1. with open('nothing.txt','r') as f:  
2.     f.read()  
3.     print 2/0  
4.     print 'continue'
```

我们平时在使用类似文件的流对象时,使用完毕后要调用 close 方法关闭,很麻烦。这里 with...as 语句提供了一个非常方便的替代方法:open 打开文件后将返回的文件流对象赋值给 f,然后在 with 语句块中使用。with 语句块完毕之后,会隐藏地自动关闭文件。

如果 with 语句或语句块中发生异常,会调用默认的异常处理器处理,但文件还是会正常关闭。

这种情况下,会抛出异常,最后的 print 不执行。

## 5. jQuery 库中\$()是什么? 网上有 5 个<div>元素,如何使用 jQuery 来选择它们?

\$()函数是 JQuery 函数的别称,\$()函数用于将任何对象包裹成 jQuery 对象,接着就可以被允许调用定义在 jQuery 对象上的多个不同方法。甚至可以将一个选择器字符串传入 \$()函数,它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。可以用 each()方法进行遍历里面的对象。

选择<div>元素:这个问题是 jQuery 基于选择器的。jQuery 支持不同类型的选择器,有 ID 选择器、class 选择器、标签选择器。这个问题的答案是使用标签选择器来选择所有的 div 元素。

jQuery 代码:

```
$("div").
```

其返回值是一个包含 5 个 div 标签的 jQuery 对象。

## 6. 写一个 Bash Shell 脚本来得到当前的日期、时间、用户名和当前工作目录？

输出用户名，当前日期和时间，以及当前工作目录的命令就是 `logname`，`date`，`who i am` 和 `pwd`。

## 7. Django 中使用 memcached 作为缓存的具体方法？有缺点说明？

memcached 是一种缓存技术, 基于 c/s 模式, 他可以把你的数据放入内存, 从而通过内存访问提速, 因为内存最快的。

详细请扫下面二维码查看：



## 九、微影时代 ( 2018-5-2 lyf )

### 1. 装饰器的理解？

详见第三章 ( Python 高级 ) → 四. 设计模式 → 3. 装饰器。

## 2. scrapy 框架，反扒机制，手写中间件？

答案详见第十二章（企业真题实战）→四.北京号外科技面试题→7。

## 3. HTTP 头有什么字段？

每个 HTTP 请求和响应都会带有相应的头部信息。默认情况下，在发送 XHR 请求的同时，还会发送下列头部信息：

Accept:浏览器能够处理的内容类型

Accept-Charset:浏览器能够显示的字符集

Accept-Encoding：浏览器能够处理的压缩编码

Accept-Language：浏览器当前设置的语言

Connection：浏览器与服务器之间连接的类型

Cookie：当前页面设置的任何 Cookie

Host：发出请求的页面所在的域

Referer：发出请求的页面的 URL

User-Agent：浏览器的用户代理字符串

HTTP 响应头部信息:

Date：表示消息发送的时间，时间的描述格式由 rfc822 定义

server:服务器名字。

Connection：浏览器与服务器之间连接的类型

content-type:表示后面的文档属于什么 MIME 类型

Cache-Control：控制 HTTP 缓存

#### 4. POST 登录数据方式？

HTTP 协议是以 ASCII 码传输，建立在 TCP/IP 协议之上的应用层规范。规范把 HTTP 请求分为三个部分：状态行、请求头、消息主体。协议规定 POST 提交的数据必须放在消息主体 (entity-body) 中，但协议并没有规定数据必须使用什么编码方式。

具体请求方式请扫下面二维码：



#### 5. 加密页面如何解析？

答案详细请扫下面二维码：



## 6. 爬虫如何定时增量更新数据

答案详细请扫下面二维码：



## 7. scrapy 怎么设置时间间隔？

在 spiders 文件中写如下：

```
custom_settings = { 'DOWNLOAD_DELAY': 0.2, 'CONCURRENT_REQUESTS_PER_IP': 4,  
'DOWNLOADER_MIDDLEWARES': {}, }
```

## 8. 爬虫遇到验证码如何解决？

答案课参考第六章（爬虫）→一.常见库和模块→10.验证码的解决。

了解实际案例可以扫下面二维码：



## 十、斯沃创智

### 1. 简述 Python 中 is 和 == 的区别 （2018-5-2-zcz）

Python 中的对象包含三要素：id、type、value。

其中 id 用来唯一标识一个对象，type 标识对象的类型，value 是对象的值。is 判断的是 a 对象是否是 b 对象，是通过 id 来判断的。== 判断的是 a 对象的值是否和 b 对象的值相等，是通过 value 来判断的。

## 2. 简述 read, readline 和 readlines 的区别 ( 2018-5-2-zcz )

1. read() 每次读取整个文件，它通常将读取到底文件内容放到一个字符串变量中，也就是说 .read() 生成文件内容是一个字符串类型；
2. readline()每只读取文件的一行，通常也是读取到的一行内容放到一个字符串变量中，返回 str 类型；
3. readlines()每次按行读取整个文件内容，将读取到的内容放到一个列表中，返回 list 类型；

## 3. 举例说明创建字典的至少两种方法 ( 2018-5-2-zcz )

1. 用{}创建字典
2. 用内置函数 dict()

## 4. 简述 Python 里面 search 和 match 的区别 ( 2018-5-2-zcz )

match()函数只检测 RE 是不是在 string 的开始位置匹配，search()会扫描整个 string 查找匹配；也就是说 match()只有在 0 位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返回 none；

例如：

```
print(re.match('super', 'superstition').span()) 会返回(0, 5)
```

而 print(re.match('super', 'insuperable')) 则返回 None

search()会扫描整个字符串并返回第一个成功的匹配：

例如：

```
print(re.search('super', 'superstition').span())返回(0, 5)
```



`print(re.search('super', 'insuperable').span())`返回(2, 7)

其中 `span` 函数定义如下，返回位置信息：

`span([group]):`

返回(`start(group)`, `end(group)`)。

## 5. Python 代码实现:删除一个 list 里面的重复元素 ( 2018-5-2-zcz )

方法一：是利用 `map` 的 `fromkeys` 来自动过滤重复值，`map` 是基于 `hash` 的，大数组的时候应该会比排序快点。

方法二：是用 `set()`, `set` 是定义集合的,无序，非重复。

方法三：是排序后，倒着扫描，遇到已有的元素删之。

```
1.  #!/usr/bin/python
2.  #coding=utf-8
3.  ""
4.  Created on 2012-2-22
5.  Q: 给定一个列表，去掉其重复的元素，并输出
6.  ""
7.  def distFunc1():
8.      a=[1,2,4,2,4,5,6,5,7,8,9,0]
9.      b={}
10.     b=b.fromkeys(a)
11.     print b
12.     #print b.keys()
13.     a=list(b.keys())
14.     print a
15. def distFunc2():
16.     a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
17.     a=list(set(a)) # set 是非重复的，无序集合。可以用 list 来的排队对 set 进行排序，list()转换为列表，a.sort 来排序
18.     print a
19. def distFunc3():
20.     #可以先把 list 重新排序，然后从 list 的最后开始扫描，代码如下：
21.     List=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
22.     if List:
23.         List.sort()
24.         #print List
```

```
25.         last = List[-1]
26.         #print last
27.         for i in range(len(List)-2, -1, -1):
28.             if last==List[i]:
29.                 del List[i]
30.             else: last=List[i]
31.
32. if __name__ == '__main__':
33.     distFunc1()
34.     distFunc2()
35. distFunc3()
```

## 6. Python 代码中(\*args, \*\*kwargs)是什么意思 ( 2018-5-2-zcz )

\*args 表示任何多个无名参数，它是一个 tuple。

\*\*kwargs 表示关键字参数，它是一个 dict。

## 十一、天广汇通

### 1. 说明 os.path 和 sys.path 分别代表什么？ ( 2018-5-2-zcz )

sys.path 是喜闻乐见的 PATH 环境变量，os.path 是一个 module，提供 split、join、basename 等“处理目录、文件名”的工具。

### 2. 解释一下并行 ( parallel ) 和并发 ( concurrency ) 的区别 ( 2018-5-2-zcz )

并行 ( parallel ) 是指同一时刻，两个或两个以上时间同时发生。

并发 ( parallel ) 是指同一时间间隔 ( 同一段时间 )，两个或两个以上时间同时发生。

### 3. 在 Python 中可以实现并发的库有哪些？ ( 2018-5-2-zcz )

1 ) 线程    2 ) 进程    3 ) 协程    4 ) threading。

4. 如果一个程序需要进行大量的 IO 操作，应当使用并行还是并发？（2018-5-2-zcz）

并发。

5. 如果程序需要进行大量的逻辑运算操作，应当使用并行还是并发？（2018-5-2-zcz）

并行。

## 十二、信德数据

1. 网络七层协议是哪几层？HTTP 协议输入是第几层？（2018-5-2-zcz）

7 层从上到下分别是 7 应用层 6 表示层 5 会话层 4 传输层 3 网络层 2 数据链路层 1 物理层；其中高层（即 7、6、5、4 层）定义了应用程序的功能，下面 3 层（即 3、2、1 层）主要面向通过网络端到端的数据流。

HTTP 属于应用层。

2. 什么是 HTTP 协议？HTTP 请求有哪几种？（2018-5-2-zcz）

HTTP 是 hypertext transfer protocol（超文本传输协议）的简写，它是 TCP/IP 协议的一个应用层协议，用于定义 WEB 浏览器与 WEB 服务器之间交换数据的过程。客户端连上 web 服务器后，若想获得 web 服务器中的某个 web 资源，需遵守一定的通讯格式，HTTP 协议用于定义客户端与 web 服务器通讯的格式。

HTTP 请求有 8 种：

OPTIONS / HEAD / GET / POST / PUT / DELETE / TRACE / CONNECT。

### 3. 什么是 HTTP 代理？作用是什么？（2018-5-2-zcz）

代理服务器英文全称是 Proxy Server，其功能就是代理网络用户去取得网络信息。形象的说：它是网络信息的中转站。

代理服务器可以实现各种时髦且有用的功能。它们可以改善安全性，提高性能，节省费用。

### 4. 什么是反向代理？作用是什么？（2018-5-2-zcz）

代理可以假扮 Web 服务器。这些被称为替换物(surrogate)或反向代理(reverse proxy)的代理接收发送给 Web 服务器的真实请求，但与 Web 服务器不同的是，它们可以发起与其他服务器的通信，以便按需定位所请求的内容。

可以用这些反向代理来提高访问慢速 Web 服务器上公共内容的性能。在这种配置中，通常将这些反向代理称为服务器加速器(server accelerator)。还可以将替换物与内容路由功能配合使用，以创建按需复制内容的分布式网络。

### 5. HTTPS 和 HTTP 的区别（2018-5-2-zcz）

1) https 协议需要到 ca 申请证书，一般免费证书很少，需要交费。

2) http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

3) http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

4) http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

### 6. 什么是进程？什么事协程？（2018-5-2-zcz）

答案详见：Python 高级-系统编程-进程总结-01。

## 7. 什么事死锁？死锁产生的四个必要条件？

答案详见：Python 高级-系统编程-进程总结-04。

## 8. 什么是内存泄漏？（2018-5-2-zcz）

答案详见：Python 高级-内存管理与垃圾回收机制-02。

## 9. Python 代码中(\*args, \*\*kwargs)是什么意思？（2018-5-2-zcz）

答案详见：企业真题实战-斯沃创智-06。

## 十三、成安

### 1. Python 中使用%与.format 格式化文本（2018-5-2-zcz）



### 2. Python 的 logging 模块常用的几个等级？（2018-5-2-zcz）

日志级别：

critical > error > warning > info > debug,notset

级别越高打印的日志越少，反之亦然，即

Debug : 打印全部的日志(notset 等同于 debug)

info : 打印 info,warning,error,critical 级别的日志

warning : 打印 warning,error,critical 级别的日志

error : 打印 error,critical 级别的日志

critical : 打印 critical 级

### 3. 在 HTTP1.1 中常见的状态码有哪些，如何设置状态码？（2018-5-2-zcz）

1XX Informational 信息性状态码，表示接受的请求正在处理

2XX Success 成功状态码，表示请求正常处理完毕

3XX Redirection 重定向状态码，表示需要客户端需要进行附加操作

4XX Client Error 客户端错误状态码，表示服务器无法处理请求

5XX Server Error 服务器错误状态码，表示服务器处理请求出错

### 4. Python 如何处理上传文件？（2018-5-2-zcz）

Python 中使用 GET 方法实现上传文件，下面就是用 Get 上传文件的例子，client 用来发 Get 请求，server 用来收请求。

请求端代码：

```
1. import requests #需要安装 requests
2. with open('test.txt', 'rb') as f:
3.     requests.get('http://服务器 IP 地址:端口', data=f)
4. 服务端代码：
5. var http = require('http');
6. var fs = require('fs');
7. var server = http.createServer(function(req, res){
```

```
8.    //console.log(req);
9.    var recData = "";
10.   req.on('data', function(data){
11.       recData += data;
12.   })
13.   req.on('end', function(data){
14.       recData += data;
15.       fs.writeFile('recData.txt', recData, function(err){
16.           console.log('file received');
17.       })
18.   })
19.   res.end('hello');
20. })
21. server.listen(端口);
```

## 5. 用 Python 写一段排序算法。（2018-5-2-zcz）

参考博客：[https://blog.csdn.net/littlely\\_ll/article/details/78089234](https://blog.csdn.net/littlely_ll/article/details/78089234)



## 十四、博派通达

### 1. 请列举你使用过的 Python 代码检测工具(2018-5-2-xhq)

1. 移动应用自动化测试 Appium
2. OpenStack 集成测试 Tempest
3. 自动化测试框架 STAF
4. 自动化测试平台 TestMaker
5. JavaScript 内存泄露检测工具 Leak Finder
6. Python 的 Web 应用验收测试 Splinter
7. 即插即用设备调试工具 UPnP-Inspector

更多详情了解二维码：





## 2. 简述 Python 垃圾回收机制和如何解决循环引用(2018-5-2-xhq)

引用计数：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，当一个对象的引用被创建或者复制时，对象的引用计数加 1；当一个对象的引用被销毁时，对象的引用计数减 1；当对象的引用计数减少为 0 时，就意味着对象已经没有人使用了，可以将其所占用的内存释放了。虽然引用计数必须在每次分配和释放内存的时候加入管理引用计数的动作，然而与其他主流的垃圾收集技术相比，引用计数有一个最大的优点，即“实时性”，任何内存，一旦没有指向它的引用，就会立即被回收。而其他的垃圾收集计数必须在某种特殊条件下（比如内存分配失败）才能进行无效内存的回收。

引用计数机制执行效率问题：引用计数机制所带来的维护引用计数的额外操作与 Python 运行中所进行的内存分配和释放，引用赋值的次数是成正比的。而这点相比其他主流的垃圾回收机制，比如“标记-清除”，“停止-复制”，是一个弱点，因为这些技术所带来的额外操作基本上只是与待回收的内存数量有关。

如果说执行效率还仅仅是引用计数机制的一个软肋的话，那么很不幸，引用计数机制还存在着一个致命的弱点，正是由于这个弱点，使得狭义的垃圾收集从来没有将引用计数包含在内，能引发出这个致命的弱点就是循环引用（也称交叉引用）。

问题说明：

循环引用可以使一组对象的引用计数不为 0，然而这些对象实际上并没有被任何外部对象所引用，它们之间只是相互引用。这意味着不会再有人使用这组对象，应该回收这组对象所占用的内存空间，然后由于相互引用的存在，每一个对象的引用计数都不为 0，因此这些对象所占用的内存永远不会被释放。比如：这一点是致命的，这与手动进行内存管理所产生的内存泄露毫无区别。

要解决这个问题，Python 引入了其他的垃圾收集机制来弥补引用计数的缺陷：“标记-清除”，“分代回收”两种收集技术。

标记-清除：“标记-清除”是为了解决循环引用的问题。可以包含其他对象引用的容器对象( 比如 :list , set , dict , class , instance ) 都可能产生循环引用。

我们必须承认一个事实，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。我们必须先将循环引用摘掉，那么这两个对象的有效计数就现身了。假设两个对象为 A、B，我们从 A 出发，因为它有一个对 B 的引用，则将 B 的引用计数减 1；然后顺着引用达到 B，因为 B 有一个对 A 的引用，同样将 A 的引用减 1，这样，就完成了循环引用对象间环摘除。

但是这样就有一个问题，假设对象 A 有一个对象引用 C，而 C 没有引用 A，如果将 C 计数引用减 1，而最后 A 并没有被回收，显然，我们错误的将 C 的引用计数减 1，这将导致在未来的某个时刻出现一个对 C 的悬空引用。这就要求我们必须在 A 没有被删除的情况下复原 C 的引用计数，如果采用这样的方案，那么维护引用计数的复杂度将成倍增加。

原理：“标记-清除”采用了更好的做法，我们并不改动真实的引用计数，而是将集合中对象的引用计数复制一份副本，改动该对象引用的副本。对于副本做任何的改动，都不会影响到对象生命走起的维护。

这个计数副本的唯一作用是寻找 root object 集合（该集合中的对象是不能被回收的）。当成功寻找到 root object 集合之后，首先将现在的内存链表一分为二，一条链表中维护 root object 集合，成为 root 链表，而另外一条链表中维护剩下的对象，成为 unreachable 链表。之所以要剖成两个链表，是基于这样的一种考虑：现在的 unreachable 可能存在被 root 链表中的对象，直接或间接引用的对象，这些对象是不能被回收的，一旦在标记的过程中，发现这样的对象，就将其从 unreachable 链表中移到 root 链表中；当完成标记后，unreachable 链表中剩下的所有对象就是名副其实的垃圾对象了，接下来的垃圾回收只需限制在 unreachable 链表中即可。

**分代回收 背景：**分代的垃圾收集技术是在上个世纪 80 年代初发展起来的一种垃圾收集机制，一系列的研究表明：无论使用何种语言开发，无论开发的是何种类型，何种规模的程序，都存在这样一点相同之处。即：一定比例的内存块的生存周期都比较短，通常是几百万条机器指令的时间，而剩下的内存块，起生存周期比较长，甚至会从程序开始一直持续到程序结束。

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。为了提高垃圾收集的效率，采用“空间换时间的策略”。

**原理：**将系统中的所有内存块根据其存活时间划分为不同的集合，每一个集合就成为一个“代”，垃圾收集的频率随着“代”的存活时间的增大而减小。也就是说，活得越长的对象，就越不可能是垃圾，就应该减少对它的垃圾收集频率。那么如何来衡量这个存活时间：通常是利用几次垃圾收集动作来衡量，如果一个对象经过的垃圾收集次数越多，可以得出：该对象存活时间就越长。

### 3. 请简述如何编写清晰可读的代码（2018-5-2-xhq）

一、写 pythonic 代码

二、理解 Python 和 C 语言的不同之处

三、在代码中适当添加注释

Python 中有三种形式的代码注释：块注释、行注释以及文档注释。

使用块注释或者行注释的时候仅仅注释那些复杂的操作、算法，或者难以理解，不能一目了然的代码给外部可访问的函数和方法（无论简单与否）添加文档注释。注释要清楚的描述方法的功能，并对参数、返回值以及可能发生的异常进行说明，使得外部调用它的人员仅仅看文档注释就能正确使用。较为复杂的内部方法也需要进行注释。

四、通过适当添加空行使代码布局更为优雅、合理。

#### 五、编写函数的 4 个原则

- 1) 函数设计要尽量短小
- 2) 函数声明要做到合理、简单、易于使用
- 3) 函数参数设计应该考虑向下兼容
- 4) 一个函数只做一件事情，尽量保证函数语句粒度的一致性

#### 六、将常量集中到一个文件

在 Python 中如何使用常量呢，一般来说有以下两种方式：

1) 通过命名风格来提醒使用者该变量代表的意义为常量。如 TOTAL，MAX\_OVERFLOW，然而这种方式并没有实现真正的常量，其对应的值仍然可以改变，这只是一种约定俗成的风格。

2) 通过自定义的类实现常量功能，这要求符合“命名全部为大写”和“值一旦绑定便不可再修改”这两个条件。

参考博客：<https://www.cnblogs.com/cotyb/p/5086771.html>



#### 4. 请列举出常见的 Mysql 存储引擎 ( 2018-5-2-xhq )

详见数据库 Mysql 是哪个引擎，各引擎之间有什么区别。

#### 5. InnoDB 有哪些特性 ( 2018-5-2-xhq )

详见数据库 Mysql 是哪个引擎，各引擎之间有什么区别。

#### 6. 请列出 MySQL 数据库查询的技巧 ( 2018-5-2-xhq )

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null 可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：select id from t where num=0。

3.应尽量避免在 where 子句中使用!=或<>操作符，否则引擎将放弃使用索引而进行全表扫描。

4.应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num=10 or num=20 可以这样查询：select id from t where num=10 union all select id from t where num=20。

5.in 和 not in 也要慎用，否则会导致全表扫描，如：select id from t where num in(1,2,3) 对于连续的数值，能用 between 就不要用 in 了：select id from t where num between 1 and 3。

6.下面的查询也将导致全表扫描：select id from t where name like '%李%' 若要提高效率，可以考虑全文检索。

7. 如果在 where 子句中使用参数，也会导致全表扫描。因为 SQL 只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译

时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

`select id from t where num=@num` 可以改为强制查询使用索引：`select id from t with(index(索引名)) where num=@num`。

8.应尽量避免在 `where` 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：`select id from t where num/2=100` 应改为：`select id from t where num=100*2`。

9.应尽量避免在 `where` 子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：`select id from t where substring(name,1,3)=' abc'`，`name` 以 `abc` 开头的 `id` 应改为：`select id from t where name like 'abc%'`。

10.不要在 `where` 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12.不要写一些没有意义的查询，如需要生成一个空表结构：`select col1,col2 into #t from t where 1=0`；这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：`create table #t(...)`。

13.很多时候用 `exists` 代替 `in` 是一个好的选择：`select num from a where num in(select num from b)`；用下面的语句替换：`select num from a where exists(select 1 from b where num=a.num)`

14.并不是所有索引对查询都有效，SQL 是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL 查询可能不会去利用索引，如一表中有字段 `sex`，`male`、`female` 几乎各一半，那么即使在 `sex` 上建了索引也对查询效率起不了作用。



15. 索引并不是越多越好，索引固然可以提高相应的 `select` 的效率，但同时也降低了 `insert` 及 `update` 的效率，因为 `insert` 或 `update` 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过 6 个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

16. 应尽可能的避免更新 `clustered` 索引数据列，因为 `clustered` 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 `clustered` 索引数据列，那么需要考虑是否应将该索引建为 `clustered` 索引。

17. 尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

18. 尽可能的使用 `varchar/nvarchar` 代替 `char/nchar`，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

19. 任何地方都不要使用 `select * from t`，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

20. 尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

21. 避免频繁创建和删除临时表，以减少系统表资源的消耗。

22. 临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

23. 在新建临时表时，如果一次性插入数据量很大，那么可以使用 `select into` 代替 `create table`，避免造成大量 `log`，以提高速度；如果数据量不大，为了缓和系统表的资源，应先 `create table`，然后 `insert`。

24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 `truncate table`，然后 `drop table`，这样可以避免系统表的较长时间锁定。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过 1 万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

27. 与临时表一样，游标并不是不可使用。对小型数据集使用 `FAST_FORWARD` 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

28.在所有的存储过程和触发器的开始处设置 `SET NOCOUNT ON`，在结束时设置 `SET NOCOUNT OFF`。无需在执行存储过程和触发器的每个语句后向客户端发送 `DONE_IN_PROC` 消息。

29.尽量避免大事务操作，提高系统并发能力。

30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

## 7. 请简述 SQL 注入的原理及如何在代码层面防止 SQL 注入 (2018-5-2-xhq)

详见 MySQL Sql 注入是如何产生的，如何防止。

## 8. 请列出常见的 HTTP 头及其作用 (2018-5-2-xhq)

http 请求中的常用头（请求头）的含义：

Accept：告诉服务器，客户端支持的数据类型。

Accept-Charset：告诉服务器，客户端采用的编码。



Accept-Encoding：告诉服务器，客户机支持的数据压缩格式。

Accept-Language：告诉服务器，客户机的语言环境。

Host：客户机通过这个头告诉服务器，想访问的主机名。

If-Modified-Since:客户机通过这个头告诉服务器，资源的缓存时间。

Referer:客户机通过这个头告诉服务器，它是从哪个资源来访问服务器的。（一般用于防盗链）

User-Agent:客户机通过这个头告诉服务器，客户机的软件环境。

Cookie：客户机通过这个头告诉服务器，可以向服务器带数据。

cookie 是临时文件的意思，保存你浏览网页的痕迹，使得再次上同一页面的时候提高网速，判断你是否登录过这个网站，有些可以帮你自动登录的。

Cookie 就是服务器暂存放在你的电脑里的资料（.txt 格式的文本文件），通过在 HTTP 传输中的状态好让服务器用来辨认你的计算机。当你在浏览网站的时候，Web 服务器会先送一小小资料放在你的计算机上，Cookie 会帮你在网站上所打的文字或是一些选择都记录下来。当下次你再访问同一个网站，Web 服务器会先看看有没有它上次留下的 Cookie 资料，有的话，就会依据 Cookie 里的内容来判断使用者，送出特定的网页内容给你。

http 请求是指从客户端到服务器端的请求消息。包括：消息首行中，对资源的请求方法、资源的标识符及使用的协议。

Connection：客户机通过这个头告诉服务器，请求完后是关闭还是保持链接。

Date：客户机通过这个头告诉服务器，客户机当前请求时间。

http 请求中常用的响应头的含义：

Location:这个头配合 302 状态码使用，告诉用户端找谁。

Server:服务器通过这个头，告诉浏览器服务器的类型

Content-Encoding:服务器通过这个头，告诉浏览器数据采用的压缩格式。

Content-Length:服务器通过这个头，告诉浏览器回送数据的长度。

Content-Language：服务器通过这个头，告诉服务器的语言环境。

Content-Type:服务器通过这个头，回送数据的类型

Last-Modified:服务器通过这个头，告诉浏览器当前资源的缓存时间。

Refresh:服务器通过这个头，告诉浏览器隔多长时间刷新一次。

Content-Disposition:服务器通过这个头，告诉浏览器以下载的方式打开数据。

Transfer-Encoding:服务器通过这个头，告诉浏览器数据的传送格式。

ETag:与缓存相关的头。

Expires:服务器通过这个头，告诉浏览器把回送的数据缓存多长时间。 -1 或 0 不缓存。

Cache-Control 和 Pragma：服务器通过这个头，也可以控制浏览器不缓存数据。

Connection:服务器通过这个头，响应完是保持链接还是关闭链接。

Date:告诉客户机，返回响应的时间。

## 9. 请列举常见的 HTTP 状态码及其意义 ( 2018-5-2-xhq )

成功 2××                  成功处理了请求的状态码。

200                          服务器已成功处理了请求并提供了请求的网页。

204                          服务器成功处理了请求，但没有返回任何内容。

重定向 3××                每次请求中使用重定向不要超过 5 次。

301                          请求的网页已永久移动到新位置。当 URLs 发生变化时，使用 301 代码。搜索引擎索引中保存新的 URL。

302                          请求的网页临时移动到新位置。搜索引擎索引中保存原来的 URL。

304 如果网页自请求者上次请求后没有更新，则用 304 代码告诉搜索引擎机器人，可节省带宽和开销。

客户端错误 4×× 表示请求可能出错，妨碍了服务器的处理。

400 服务器不理解请求的语法。

403 服务器拒绝请求。

404 服务器找不到请求的网页。服务器上不存在的网页经常会返回此代码。

410 请求的资源永久删除后，服务器返回此响应。该代码与 404（未找到）代码相似，但在资源以前存在而现在不存在的情况下，有时用来替代 404 代码。如果资源已永久删除，应当使用 301 指定资源的新位置。

服务器错误 5×× 表示服务器在处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错。

500 服务器遇到错误，无法完成请求。

503 服务器目前无法使用（由于超载或停机维护）。通常，这只是暂时状态。

## 10.请简述 RESTfulAPI 设计规范的理解（2018-5-2-xhq）

### 一、域名

将 api 部署在专用域名下：<http://api.example.com>。

或者将 api 放在主域名下：<http://www.example.com/api/>。

### 二、版本

将 API 的版本号放在 url 中。<http://www.example.com/app/1.0/info>。

### 三、路径

路径表示 API 的具体网址。每个网址代表一种资源。资源作为网址，网址中不能有动词只能有名词，一般名词要与数据库的表名对应。而且名词要使用复数。

错误示例：`http://www.example.com/getGoods`

`http://www.example.com/listOrders`

正确示例：

#获取单个商品

`http://www.example.com/app/goods/1`

#获取所有商品

`http://www.example.com/app/goods`

#### 四、使用标准的 HTTP 方法：

对于资源的具体操作类型，由 HTTP 动词表示。常用的 HTTP 动词有四个。

GET      SELECT ：从服务器获取资源。

POST     CREATE ：在服务器新建资源。

PUT      UPDATE ：在服务器更新资源。

DELETE   DELETE ：从服务器删除资源。

示例：

#获取指定商品的信息

GET `http://www.example.com/goods/ID`

#新建商品的信息

POST `http://www.example.com/goods`

#更新指定商品的信息

PUT `http://www.example.com/goods/ID`

#删除指定商品的信息

DELETE <http://www.example.com/goods/ID>

## 五、过滤信息

如果资源数据较多，服务器不能将所有数据一次全部返回给客户端。API 应该提供参数，过滤返回结果。 实例：

#指定返回数据的数量

<http://www.example.com/goods?limit=10>

#指定返回数据的开始位置

<http://www.example.com/goods?offset=10>

#指定第几页，以及每页数据的数量

[http://www.example.com/goods?page=2&per\\_page=20](http://www.example.com/goods?page=2&per_page=20)

## 六、状态码

服务器向用户返回的状态码和提示信息，常用的有：

200 OK ：服务器成功返回用户请求的数据

201 CREATED ：用户新建或修改数据成功。

202 Accepted ：表示请求已进入后台排队。

400 INVALID REQUEST ：用户发出的请求有错误。

401 Unauthorized ：用户没有权限。

403 Forbidden ：访问被禁止。

404 NOT FOUND ：请求针对的是不存在的记录。

406 Not Acceptable ：用户请求的的格式不正确。

500 INTERNAL SERVER ERROR ：服务器发生错误。

## 七、错误信息

一般来说，服务器返回的错误信息，以键值对的形式返回。

```
{  
  
    error:'Invalid API KEY'  
  
}
```

## 八、响应结果：

针对不同结果，服务器向客户端返回的结果应符合以下规范。

### #返回商品列表

```
GET    http://www.example.com/goods
```

### #返回单个商品

```
GET    http://www.example.com/goods/cup
```

### #返回新生成的商品

```
POST   http://www.example.com/goods
```

### #返回一个空文档

```
DELETE http://www.example.com/goods
```

## 九、使用链接关联相关的资源

在返回响应结果时提供链接其他 API 的方法，使客户端很方便的获取相关联的信息。

## 十、其他

服务器返回的数据格式，应该尽量使用 JSON，避免使用 XML。

## 11. 请简述标准库中 `functools.wraps` 的作用 ( 2018-5-2-xhq )

Python 中使用装饰器对在运行期对函数进行一些外部功能的扩展。但是在使用过程中，由于装饰器的加入导致解释器认为函数本身发生了改变，在某些情况下——比如测试时——会导致一些问题。

Python 通过 `functools.wraps` 为我们解决了这个问题：在编写装饰器时，在实现前加入

`@functools.wraps(func)` 可以保证装饰器不会对被装饰函数造成影响。

## 十五、乐飞天下

### 1. 如何判断一个 python 对象的类型 ( 2018-5-2-xhq )

`type()`

`isinstance()`

### 2. Python 里面如何生成随机数 ( 2018-5-2-xhq )

Python 中的 `random` 函数，可以生成随机浮点数、整数、字符串，甚至帮助你随机选择列表序列中的一个元素，打乱一组数据等。

### 3. 请写出匹配 ip 的 Python 正则表达式 ( 2018-5-2-xhq )

IPv4 的 ip 地址都是 ( 1~255 ) . ( 0~255 ) . ( 0~255 ) . ( 0~255 ) 的格式。

下面给出相对应的正则表达式：

`"^(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|[1-9])\."`

`+(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\\d)\."`

`+(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\\d)\."`

`+(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\\d)$"`

简单的讲解一下：

`\\d` 表示 0~9 的任何一个数字

`{2}`表示正好出现两次

`[0-4]`表示 0~4 的任何一个数字

`|` 的意思是或者

`()`上面的括号不能少，是为了提取匹配的字符串，表达式中有几个`()`就表示有几个相应的匹配字符串

`1\\d{2}`的意思就是 100~199 之间的任意一个数字

`2[0-4]\\d`的意思是 200~249 之间的任意一个数字

`25[0-5]`的意思是 250~255 之间的任意一个数字

`[1-9]\\d`的意思是 10~99 之间的任意一个数字

`[1-9])`的意思是 1~9 之间的任意一个数字

`\\.`的意思是.点要转义（特殊字符类似，@都要加[\\转义](#)）

#### 4. 尽可能多的 str 方法（2018-5-2-xhq）

参见本文第十二章，第四部分。

#### 5. 全局变量和局部变量的区别，如何在 function 里面给一个全局变量赋值 （2018-5-2-xhq）

一、局部变量：在函数内部定义的变量，叫局部变量。

当这个函数被调用的时候，这个变量存在，当这个函数执行完成之后，因为函数都已经结束了，所有函数里面定义的变量也就结束了。

在一个函数中定义的局部变量，只能在这个函数中使用，不能再其他的函数中使用。



二、全局变量:子函数外边定的变量，叫做全局变量。

所有的函数都可以使用它的值，如果函数需要修改全局变量的值，那么需要在这个函数中，使用 `global xxx` 进行说明。

6. Tuple 和 list 的区别，有两个 list `b1 = [1,2,3]` `b2=[2,3,4]`写出合并代码  
( 2018-5-2-xhq )

list :

Python 内置的一种数据类型是列表 list。list 是一种有序的集合，可以随时添加和删除其中的元素。用 `len()`函数可以获得 list 元素的个数，用索引来访问 list 中每一个位置的元素下标从 0 开始，要删除 list 末尾的元素，用 `pop()`方法 要删除指定位置的元素，用 `pop(i)`方法，其中 i 是索引位置，要把某个元素替换成别的元素，可以直接赋值给对应的索引位置，list 里面的元素的数据类型也可以不同，list 元素也可以是另一个 list。

tuple

另一种有序列表叫元组：tuple。tuple 和 list 非常类似，但是 tuple 一旦初始化就不能修改现在，classmates 这个 tuple 不能变了，它也没有 `append()` ,`insert()`这样的方法。其他获取元素的方法和 list 是一样的，你可以正常地使用 `classmates[0]`，`classmates[-1]`，但不能赋值成另外的元素。

不可变的 tuple 有什么意义？因为 tuple 不可变，所以代码更安全。如果可能，能用 tuple 代替 list 就尽量用 tuple。

tuple 的陷阱：当你定义一个 tuple 时，在定义的时候，tuple 的元素就必须被确定下来。

```
1. b1 = [1,2,3]
2. b2=[2,3,4]
3. b1.extend(b2)
4. print(b1)
```

7. 请写出一段代码删除一个 list 里面的重复元素 l=[1,1,2,3,4,5,4](2018-5-2-xhq)

```
1. l = [1, 1, 2, 3, 4, 5, 4]
2. l2=set(l)
3. l = list(l2)
4. print(l)
```

8. 写出 list 的交集与差集的代码 b1=[1,2,3] b2=[2,3,4](2018-5-2-xhq)

```
1. b1=[1,2,3]
2. b2=[2,3,4]
3. b3=[]
4. b4=[]
5. #方法一
6. for val in b2:
7.     if val in b1:
8.         b3.append(val)
9. print(b3)
10. for val in b2:
11.     if val not in b1:
12.         b4.append(val)
13. for val2 in b1:
14.     if val2 not in b2:
15.         b4.append(val2)
16. print(b4)
17. #方法二
18. t1 = set(b1)
19. t2 = set(b2)
20. print(t1&t2)
21. list1 = list(t1-t2)
22. list1.extend(list(t2-t1))
23. print(list1)
```

9. 写一段 Python 代码实现 list 里排 a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3](2018-5-2-xhq)

```
1. a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
2. print(sorted(a,reverse=False))#返回新的列表
3. print(a)
4. a.sort()#注意 sort 没有返回值在原列表上修改
5. print(a)
```

10. `D = [i**2 for i in range(1,10)]` 请写出 D 的输出结果 ( 2018-5-2-xhq )

[1, 4, 9, 16, 25, 36, 49, 64, 81]。

11. 什么时 lambda 函数，下面这段代码的输出是什么 ( 2018-5-2-xhq )

```
1. nums = range(2,20)
2. for i in nums:
3.     nums = filter(lambda x :x == i or x % i,nums)
4. nums
```

nums 为 2 到 19 的数字。

#注解 filter 函数为用于过滤序列函数。

#Python3 中 nums 是一个可迭代对象 迭代后的结果是 2 到 19 的数字。

12. 说说用过的几种爬虫框架及他们的优缺点，用过哪些解析 html 的库( 2018-5-2-xhq )

html 解析库, re jsonpath xpath bs4。

备注：详细见本文《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？》。

13. 谈一下对于多线程的理解，对于 cpu 密集性 IO 密集性怎样使用线程，说说线程池，线程锁的用法，有么有用过 multiprocessing 或者 concurrent.futures?(2018-5-2-xhq)

线程：可以理解成程序中的一个可以执行的分支，它是 cup 调度 0 的基本单元，python 的 thread 模块是比较底层的模块，python 的 threading 模块是对 thread 做了一些包装的，可以更加方便的被使用。

线程池用法：

## 1)、安装

使用安装:

```
pip installthreadpool
```

## 2)、使用

( 1 ) 引入 threadpool 模块

( 2 ) 定义线程函数

( 3 ) 创建线程 池 threadpool.ThreadPool()

( 4 ) 创建需要线程池处理的任务即 threadpool.makeRequests()

( 5 ) 将创建的多个任务 put 到线程池中,threadpool.putRequest

( 6 ) 等到所有任务处理完毕 theadpool.pool()

参考文档:<https://blog.csdn.net/hehe123456zxc/article/details/52258358>



锁的使用：

threading 模块中定义了 Lock 类，可以方便的处理锁定：

# 创建锁

```
mutex = threading.Lock()
```

```
# 锁定
```

```
mutex.acquire()
```

```
# 释放
```

```
mutex.release()
```

参考文档：<https://www.cnblogs.com/xuanan/p/7202492.html>



## 十六、莉莉丝广告开发工程师初始题目

### 1. 用递归实现快速排序 quik\_sort(A)(2018-5-2-xhq)

```
1. def sub_sort(array,low,high):
2.     pivotkey=array[low]
3.     while low<high :
4.         while low<high and array[high]>=pivotkey:
5.             high -= 1
6.         array[low]=array[high]
7.         while low<high and array[low]<=pivotkey:
8.             low += 1
9.         array[high]=array[low]
10.    array[low]=pivotkey
11.    return low
```

```
12.
13. def quick_sort(array,low,high):
14.     if low < high :
15.         pivoloc=sub_sort(array,low,high)
16.         quick_sort(array,low,pivoloc-1)
17.         quick_sort(array,pivoloc+1,high)
18.
19. if __name__=="__main__":
20.     array=[49,38,65,97,76,13,27]
21.     print array
22.     quick_sort(array,0,len(array)-1)
23.     print array
```

## 2. 简述 Python 在异常处理中，else 和 finally 的作用分别是什么？（2018-5-2-xhq）

如果一个 Try - exception 中，没有发生异常，即 exception 没有执行，那么将会执行 else 语句的内容。反之，如果触发了 Try - exception（异常在 exception 中被定义），那么将会执行 exception 中的内容，而不执行 else 中的内容。

如果 try 中的异常没有在 exception 中被指出，那么系统将会抛出 Traceback(默认错误代码)，并且终止程序，接下来的所有代码都不会被执行，但如果有 Finally 关键字，则会在程序抛出 Traceback 之前（程序最后一口气的时候），执行 finally 中的语句。这个方法在某些必须要结束的操作中颇为有用，如释放文件句柄，或释放内存空间等。

## 3. 简述 Python GIL 的概念，以及它对 Python 多线程的影响。如何实现一个抓取网页的程序，使用多线程是否比单线程有性能提升，并解释原因（2018-5-2-xhq）

GIL 锁 全局解释器锁（只在 cpython 里才有）

作用：限制多线程同时执行，保证同一时间只有一个线程执行，所以 cpython 里的多线程其实是伪多线程！

所以 Python 里常常使用协程技术来代替多线程，协程是一种更轻量级的线程，

进程和线程的切换时由系统决定，而协程由我们程序员自己决定，而模块 `gevent` 下切换是遇到了耗时操作才会切换。

会有所提升，因为下载完图片之后进行存储就是 IO 密集性的操作，线程对 IO 密集性的操作有所提升。

## 十七、罗格数据

### 1. 滑动验证码如何解决（2018-5-2-xhq）

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。

## 2. ajax 请求页面如何加载 ( 2018-5-2-xhq )

ajax 可以实现局部刷新，也叫做无刷新，无刷新指的是整个页面不刷新，只是局部刷新，ajax 可以自己发送 http 请求，不用通过浏览器的地址栏，所以页面整体不会刷新，ajax 获取到后台数据，更新页面显示数据的部分，就做到了页面局部刷新。

## 3. Selenium+Phantom JS 的使用(2018-5-2-xhq)

Selenium 是一个 Web 的自动化测试工具，最初是为网站自动化测试而开发的，类型像我们玩游戏用的按键精灵，可以按指定的命令自动操作，不同的是 Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器（包括 PhantomJS 这些无界面的浏览器）。

Selenium 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。

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 JavaScript、Cookie、headers，以及任何我们真实用户需要做的事情。

参考博客一：<https://www.cnblogs.com/luxiaojun/p/6144748.html>





参考博客二：<https://www.jianshu.com/p/4b89c92ff9b4>



#### 4. 数据如何存储，数据库字段如何设计（2018-5-2-xhq）

看数据的用途，如果是分析的话存本地 csv，如果要是用的话存 MongoDB。

## 5. 加密页面如何解析 ( 2018-5-2-xhq )

js 加密参考博客：

<http://baijiahao.baidu.com/s?id=1596146532667769046&wfr=spider&for=pc>



app 加密参考博客：<http://www.site-digger.com/html/articles/20170703/136.html>



## 6. HTTPS 网站如何爬取(2018-5-2-xhq)

1.在使用 requests 前加入：requests.packages.urllib3.disable\_warnings()。

2.为 requests 添加 verify=False 参数，比如：r =

requests.get('https://blog.bbzhhh.com',verify=False)。

## 十八、牧游科技

### 1. 函数参数传递，下面程序运行的结果是？（2018-5-2-xhq）

```
1. def add(a,s_list=[]):
2.     s_list.append(a)
3.     return s_list
4.
5. print(add(1))
6. Print(add(2))
7. print(add(3))
```

结果是[1],[1,2],[1,2,3]。

### 2. Python 中类方法，静态方法的区别及调用（2018-5-2-xhq）

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。

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。

### 3. 类变量，实例变量（2018-5-2-xhq）

```
1. class Person:
2.     name = "aaa"
3. p1 = Person()
4. p2 = Person()
```

```
5. pl.name= " bbb"  
6. print(pl.name,p2.name)  
7. print(Person.name)
```

结果 bbb aaa   aaa

#### 4. 函数式编程与内置函数 ( 2018-5-2-xhq )

```
1. a = [1,2,3,4,5,6,7]  
2. b =filter(lambda x:x>5,a)  
3. for i in b :  
4.     print(i)  
5. a = map(lambda x:x*2,[1,2,3])  
6. print(list(a))
```

第一个 print 返回的是一个可迭代对象 6,7 第二个返回的是 2,4,6。

## 十九、上海金台灯

#### 1. 什么是 lambda 函数，它有什么好处 ( 2018-5-2-xhq )

lambad 表达式就是一个函数，可以赋值给一个变量，既然是表达式，可以参与运算。lambda x: x \*\* 2 这个匿名函数的形参是 x，表达式 x \*\* 2 的值就是这个函数的返回值。

好处：

1.lambda 只是一个表达式，函数体比 def 简单很多。

2.lambda 的主体是一个表达式，而不是一个代码块。仅仅能在 lambda 表达式中封装有限的逻辑进去。

3.lambda 表达式是起到一个函数速写的作用。允许在代码内嵌入一个函数的定义。

#### 2. 什么是 Python 的 list and dict comprehensions,写一段代码 ( 2018-5-2-xhq )

List 是 Python 的列表

```
1. a = list()
2. a.append("2")
```

dict comprehensions 是 Python 的字典的推导式

```
1. mcase = {'a': 10, 'b': 34}
2. mcase_frequency = {v: k for k, v in mcase.items()}
```

### 3. Python 里面如何实现 tuple 和 list 的转换 ( 2018-5-2-xhq )

```
1. list2 = ["2", "3", "4"]
2. t = tuple(list2)
```

### 4. Python 里面如何拷贝一个对象 ( 2018-5-2-xhq )

```
1. list2 = ["2", "3", "4"]
2. q = list2.copy()
3. print(q)
```

### 5. 写一段 except 的函数 ( 2018-5-2-xhq )

```
1. a=10
2. b=0
3. try:
4.     c=a/b
5.     print(c)
6. except Exception as e:
7.     print(e)
8. print("done")
```

### 6. Python 里面 pass 语句的作用是什么 ? (2018-5-2-xhq)

Python pass 是空语句，是为了保持程序结构的完整性。pass 不做任何事情，一般用做占位语句。

### 7. 如何知道 Python 对象的类型 ? (2018-5-2-xhq)

isinstance(变量名, 类型);

type();

## 8. Python 中 range()函数的用法(2018-5-2-xhq)

range 函数大多数时常出如今 for 循环中。在 for 循环中可做为索引使用。事实上它也能够出如今不论什么须要整数列表的环境中，在 python 3.0 中 range 函数是一个迭代器。

## 9. Python re 模块匹配 HTML tag 的的时候，<.\*>和<.\*?>有什么区别(2018-5-2-xhq)

<.\*> 匹配前一个字符 0 或多次

<.\*?> 匹配一个字符 0 次或 1 次

## 10. Python 里面如何生成随机数(2018-5-2-xhq)

```
1. import random
2. random.randint()
```

## 11.如何在 function 里面设置一个全局变量(2018-5-2-xhq)

```
1. globals()
2.
3. globals() 返回包含当前作用域全局变量的字典。
4. global 变量 设置使用全局变量
```

## 12. Python 程序中中文乱码如何解决(2018-5-2-xhq)

```
#coding:utf-8
```

```
sys.setdefaultencoding('utf-8')
```

## 13. Python 的传参是传值还是传址(2018-5-2-xhq)

Python 是传对象引用。

## 14.with 语句的作用,写一段代码(2018-5-2-xhq)

with 语句适用于对资源进行访问的场合,确保不管使用过程中是否发生异常都会执行必要的“清理”操作,释放资源,比如文件使用后自动关闭、线程中锁的自动获取和释放等。

```
1. with open("a.book","w")as f:
```

## 二十、钱方好近

### 1. 求字符串“是一个 test 字符串” 的字符个数(字符编码为 utf8)(2018-5-2-xhq)

```
1. a='是一个 test 字符串'  
2. print(len(a))
```

### 2. 一个 list 对象 a = [1,2,4,3,2,2,3,4]需要去掉里面重复的的值(2018-5-2-xhq)

```
1. a = [1,2,4,3,2,2,3,4]  
2. t =set(a)  
3. print(list(t))
```

### 3. 有一个文件 test.txt 里面有数据(2018-5-2-xhq)

```
1 test    100  2012-04-18  
2 aaa     12   2012-04-19  
3 bbb     333  2012-04-18  
4 ccc     211  2012-04-17  
5 ddd     334  2012-04-16
```

一共有 5 行 4 列数据,最后一列为日期,按日期大小进行排序。

```
1. a=["2012-04-18",  
2. "2012-04-19",  
3. "2012-04-18",  
4. "2012-04-17",
```

```
5. "2012-04-16"]
6. import datetime
7. def date_sort3(x):
8.     ls=list(x)
9.     #用了冒泡排序来排序，其他方法效果一样
10.    for j in range(len(ls)-1):
11.        for i in range(len(ls)-j-1):
12.            lower=datetime.datetime.strptime(ls[i], '%Y-%m-%d')
13.            upper=datetime.datetime.strptime(ls[i+1], '%Y-%m-%d')
14.            if lower>upper:
15.                ls[i],ls[i+1]=ls[i+1],ls[i]
16.    return tuple(ls)
17. print(date_sort3(a))
```

## 二十一、西北莜面村

### 1. 列举出一些常用的设计模式？(2018-5-11-lxy)

创建型：

1. Factory Method ( 工厂方法 )
2. Abstract Factory ( 抽象工厂 )
3. Builder ( 建造者 )
4. Prototype ( 原型 )
5. Singleton ( 单例 )

结构型：

6. Adapter Class/Object ( 适配器 )
7. Bridge ( 桥接 )



8. Composite ( 组合 )

9. Decorator ( 装饰 )

10. Facade ( 外观 )

11. Flyweight ( 享元 )

12. Proxy ( 代理 )

行为型：

13. Interpreter ( 解释器 )

14. Template Method ( 模板方法 )

15. Chain of Responsibility ( 责任链 )

16. Command ( 命令 )

17. Iterator ( 迭代器 )

18. Mediator ( 中介者 )

19. Memento ( 备忘录 )

20. Observer ( 观察者 )

21. State ( 状态 )

22. Strategy ( 策略 )

23. Visitor ( 访问者 )

## 2. Python 关键字 yield 的用法 ? (2018-5-11-lxy)

yield 就是保存当前程序执行状态。你用 for 循环的时候，每次取一个元素的时候就会计算一次。用 yield 的函数叫 generator 和 iterator 一样，它的好处是不用一次计算所有元素，而是用一次算一次，可以节省很多空间。generator 每次计算需要上一次计算结果，所以用 yield，否则一 return，上次计算结果就没了。

```
1. >>> def createGenerator():
2. ...     mylist = range(3)
3. ...     for i in mylist:
4. ...         yield i*i
5. ...
6. >>> mygenerator = createGenerator() # create a generator
7. >>> print(mygenerator) # mygenerator is an object!
8. <generator object createGenerator at 0xb7555c34>
9. >>> for i in mygenerator:
10. ...     print(i)
11. 0
12. 1
13. 4
```

## 3. 深拷贝，浅拷贝的区别 ? (2018-5-11-lxy)

详情请见 python 基础-模块与包-5.2。

## 4. 简化代码 ? (2018-5-11-lxy)

```
1. l = []
2. for i in range(10):
3.     l.append(i**2)
4. print l
```

简化后的来：

```
1. print([x**2 for x in range(10)])
```

## 5. 解释 list dict tuple set 区别 ? (2018-5-11-lxy)

详情见 python 基础-数据结构。

## 6. 这两个参数什么意思 \*args \*\*kwargs , 我们为什么要使用他们 ? (2018-5-11-lxy)

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。

\*args 是不定长参数，他可以表示输入参数是不确定的，可以是任意多个。

\*\*kwargs 是关键字参数，赋值的时候是以键 = 值的方式，参数是可以任意多对在定义函数的时候不确定会有多少参数会传入时，就可以使用两个参数。

## 7. 数据库连表查询 ? (2018-5-11-lxy)

1. 内连接：inner join on ;
2. 左连接：left join on ;
3. 右连接：right join on ;

## 二十二、浙江从泰

### 1. 数据库优化 ? (2018-5-11-lxy)

详情请见第九章-数据库-Mysql-16。

### 2. 反爬虫措施 ? (2018-5-11-lxy)

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。

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，例如同一个 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。

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。

### 3. 分布式爬虫原理？(2018-5-11-lxy)

scrapy-redis 实现分布式，其实从原理上来说很简单，这里为描述方便，我们把自己的核心服务器称为 master，而把用于跑爬虫程序的机器称为 slave。

我们知道，采用 scrapy 框架抓取网页，我们需要首先给定它一些 start\_urls，爬虫首先访问 start\_urls 里面的 url，再根据我们的具体逻辑，对里面的元素、或者是其他的二级、三级页面进行抓取。而要实现分布式，我们只需要在这个 start\_urls 里面做文章就行了。

我们在 master 上搭建一个 redis 数据库（注意这个数据库只用作 url 的存储，不关心爬取的具体数据，不要和后面的 mongodb 或者 mysql 混淆），并对每一个需要爬取的网站类型，都开辟一个单独的列表字段。通过设置 slave 上 scrapy-redis 获取 url 的地址为 master 地址。这样的结果就是，尽管有多个 slave，然而大家获取 url 的地方只有一个，那就是服务器 master 上的 redis 数据库。

并且，由于 scrapy-redis 自身的队列机制，slave 获取的链接不会相互冲突。这样各个 slave 在完成抓取任务之后，再把获取的结果汇总到服务器上（这时的数据存储不再是在 redis，而是 mongodb 或者 mysql 等存放具体内容的数据库了）

这种方法的还有好处就是程序移植性强，只要处理好路径问题，把 slave 上的程序移植到另一台机器上运行，基本上就是复制粘贴的事情。

## 二十三、tataUFO

1. 将字符串："k:1|k1:2|k2:3|k3:4"，处理成 Python 字典：{k:1, k1:2, ...} # 字典里的 K 作为字符串处理(2018-5-11-lxy)

```
1. str1 = "k:1|k1:2|k2:3|k3:4"
2. def str2dict(str1):
3.     dict1 = {}
4.     for items in str1.split('|'):
5.         key, value = items.split(':')
6.         dict1[key] = value
7.     return dict1
```

2. 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？ (2018-5-11-lxy)

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

3. 写一个装饰器？(2018-5-11-lxy)

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。

```
1.  #! coding=utf-8
2.  import time
3.  def timeit(func):
4.      def wrapper(a):
5.          start = time.clock()
6.          func(1,2)
```

```
7.     end =time.clock()
8.     print 'used:', end - start
9.     print a
10.    return wrapper
11. # foo = timeit(foo)完全等价,
12. # 使用之后,foo 函数就变了,相当于是 wrapper 了
13. @timeit
14. def foo(a,b):
15.     pass
16. #不带参数的装饰器
17. # wraper 将 fn 进行装饰, return wraper ,返回的 wraper 就是装饰之后的 fn
18. def test(func):
19.     def wraper():
20.         print "test start"
21.         func()
22.         print "end start"
23.     return wraper
24. @test
25. def foo():
26.     print "in foo"
27. foo()
28. 输出 :
29. test start
30. in foo
31. end start
```

#### 4. Python 中可变类型和不可变类型有哪些 ? (2018-5-11-lxy)

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。

## 二十四、全品教育

### 1. Tuple 和 list 区别 ( 2018-5-11-xhq )

Python 内置的一种数据类型是列表 :list。list 是一种有序的集合 ,可以随时添加和删除其中的元素。

另一种有序列表叫元组 :tuple。tuple 和 list 非常类似 ,但是 tuple 一旦初始化就不能修改

### 2. 这两个参数\*args \*\*kwargs 是什么意思 ( 2018-5-11-xhq )

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。

\*args 是不定长参数 ,他可以表示输入参数是不确定的 ,可以是任意多个。

\*\*kwargs 是关键字参数 ,赋值的时候是以键 = 值的方式 ,参数是可以任意多对在定义函数的时候不确定会有多少参数会传入时 ,就可以使用两个参数。

### 3. Python 里面如何实现 tuple 和 list 的转换 ( 2018-5-11-xhq )

```
1. t =(1,5,8)
2. l = list(t)
```

### 4. Python 里面 range 和 xrange 的区别(2018-5-11-xhq)

xrange 和 range 的参数和用法是相同的。只是 xrange()返回的不再是一个数列 ,而是一个 xrange 对象。这个对象可以按需生成参数指定范围内的数字 ( 即元素 )。由于 xrange 对象是按需生成单个的元素 ,而不像 range 那样 ,首先创建整个 list。所以 ,在相同的范围内 ,xrange 占用的内存空间将更小 ,xrange 也会更快。实际上 ,xrange 由于是在循环内被调用时才会生成元素 ,因此无论循环多少次 ,只有当前一个元素占用了内存空间 ,且每次循环占用的都是相同的单个元素空间。我们可以粗略的认为 ,



相同  $n$  个元素的话，`range` 占用的空间是 `xrange` 的  $n$  倍。因此，在循环很大情况下，`xrange` 的高效率和快速将表现的很明显。

注意：Python3 中已经没有 `xrange` 了。

## 5. Python 里面 `classmethod` 和 `staticmethod` 的区别 (2018-5-11-xhq)

如果在 `@staticmethod` 中要调用到这个类的一些属性方法，只能直接类名.属性名或类名.方法名。

而 `@classmethod` 因为持有 `cls` 参数，可以来调用类的属性，类的方法，实例化对象等。

## 6. 如何反向输出序列比如[2,6,5,3],输出为[3,5,6,2](2018-5-11-xhq)

```
1. l = [2,6,5,3]
2. l.reverse()
3. print(l)
```

## 7. Python 里面实现删除重复的元素(2018-5-11-xhq)

```
1. lis = [1,1,2,3,1,2,1,3,2,4,5,5,4]
2. t = set(lis)
3. print(t)
```

## 8. Python 里面 `copy` 和 `deepcopy` 的区别(2018-5-11-xhq)

`deepcopy`（深复制），即将被复制对象完全再复制一遍作为独立的新个体单独存在。所以改变原有被复制对象不会对已经复制出来的新对象产生影响。

而等于（`=`）赋值，并不会产生一个独立的对象单独存在，他只是将原有的数据块打上一个新标签，所以当其中一个标签被改变的时候，数据块就会发生变化，另一个标签也会随之改变。

而 `copy`（浅复制）要分两种情况进行讨论：

1) 当浅复制的值是不可变对象（数值，字符串，元组）时和“等于赋值”的情况一样，对象的 `id` 值与浅复制原来的值相同。

2) 当浅复制的值是可变对象 ( 列表和元组 ) 时会产生一个 “不是那么独立的对象” 存在。有两种情况：

第一种情况：复制的 对象中无 复杂 子对象，原来值的改变并不会影响浅复制的值，同时浅复制的值改变也并不会影响原来的值。原来值的 id 值与浅复制原来的值不同。

第二种情况：复制的对象中有 复杂 子对象（例如列表中的一个子元素是一个列表），如果不改变其中复杂子对象，浅复制的值改变并不会影响原来的值。但是改变原来的值 中的复杂子对象的值 会影响浅复制的值。

对于简单的 object，例如不可变对象（数值，字符串，元组），用 shallow copy 和 deep copy 没区别。复杂的 object，如 list 中套着 list 的情况，shallow copy 中的 子 list，并未从原 object 真的「独立」出来。也就是说，如果你改变原 object 的子 list 中的一个元素，你的 copy 就会跟着一起变。这跟我们直觉上对「复制」的理解不同。

## 9. Python 里面的 search 和 match 的区别(2018-5-11-xhq)

match ( ) 函数只检测 RE 是不是在 string 的开始位置匹配，search() 会扫描整个 string 查找匹配，也就是说 match ( ) 只有在 0 位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match() 就返回 none。

## 10. 输出下列代码的结果 ( 2018-5-11-xhq )

```
1. class Parent(object):
2.     x=1
3. class Child1(Parent):
4.     pass
5. class Child2(Parent):
6.     pass
7. print(Parent.x,Child1.x,Child2.x)
8. Child1.x=2
9. print(Parent.x,Child1.x,Child2.x)
```

```
10. Parent.x=3
11. print(Parent.x,Child1.x,Child2.x)
```

结果 1 1 1 , 1 2 1 , 3 2 3

### 11. Python 代码如何得到列表的交集和差集？ ( 2018-5-11-xhq )

```
1. b1 =[1,2,3]
2. b2 =[2,3,4]
3. t1 = set(b1)
4. t2 =set(b2)
5. t3 =t1&t2
6. t4 = t1-t2
7. t5 = t2-t1
8. print(list(t3))
9. print(list(t4))
10. print(list(t5))
```

### 12. 请写出一段代码求出 1 到 100 的和？ (2018-5-11-xhq)

```
1. i =1
2. su=0
3. while i <=100:
4.     su =su+i
5.     i+=1
6. print(su)
```

### 13. Python 中正则表达式提取出字符串中的数字(2018-5-11-xhq)

```
1. s = '12j33jk12ksdjfkj23jk4h1k23h'
2. import re
3. b=re.findall("\d",s)
4. b="".join(b)
5. print(b)
```

### 14. 补全下列代码(2018-5-11-xhq)

```
1. def deco(func):
2.     pass "补全代码"
```

```
3. @deco
4. def myfunc(a,b):
5.     print("myfunc(%s,%s) called" %(a,b))
6.     return a+b
```

补充后：

```
1. def deco(func):
2.     def inner(a,b):
3.         return func(a,b)
4.     return inner
5.
6. @deco
7. def myfunc(a,b):
8.     print("myfunc(%s,%s) called" %(a,b))
9.     return a+b
```

## 二十五、名企爬虫面试题

1. 简述一次完整的 http 的通信过程、常用的响应状态码、http 的无状态性、Cookies 等这些概念(2018-5-11-xhq)

### 一、http 过程

HTTP 通信机制是在一次完整的 HTTP 通信过程中，Web 浏览器与 Web 服务器之间将完成下列 7 个步骤：

#### 1. 建立 TCP 连接

在 HTTP 工作开始之前，Web 浏览器首先要通过网络与 Web 服务器建立连接，该连接是通过 TCP 来完成的，该协议与 IP 协议共同构建 Internet，即著名的 TCP/IP 协议族，因此 Internet 又被称作是 TCP/IP 网络。HTTP 是比 TCP 更高层次的应用层协议，根据规则，只有低层协议建立之后才能进行更高层协议的连接，因此，首先要建立 TCP 连接，一般 TCP 连接的端口号是 80。

#### 2. Web 浏览器向 Web 服务器发送请求命令

一旦建立了 TCP 连接 ,Web 浏览器就会向 Web 服务器发送请求命令。例如 :GET/sample/hello.jsp HTTP/1.1。

### 3. Web 浏览器发送请求头信息

浏览器发送其请求命令之后，还要以头信息的形式向 Web 服务器发送一些别的信息，之后浏览器发送了一空白行来通知服务器，它已经结束了该头信息的发送。

### 4. Web 服务器应答

客户机向服务器发出请求后，服务器会客户机回送应答， HTTP/1.1 200 OK ，应答的第一部分是协议的版本号和应答状态码。

### 5. Web 服务器发送应答头信息

正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。

### 6. Web 服务器向浏览器发送数据

Web 服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以 Content-Type 应答头信息所描述的格式发送用户所请求的实际数据。

### 7. Web 服务器关闭 TCP 连接

一般情况下，一旦 Web 服务器向浏览器发送了请求数据，它就要关闭 TCP 连接，然后如果浏览器或者服务器在其头信息加入了这行代码：Connection:keep-alive

TCP 连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

## 二、常见的响应状态码

1 . 200 301 302 404 500。

## 三、无状态

无状态是指协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。即我们给服务器发送 HTTP 请求之后，服务器根据请求，会给我们发送数据过来，但是，发送完，不会记录任何信息。

#### 四、Cookie

Cookie 是由 HTTP 服务器设置的，保存在浏览器中，但 HTTP 协议是一种无状态协议，在数据交换完毕后，服务器端和客户端的链接就会关闭，每次交换数据都需要建立新的链接。就像我们去超市买东西，没有积分卡的情况下，我们买完东西之后，超市没有我们的任何消费信息，但我们办了积分卡之后，超市就有了我们的消费信息。cookie 就像是积分卡，可以保存积分，商品就是我们的信息，超市的系统就像服务器后台，http 协议就是交易的过程。

## 2. 说说进程和线程和锁之间的关系(2018-5-17-ydy)

### 一、进程

首先进程是指在系统中正在运行的一个应用程序；程序一旦运行就是进程，或者更专业化来说：进程是指程序执行时的一个实例，即它是程序已经执行到课中程度的数据结构的汇集。从内核的观点看，进程的目就是担当分配系统资源（CPU 时间、内存等）的基本单位，进程有五方面的特点：第一：动态性:进程的实质是程序的一次执行过程，进程是动态产生，动态消亡的。第二：并发性:任何进程都可以同其他进程一起并发执行第三：独立性:进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位;第四：异步性:由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进第五：结构特征:进程由程序、数据和进程控制块三部分组成。进程可以使用 fork（）函数来创建子进程也可以使用 vfork（）来实现进程，使用的时候注意不要产生僵尸进程和孤儿进程。

### 二、线程

线程是系统分配处理器时间资源的基本单元，或者说进程之内独立执行的一个单元执行流，线程有四个方面特点：第一，线程有独立的堆栈段，共享地址空间，开销较小，切换速度较快。第二，线程间的通信机制比较方便。第三，因为操作系统会保证当线程数不大于 CPU 数目时，不同的线程运行于不同的 CPU 上。线程使 CPU 系统更加有效。第四，线程改善了程序结构，避免了一些嵌套循环。使用 `pthread_create()` 函数来创建线程，使用线程的时候有两点注意事项：第一，当多线程访问同一全局变量的时候，一定要加互斥量，也就是上锁。当然最后不要忘记了解锁。第二：正确处理线程结束的问题：因为一个线程的终止，线程的资源不会随线程的终止释放，我们需要调用 `pthread_join()` 来获得另一个线程的终止状态并且释放该线程所占的资源。

一个程序至少有一个进程，一个进程至少有一个线程。线程不能够独立执行，必须依存在进程中。

### 三、锁

当多线程访问同一全局变量的时候，一定要加互斥量，也就是上锁。

## 3. MySQL 操作：为 person 表的 name 创建普通的索引(2018-5-11-xhq)

```
1. CREATE INDEX name ON table_name (person)
```

## 4. \*args and \*\*kwargs 的区别(2018-5-11-xhq)

在函数定义中使用 `*args` 和 `kwargs` 传递可变长参数。 `*args` 用作传递非命名键值可变长参数列表 (位置参数)； `kwargs` 用作传递键值可变长参数列表，并且 `*args` 必须位于 `**kwargs` 之前，因为 positional arguments 必须位于 keyword arguments 之前。

## 5. 写一个匹配 Email 地址的正则表达式(2018-5-11-xhq)

```
1. 只允许英文字母、数字、下划线、英文句号、以及中划线组成
2. ^[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$
3. 名称允许汉字、字母、数字，域名只允许英文域名
4. ^[A-Za-z0-9\u4e00-\u9fa5]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$
```

## 6. 常见的反爬虫措施有哪些？一般怎么克服(2018-5-11-xhq)

### 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，例如同一个 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 Firebug 或者 HttpFox 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。我这几天爬的那个网站就是这样，除了加密 ajax 参数，它还把一些基本的功能都封装了，全部都是在调用自己的接口，而接口参数都是加密的。遇到这样的网站，我们就不能用上面的方法了，我用的是 selenium+phantomJS 框架，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。利用 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等。

## 7. 编写爬虫的常用模块或者框架有哪些?请说明一个爬虫的行为步骤(2018-5-11-ydy)

爬虫的行为步骤：

- 1、获取网页
- 2、提取数据
- 3、高效抓取数据
- 4、持续抓取数据（增量式爬虫）
- 5.爬虫和反爬虫和反反爬虫

推荐参考文档：<https://blog.csdn.net/zangker/article/details/77864701>



## 8. 排序算法有哪些用 Python 写一种排序算法(2018-5-11-xhq)

冒泡排序：

```
1.  #!/usr/bin/python
2.  # -*- coding: utf-8 -*-
3.  import random
4.  unsortedList=[]
5.  # generate an unsorted list
6.  def generateUnsortedList(num):
7.      for i in range(0,num):
8.          unsortedList.append(random.randint(0,100))
9.      print unsortedList
10. # 冒泡排序
11. def bubbleSort(unsortedList):
12.     list_length=len(unsortedList)
13.     for i in range(0,list_length-1):
14.         for j in range(0,list_length-i-1):
15.             if unsortedList[j]>unsortedList[j+1]:
16.                 unsortedList[j],unsortedList[j+1]=unsortedList[j+1],unsortedList[j]
17.     return unsortedList
18. generateUnsortedList(20)
19. print bubbleSort(unsortedList)
```

# 选择排序

```
1.  def selectionSort(unsortedList):
2.      list_length=len(unsortedList)
```

```
3.     for i in range(0,list_length-1):
4.         for j in range(i+1,list_length):
5.             if unsortedList[i]>unsortedList[j]:
6.                 unsortedList[i],unsortedList[j]=unsortedList[j],unsortedList[i]
7.     return unsortedList
```

快排：

```
1.  def quickSort(unsortedList):
2.      if len(unsortedList)<2:
3.          return unsortedList
4.      less=[]
5.      greater=[]
6.      middle=unsortedList.pop(0)
7.      for item in unsortedList:
8.          if item<middle:
9.              less.append(item)
10.         else:
11.             greater.append(item)
12.     return quickSort(less)+[middle]+quickSort(greater)
```

归并排序：

```
# 归并排序
1.  def mergeSort(unsortedList):
2.      if len(unsortedList)<2:
3.          return unsortedList
4.      sortedList=[]
5.      left=mergeSort(unsortedList[:len(unsortedList)/2])
6.      right=mergeSort(unsortedList[len(unsortedList)/2:])
7.      while len(left)>0 and len(right)>0:
8.          if left[0]<right[0]:
9.              sortedList.append(left.pop(0))
10.         else:
11.             sortedList.append(right.pop(0))
12.     if len(left)>0:
13.         sortedList.extend(mergeSort(left))
14.     else:
15.         sortedList.extend(mergeSort(right))
16.     return sortedList
```

## 二十六、欧特咨询

### 1. 对 Cookie 的理解，遇到没有 Cookie 登陆的问题(2018-5-11-xhq)

Cookie 是由 HTTP 服务器设置的，保存在浏览器中，但 HTTP 协议是一种无状态协议，在数据交换完毕后，服务器端和客户端的链接就会关闭，每次交换数据都需要建立新的链接。就像我们去超市买东西，没有积分卡的情况下，我们买完东西之后，超市没有我们的任何消费信息，但我们办了积分卡之后，超市就有了我们的消费信息。Cookie 就像是积分卡，可以保存积分，商品就是我们的信息，超市的系统就像服务器后台，http 协议就是交易的过程。

如果遇到没有 Cookie 的话可以借助 phantomjs selenium 进行登陆。

### 2. 如何解决验证码的问题，用什么模块，听过哪些人工打码平台(2018-5-11-xhq)

PIL、pytesseract、tesseract。

平台的话有：云打码、答题吧打码平台、挣码、斐斐打码、若快打码。

### 3. 对于 scrapy\_redis 的理解(2018-5-11-xhq)

scrapy-redis 是一个基于 redis 的 scrapy 组件，通过它可以快速实现简单分布式爬虫程序，该组件本质上提供了三大功能：

scheduler - 调度器

dupefilter - URL 去重规则（被调度器使用）

pipeline - 数据持久化

#### 一、scrapy-redis 组件

1. URL 去重 去重规则通过 redis 的集合完成，去重规则中将 url 转换成唯一标示，然后在 redis 中检查是否已经在集合中存在

2. 调度器 调度器，调度器使用 PriorityQueue（有序集合）、FifoQueue（列表）、LifoQueue（列表）进行保存请求，并且使用 RFPDufilter 对 URL 去重

3. 数据持久化

定义持久化，爬虫 yield Item 对象时执行 RedisPipeline 将 item 持久化到 redis 时，指定 key 和序列化函数 REDIS\_ITEMS\_KEY = '%(spider)s:items' REDIS\_ITEMS\_SERIALIZER = 'json.dumps'  
使用列表保存 item 数据

4. 起始 URL 相关

获取起始 URL 时，去集合中获取还是去列表中获取？True，集合；False，列表

REDIS\_START\_URLS\_AS\_SET = False # 获取起始 URL 时 如果为 True 则使用 self.server.spop；  
如果为 False，则使用 self.server.lpop。

编写爬虫时，起始 URL 从 redis 的 Key 中获取。

4. ip 被封了怎么解决，自己做过 ip 池么？(2018-5-11-xhq)

关于 ip 可以通过 ip 代理池来解决问题 ip 代理池相关的可以在 github 上搜索 ip proxy 自己选一个去说 <https://github.com/awolfly9/IPProxyTool> 提供大体思路：

1. 获取器 通过 requests 的爬虫爬取免费的 IP 代理网址获取 IP。
2. 过滤器通过获取器获取的代理请求网页数据有数据返回的保存进 Redis。
3. 定时检测器定时拿出一部分 Proxy 重新的用过滤器进行检测剔除不能用的代理。
4. 利用 Flask web 服务器提供 API 方便提取 IP



## 二十七、多来点

1. 请阐述 Python 的特点(2018-5-8-zcz)

- 1) 面向对象
- 2) 免费
- 3) 开源
- 4) 可移植
- 5) 功能强大
- 6) 可混合
- 7) 简单易用

.....

2. Python 字典参数如何传递？(2018-5-8-zcz)



3. 请举例说明 Python 闭包的使用场景(2018-5-8-zcz)



#### 4. 阐述 Python 变量作用域(2018-5-8-zcz)



## 二十八、傲盾

#### 1. Python 线程和进程的区别？(2018-5-8-zcz)

1)调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位。

2)并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行。

3)拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源。

4)系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

#### 2. 如何保证线程安全？(2018-5-8-zcz)

通常加锁也有 2 种不同的粒度的锁：

1. fine-grained(细粒度)，程序员需要自行加/解锁来保证线程安全



2. coarse-grained(粗粒度)，语言层面本身维护着一个全局的锁机制用来保证线程安全

前一种方式比较典型的是 Java, Jython 等, 后一种方式比较典型的是 CPython (即 Python)。

### 3. 编程实现 list 转 dict(2018-5-8-zcz)

```
1. # 方法一，使用 zip 函数
2. a = ['hello', 'world', '1', '2']
3. b = dict(zip(a[0::2], a[1::2]))
4. print(b)
5.
6. # 方法二，利用循环
7. b = {}
8. for i in range(0, len(a), 2):
9.     b[a[i]] = a[i+1]
10. print(b)
11.
12. # 使用 enumerate 函数生成 index 实现
13. my_dict = {}
14. for index, item in enumerate(a):
15.     if index % 2 == 0:
16.         my_dict[item] = a[index+1]
17. print(b)
```

### 4. 编程实现两个 list 的交集、并集、差集(2018-5-8-zcz)



## 二十九、汉迪

### 1. 在 Python 中，list，tuple，dict，set 有什么区别，主要应用在什么场景？

(2018-5-8-zcz)

区别：

list：链表,有序的数据结构, 通过索引进行查找,使用方括号“ []”；

tuple：元组,元组将多样的对象集合到一起,不能修改,通过索引进行查找, 使用括号“ ()”；

dict：字典,字典是一组键(key)和值(value)的组合,通过键(key)进行查找,没有顺序, 使用大括号“ {}”；

set：集合,无序,元素只出现一次, 自动去重,使用“ set([])”

应用场景:

list：简单的数据集合,可以使用索引;

tuple：把一些数据当做一个整体去使用,不能修改;

dict：使用键值和值进行关联的数据;

set：数据只出现一次,只关心数据是否出现, 不关心其位置。

### 2. 说明 Session 和 Cookie 的联系(2018-5-8-zcz)

Session 对 Cookie 的依赖：Cookie 采用客户端存储，Session 采用的服务端存储的机制。Session 是针对每个用户（浏览器端）的，Session 值保存在服务器上，通过 SessionId 来区分哪个用户的 Session。因此 SessionId 需要被绑定在浏览器端。SessionId 通常会默认通过 Cookie 在浏览器端绑定，当浏览器端禁用 cookie 时，可通过 Url 重写（可以在地址栏看到 sessionid=KWJHUG

6JJM65HS2K6 之类的字符串) 或者表单隐藏字段的方式将 SessionId 传回给服务器, 以便服务通过 SessionId 获取客户端对应的 Session。

具体一次的请求流程: 当程序需要为客户端创建一个 Session 的时候, 服务器首先检测这个客户端请求里面是否已经包含了 Session 的表示 (SessionId), 如果已经包含, 则说明已经为客户端创建过一个 Session, 服务端根据 SessionId 检索出来 Session 并使用。如果客户端请求不包含 SessionId, 则为客户端创建一个 Session, 并生成一个 SessionId 返回给客户端保存。

### 3. 说明 Session 和 Cookie 的区别(2018-5-8-zcz)

- 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, 会比较占用你服务器的性能, 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、可以考虑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。

#### 4. 简述 decorator 的用法(2018-5-8-zcz)



### 三十、大会信统 Python 工程师 ( 2018-5-9-lyf )

#### 1. 请写出一段 python 代码实现删除一个 list 里面的重复元素

方法一：利用 Set 集合去重实现。

```
1. l1 = ['b','c','d','b','c','a','a']
2. l2 = list(set(l1))
3. l2.sort(key=l1.index)
4. print l2
```

方法二：使用字典函数。

```
1. a=[1,2,4,2,4,5,6,5,7,8,9,0]
2. b={}
3. b=b.fromkeys(a)
4. c=list(b.keys())
5. print('去重后的 list 为：',c)
```

方法三：用 append 方法实现。

```
1. def delList(L):
2.     L1 = []
3.     for i in L:
```

```
4.         if i not in L1:
5.             L1.append(i)
6.
7.     return L1
8.
9. print(delList([1,2,2,3,3,4,5]))
10. print(delList([1,8,8,3,9,3,3,3,3,3,6,3]))
```

换成列表推导式更简单：

```
1. l1 = ['b','c','d','b','c','a','a']
2. l2 = []
3. [l2.append(i) for i in l1 if not i in l2]
4. print l2
```

方法四：先对元素进行排序,然后从列表的最后开始扫描。

```
1. List=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
2. if List:
3.     List.sort()
4.     #print List
5.     last = List[-1]
6.     #print last
7.     for i in range(len(List)-2, -1, -1):
8.         if last==List[i]:
9.             del List[i]
10.        else:
11.            last=List[i]
12.            #print(list)
13.    print(List)
18.
```

## 2. 编程用 sort 进行排序，然后从最后一个元素开始判断

```
1. a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
2. a.sort()
3. last=a[-1]
4. for i in range(len(a)-2, -1, -1):
5.     if last==a[i]:
6.         del a[i]
7.     else:
8.         last=a[i]
9. print(a)
```

### 3. Python 里面如何拷贝一个对象？（赋值，浅拷贝，深拷贝的区别）

- 1) 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。
- 2) 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}。
- 3) 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 copy.deepcopy()函数}。

### 4. Python 里面 match()和 search()的区别？

re 模块中 match(pattern,string[,flags]),检查 string 的开头是否与 pattern 匹配。

re 模块中 research(pattern,string[,flags]),在 string 搜索 pattern 的第一个匹配值。

```
1. >>>print(re.match( 'super' , 'superstition' ).span())
2.
3. (0, 5)
4.
5. >>>print(re.match( 'super' , 'insuperable' ))
6.
7. None
8.
9. >>>print(re.search( 'super' , 'superstition' ).span())
10.
11. (0, 5)
12.
13. >>>print(re.search( 'super' , 'insuperable' ).span())
14.
15. (2, 7)
```

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

术语叫贪婪匹配( <.\*> )和非贪婪匹配(<.\*?> )。

例如：`test<.*>`：      `test<.*?>`：

`<.*>`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“<>”。

## 6. Python 里面如何生成随机数？

使用 random 模块。

1) 随机整数：`random.randint(a,b)`：返回随机整数  $x, a \leq x \leq b$

`random.randrange(start,stop,[step])`：返回一个范围在(start,stop,step)之间的随机整数，不包括结束值。

2) 随机实数：`random.random()`：返回 0 到 1 之间的浮点数

`random.uniform(a,b)`：返回指定范围内的浮点数。

## 三十一、倍通供应链 信息&数据中心工程师 笔试题 ( 2018-5-9 lyf )

### 1. OOP 编程三大特点是什么，多态应用的基础是什么？

1) 封装：就是将一个类的使用和实现分开，只保留部分接口和方法与外部联系。

2) 继承：子类自动继承其父级类中的属性和方法，并可以添加新的属性和方法或者对部分属性和方法进行重写。继承增加了代码的可重用性。

3) 多态：多个子类中虽然都具有同一个方法，但是这些子类实例化的对象调用这些相同的方法后却可以获得完全不同的结果，多态性增强了软件的灵活性。（多态的概念依赖于继承）。

## 2. 请描述抽象类和接口类的区别和联系？

1) 抽象类：规定了一系列的方法，并规定了必须由继承类实现的方法。由于有抽象方法的存在，所以抽象类不能实例化。可以将抽象类理解为毛坯房，门窗、墙面的样式由你自己来定，所以抽象类与作为基类的普通类的区别在于约束性更强。

2) 接口类：与抽象类很相似，表现在接口中定义的方法，必须由引用类实现，但他与抽象类的根本区别在于用途：与不同个体间沟通的规则（方法），你要进宿舍需要有钥匙，这个钥匙就是你与宿舍的接口，你的同室也有这个接口，所以他也能进入宿舍，你用手机通话，那么手机就是你与他人交流的接口。

3) 区别和关联：

1. 接口是抽象类的变体，接口中所有的方法都是抽象的。而抽象类中可以有非抽象方法。抽象类是声明方法的存在而不去实现它的类。

2. 接口可以继承，抽象类不行。

3. 接口定义方法，没有实现的代码，而抽象类可以实现部分方法。

4. 接口中基本数据类型为 static 而抽象类不是。

5. 接口可以继承，抽象类不行。

6. 可以在一个类中同时实现多个接口。

7. 接口的使用方式通过 implements 关键字进行，抽象类则是通过继承 extends 关键字进行。

## 3. 请解释委托的定义和作用？

委托：假装这件事是我在做，但是事实上我委托了其他人来帮我处理这件事。（Python 中的委托与此相似。）

详细了解可以扫下面二维码：





#### 4. 请描述方法重载与方法重写？

1) 方法重载：是在一个类里面，方法名字相同，而参数不同。返回类型呢？可以相同也可以不同。

重载是让类以统一的方式处理不同类型数据的一种手段。

2) 方法重写：子类不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖。

#### 5. 请用代码实现一个冒泡排序？

```
1. def bubbleSort(nums):
2.     for i in range(len(nums)-1): # 这个循环负责设置冒泡排序进行的次数
3.         for j in range(len(nums)-i-1): # j 为列表下标
4.             if nums[j] > nums[j+1]:
5.                 nums[j], nums[j+1] = nums[j+1], nums[j]
6.     return nums
7. nums = [5,2,45,6,8,2,1]
8. print ( bubbleSort(nums) ) #python3 写法
```

#### 6. 请用代码实现输出：1, 2, 3, 5, 8, 13, 21, 34, 55, 89.....

这道题考的是斐波那契数列的实现。

用生成器实现：

```
1. class FibIterator(object):
2.     """斐波那契数列迭代器"""
3.     def __init__(self, n):
4.         """
5.         :param n: int, 指明生成数列的前 n 个数
6.         """
7.         self.n = n
8.         # current 用来保存当前生成到数列中的第几个数了
9.         self.current = 0
10.        # num1 用来保存前一个数，初始值为数列中的第一个数 0
11.        self.num1 = 0
12.        # num2 用来保存前一个数，初始值为数列中的第二个数 1
13.        self.num2 = 1
14.
15.    def __next__(self):
16.        """被 next()函数调用来获取下一个数"""
17.        if self.current < self.n:
18.            num = self.num1
19.            self.num1, self.num2 = self.num2, self.num1+self.num2
20.            self.current += 1
21.            return num
22.        else:
23.            raise StopIteration
24.
25.    def __iter__(self):
26.        """迭代器的__iter__返回自身即可"""
27.        return self
```

迭代器实现：

```
1. def fib(n):
2.     ....:     current = 0
3.     ....:     num1, num2 = 0, 1
4.     ....:     while current < n:
5.     ....:         num = num1
6.     ....:         num1, num2 = num2, num1+num2
7.     ....:         current += 1
8.     ....:         yield num
9.     ....:     return 'done'
```

## 7. 请解释下 TCP/IP 协议和 HTTP 协议？

HTTP 协议：

HTTP 协议即超文本传送协议(Hypertext Transfer Protocol)，是 Web 联网的基础，也是手机联网常用的协议之一，HTTP 协议是建立在 TCP 协议之上的一种应用。HTTP 连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

TCP/IP 协议：

TCP/IP ( Transmission Control Protocol/Internet Protocol ) 协议是传输层协议，主要解决数据如何在网络中传输。HTTP 是应用层协议，主要解决如何包装数据。IP 协议对应于网络层。

详细了解可以看 第三章 Python 高级→八.网路编程

## 8. Python 里面如何实现 tuple 和 list 的转换？

list 转换成 tuple : t=tuple(l)。

tuple 转换成 list : l=list(t)。

## 9. 请写出以下 Linux 的 SHELL 命令？

显示所有文件包括隐藏文件	ls -a
切换到当前目录下的 dir 目录	cd dir
删除某一个文件	rm test
创建一个空文件	touch test
切换到 xiaoming 用户	su xiaoming
设置系统时间为 20:30:30	date -s 20:30:30

## 三十二、上海行知教育 Python 程序员笔试题 ( 2018-5-9 lyf )

### 1. Python 如何实现单例模式？请写出两种实现方法？

在 Python 中，我们可以用多种方法来实现单例模式：

1. 使用模块；
2. 使用\_\_new\_\_；
3. 使用装饰器；
4. 使用元类（metaclass）。

1) 使用模块：其实，Python 的模块就是天然的单例模式，因为模块在第一次导入时，会生成.pyc 文件，当第二次导入时，就会直接加载.pyc 文件，而不会再次执行模块代码。因此我们只需把相关的函数和数据定义在一个模块中，就可以获得一个单例对象了。

```
1. # mysingle.py
2. class MySingle:
3.     def foo(self):
4.         pass
5.
6. singleton = MySingle()
7.
8. 将上面的代码保存在文件 mysingle.py 中，然后这样使用：
9. from mysingle import singleton
10. singleton.foo()
```

2) 使用\_\_new\_\_：为了使类只能出现一个实例，我们可以使用\_\_new\_\_来控制实例的创建过程，

```
1. class Singleton(object):
2.     def __new__(cls):
3.         # 关键在于这，每一次实例化的时候，我们都只会返回这同一个 instance 对象
4.         if not hasattr(cls, 'instance'):
5.             cls.instance = super(Singleton, cls).__new__(cls)
6.         return cls.instance
7.
8. obj1 = Singleton()
9. obj2 = Singleton()
10.
11. obj1.attr1 = 'value1'
12. print obj1.attr1, obj2.attr1
13. print obj1 is obj2
14.
```

```
15. 输出结果 :  
16. value1 value1
```

3) 使用装饰器：装饰器可以动态的修改一个类或函数的功能。这里，我们也可以使用装饰器来装饰某个类，使其只能生成一个实例

```
1. def singleton(cls):  
2.     instances = {}  
3.     def getinstance(*args,**kwargs):  
4.         if cls not in instances:  
5.             instances[cls] = cls(*args,**kwargs)  
6.         return instances[cls]  
7.     return getinstance  
8.  
9. @singleton  
10. class MyClass:  
11.     a = 1  
12.  
13. c1 = MyClass()  
14. c2 = MyClass()  
15. print(c1 == c2) # True  
16.  
17.  
18. 在上面，我们定义了一个装饰器 singleton，它返回了一个内部函数 getinstance，  
19. 该函数会判断某个类是否在字典 instances 中，如果不存在，则会将 cls 作为 key，cls(*args, **kw) 作为 value 存到 instances 中，  
20. 否则，直接返回 instances[cls]。
```

4) 使用 metaclass (元类)：元类可以控制类的创建过程，它主要做三件事：

- 拦截类的创建
- 修改类的定义
- 返回修改后的类

```
1. class Singleton2(type):  
2.     def __init__(self, *args, **kwargs):  
3.         self.__instance = None  
4.         super(Singleton2,self).__init__(*args, **kwargs)  
5.  
6.     def __call__(self, *args, **kwargs):  
7.         if self.__instance is None:  
8.             self.__instance = super(Singleton2,self).__call__(*args, **kwargs)  
9.         return self.__instance
```

```
10.  
11.  
12. class Foo(object):  
13.     __metaclass__ = Singleton2 #在代码执行到这里的时候，元类中的__new__方法和__init__方法其实已经被执行了，而  
不是在 Foo 实例化的时候执行。且仅会执行一次。  
14.  
15.  
16. foo1 = Foo()  
17. foo2 = Foo()  
18. print (Foo.__dict__) #_Singleton__instance': <__main__.Foo object at 0x100c52f10> 存在一个私有属性来保存属性，  
而不会污染 Foo 类（其实还是会污染，只是无法直接通过__instance 属性访问）  
19. print (foo1 is foo2) # True
```

## 2. 什么是 lambda 函数？请举例说明？

匿名函数 lambda：是指一类无需定义标识符（函数名）的函数或子程序。lambda 函数可以接收任意多个参数（包括可选参数）并且返回单个表达式的值。

### 例 1:传入多个参数的 lambda 函数

```
1. def sum(x,y):  
2. return x+y
```

用 lambda 来实现：

```
1. p = lambda x,y:x+y  
2. print(4,6)
```

### 例 2：传入一个参数的 lambda 函数

```
1. a=lambda x:x*x  
2. print(a(3)) -----》注意：这里直接 a(3)可以执行，但没有输出的，前面的 print 不能少
```

### 例 3：多个参数的 lambda 形式：

```
1. a = lambda x,y,z:(x+8)*y-z  
2. print(a(5,6,8))
```

### 3. 如何反序地迭代一个序列？

在列表中，如果我们要将列表反向迭代通常使用 `reverse()`。但这个方法有个缺陷就是会改变列表。因此，我们推荐使用 `reversed()`，它会返回一个迭代器。这里，我们可以实现 `__reversed__()` 解决反向迭代问题。

```
1. class FloatRange:
2.
3.     def __init__(self, start, end, step):
4.         self.start = start
5.         self.end = end
6.         self.step = step
7.
8.     # 正向迭代
9.     def __iter__(self):
10.
11.         t = self.start
12.         while round(t, 2) <= round(self.end, 2):
13.             yield t
14.             t += self.step
15.
16.     # 反向迭代
17.     def __reversed__(self):
18.
19.         t = self.end
20.         while round(t, 2) >= round(self.start, 2):
21.             yield t
22.             t -= self.step
23.
24. if __name__ == "__main__":
25.
26.     for x in FloatRange(3.0, 4.0, 0.2):
27.         print x
28.
29.     print ""
30.
31.     for x in reversed(FloatRange(3.0, 4.0, 0.2)):
32.         print x
```

## 4. Python 如何生成随机数？

1. Python 中，获取随机数的方法大致有如下：
2. `import random` # 导入 random
- 3.
4. # python 中利用 random 获取一个 0 到 1 的随机浮点数
5. `a = random.random()`
6. `print a` #打印结果
- 7.
8. # python 中利用 random 获取一定范围内的（10 到 20）随机浮点数
9. `b = random.uniform(10, 20)`
10. `print b`
- 11.
12. # python 中利用 random 获取一定范围内（10 到 20）的随机整数
13. `c = random.randint(10,20)`
14. `print c`
- 15.
16. # python 中利用 random 从列表集合中获取一个随机值
17. `list = [5, 'hello', 9, 'xiong_it',3,"Python"]`
18. `d = random.choice(list)`
19. `print d`