python面试题

第一章:python基础

数据类型:

1 字典:

1.1 现有字典 dict={‘a’:24，‘g’:52，‘i’:12，‘k’:33}请按字典中的 value 值进行排序?

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

1.2 说一下字典和json的区别?

```
1. 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?

```
1.  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?

```
1. 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?

```
1. dict = {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?

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

2 字符串:

2.1 请反转字符串“aStr”?

```
1. . print(‘aStr’[::-1])
```

2.2 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

```
1. str1 = "k:1|k1:2|k2:3|k3:4"
2. def str2dict(str1):
3.  dict1 = {}
4.  for iterms in str1.split('|'):
5.      key, value = iterms.split(':')
6.      dict1[key] = value
7.  return dict1
```

2.3 请按alist中元素的age由大到小排序

alist [{‘name’:’a’，’age’:20}，{‘name’:’b’，’age’:30}，{‘name’:’c’，’age’:25}]

```
1.def sort_by_age(list1):
2.  return sorted(alist, key=lambda x:x['age'], reverse=True)
```

3 列表

3.1 下面代码的输出结果将是什么?

list = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
print(list[10:])

```
1. 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
2. 来获取某个列表的成员。
3. 例如，尝试获取list[10]和之后的成员，会导致IndexError。
```

3.2 写一个列表生成式，产生一个公差为11的等差数列

```
1. 1print([x*11 for x in range(10)])
3``2`456.
```

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？

```
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？

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)
```

如果想要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

```
1. A、B 中相同元素：print(set(A)&set(B))
2. 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？

```
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]
6. 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?

```
def extendlist(val, list=[]):
 list.append(val)
 return list
list1 = extendlist(10)
list2 = extendlist(123, []
list3 = extendlist('a')
print("list1 = %s" %list1)
print("list2 = %s" %list2)
print("list3 = %s" %list3)
输出结果:
list1 = [10, 'a']
list2 = [123]
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

```
def f1(lIn):
 l1 = sorted(lIn)
 l2 = [i for i in l1 if i<0.5]
 return [i*i for i in l2]
def f2(lIn):
 l1 = [i for i in l1 if i<0.5]
 l2 = sorted(l1)
 return [i*i for i in l2]
def f3(lIn):
    l1 = [i*i for i in lIn]
    l2 = sorted(l1)
    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。

```
import random
import cProfile
lIn = [random.random() for i in range(100000)]
cProfile.run('f1(lIn)')
cProfile.run('f2(lIn)')
cProfile.run('f3(lIn)')
```

3.9 获取1~100被7整除的偶数?

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

4. 元组

```
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中的书写方式为{}，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
sysmmetric_difference(对称差集)等数学运算.
```

快速去除列表中的重复元素

```
In [4]: a = [11,22,33,33,44,22,55]
```

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

交集：共有的部分

```
1. In [7]: a = {71,72,73,74,75}
2. In [8]: b = {72,74,75,76,77}
3. In [9]: a&b
4. Out[9]: {72, 74, 75}
```

并集：总共的部分

```
1. In [11]: a = {21,22,23,24,25}
2. In [12]: b = {22,24,25,26,27}
3. In [13]: a | b
4. Out[13]: {21, 22, 23, 24, 25, 26, 27}
```

差集：另一个集合中没有的部分

```
1. In [15]: a = {51,52,53,54,55}
2. In [16]: b = {52,54,55,56,57}
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 = {91,92,93,94,95}
2. In [20]: b = {92,94,95,96,97}
3. In [21]: a ^ b
4. Out[21]: {11, 33, 66, 77}
```

6.综合

1.通过代码实现如下转换

二进制转换成十进制：v = “0b1111011”
十进制转换成二进制：v = 18
八进制转换成十进制：v = “011”
十进制转换成八进制：v = 30
十六进制转换成十进制：v = “0x12”
十进制转换成十六进制：v = 87

```
print(int(v1,2))
print(int(v2,16))
print(int(v3,8))
print(bin(a1))
print(oct(a2))
print(hex(a3))
```

2. 求结果：

v1 = 1 or 3
v2 = 1 and 3
v3 = 0 and 2 and 1
v4 = 0 and 2 or 1
v5 = 0 and 2 or 1 or 4
v6 = 0 or Flase and 1

```
v1 = 1 or 3
v2 = 1 and 3
v3 = 0 and 2 and 1
v4 = 0 and 2 or 1
v5 = 0 and 2 or 1 or 4
v6 = 0 or False and 1
>>>1 3 0 1 1 False
```

3. python2与python3的区别

```
# py2
>>> print("hello", "world")
('hello', 'world')
# py3
>>> print("hello", "world")
hello world

py2：input_raw()
py3：input()

1/2的结果
py2：返回0
py3：返回0.5

py2：默认编码ascii
py3：默认编码utf-8

字符串
py2：unicode类型表示字符串序列，str类型表示字节序列
py3:：str类型表示字符串序列，byte类型表示字节序列
```

4. 用一行代码实现数值交换

a = 1
b = 2

```
a,b=(2,1)
```

5. xrange和range的区别?

```
xrange 用法与 range 完全相同，所不同的是生成的不是一个list对象，而是一个生成器。
注意：现在的python3中将以前的range取消了，而将xrange重新命名成了range！所以我们现在看到的range其实本质还是xrange~。
```

6. lambda表达式格式以及应用场景?

```
匿名就是没有名字
def func(x,y,z=1):
    return x+y+z

匿名
lambda x,y,z=1:x+y+z #与函数有相同的作用域，但是匿名意味着引用计数为0，使用一次就释放，除非让其有名字
func=lambda x,y,z=1:x+y+z
func(1,2,3)
#让其有名字就没有意义

与内置函数配合一起使用
```

8. *arg和**kwarg作用

```
*args用来接收溢出的位置参数，将接收的参数组织成元祖
**kwargs用来接收溢出的关键字参数，将接受的参数组织成字典
```

9. 求结果

v = dict.fromkeys(['k1','k2'],[])
v['k1'].append(666)
print(v)
v['k1'] = 777
print(v)

```
v = dict.fromkeys(['k1','k2'],[])
v['k1'].append(666)
print(v) #{'k1': [666], 'k2': [666]}
v['k1'] = 777
```

```
print(v)#{'k1': 777, 'k2': [666]}
#第一次字典的两个k指向的是同一块内存地址，所以k1的内存地址追加666，
k2的值也同样会是666，
而当给k1赋值时，改变了k1指向的内存地址，所以这个时候，k2不会随之发生变化
```

10. 一行代码实现9*9乘法表

```
print("\n".join("\t".join(["%s*%s=%s" %(x,y,x*y) for y in range(1, x+1)]))
```

11. 比较 a = [1,2,3] 和 b = [(1),(2),(3)] 以及 b = [(1,),(2,),(3,)] 的区别?

```
a与b两者值相等，而c中列表的每个元素是一个个的元祖形式
a,b元素均为数字，b中括号内没加逗号，所以仍然是数字
```

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?

```
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?

```
1. if hasattr(Parent, 'x'):
2.  print(getattr(Parent, 'x'))
3.  setattr(Parent, 'x', 29)
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?

```
1. 内存管理机制：引用计数、垃圾回收、内存池。
```

引用计数:

```
1. 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
2. 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

```
1. 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
2. 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
3. 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，
4. 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

```
1. 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
2. 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
```

```
3. 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
4. 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？
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

1.0 谈谈你对闭包的理解?

```
闭包(closure)是函数式编程的重要的语法结构。闭包也是一种组织代码的结构，它同样提高了代码的可重复使用性。
当一个内嵌函数引用其外部作作用域的变量,我们就会得到一个闭包. 总结一下,创建一个闭包必须满足以下几点:
    必须有一个内嵌函数
    内嵌函数必须引用外部函数中的变量
    外部函数的返回值必须是内嵌函数
感觉闭包还是有难度的,几句话是说不明白的,还是查查相关资料.
重点是函数运行后并不会被撤销,就像16题的instance字典一样,当函数运行完后,instance并不被销毁,而是继续留在内存空间里.这个功能类似类里的类变量,
只不过迁移到了函数上.
闭包就像个空心球一样,你知道外面和里面,但你不知道中间是什么样.
```

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?

```
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？

```
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?

```
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?

```
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?

```
a = 10
b = 20
c = [a]
a = 15
答：10对于字符串、数字，传递是相应的值。
```

2. 内置函数

2.1 map函数和reduce函数?

```
①从参数方面来讲：
map()包含两个参数，第一个参数是一个函数，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()是将传人的函数作用在序列的第一个元素得到结果后，把这个结果继续与下一个元素作用
（累积计算）。
```

2.2递归函数停止的条件?

```
递归的终止条件一般定义在递归函数内部，在递归调用前要做一个条件判断，根据判断的结果选择
是继续调用自身，还是 return;返回终止递归。
终止的条件：
1.判断递归的次数是否达到某一限定值
2.判断运算的结果是否达到某个范围等，根据设计的目的来选择
```

2.3 回调函数，如何通信的?

```
回调函数是把函数的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?

```
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?

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

2.6 hasattr() getattr() setattr() 函数使用详解?

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。

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"
functiondemo = function_demo()
res = hasattr(functiondemo, 'name')  #判断对象是否有name 属性，True
res = hasattr(functiondemo, "run") #判断对象是否有run方法，True
res = hasattr(functiondemo, "age") #判断对象是否有age属性，Falsw
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()。

```
functiondemo = function_demo()
getattr(functiondemo, 'name') #获取name属性，存在就打印出来--- demo
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__.functi
on_demo object at 0x10244f320>>
getattr(functiondemo, "age") #获取不存在的属性，报错如下：
Traceback (most recent call last):
  File "/Users/liuhuiling/Desktop/MT_code/OpAPIDemo/conf/OPCommUtil.py", line 39, in <module>
    res = getattr(functiondemo, "age")
AttributeError: 'function_demo' object has no attribute 'age'
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

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

综合使用:

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"
functiondemo = function_demo()
res = hasattr(functiondemo, 'addr') # 先判断是否存在if res:
    addr = getattr(functiondemo, 'addr')
    print(addr)else:
    addr = getattr(functiondemo, 'addr', setattr(functiondemo, 'addr', '北京首都'))
     #addr = getattr(functiondemo, 'addr', '美国纽约')
     print(addr)
```

reduce(lambda x,y: x*y, range(1,n+1)) 注意：Python3中取消了该函数。

Lambda

3.1什么是lambda函数? 有什么好处?

```
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。

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

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

另外一个解决方案就是创造一个闭包，利用默认函数立即绑定。

```
def  multipliers():
    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？

```
lambda 函数是匿名函数；使用 lambda 函数能创建小型匿名函数。这种函数得名于省略了用 def
声明函数的标准步骤；
f = lambda x，y:x+y
print(f(2017，2018))
```

四．设计模式

1. 单例

1.1请手写一个单例

```
class A(object):
    __instance = None
    def __new__(cls,  *args,  **kwargs):
        if  cls.__instance is None:
        cls.__instance = object.__new__(cls)
        return cls.__instance
        else:
        return cls.__instance
```

1.2单例模式的应用场景有哪些?

单例模式应用的场景一般发现在以下条件下：

```
（1）资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如日志文件，应用配置。
（2）控制资源的情况下，方便资源之间的互相通信。如线程池等。 1.网站的计数器 2.应用配置 3.多线程池 4.
数据库配置，数据库连接池 5.应用程序的日志应用....
```

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?

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

```
import time
```

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

@timeit
def foo():
    print 'in foo()'foo()
```

3.2解释一下什么是闭包?

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

3.3函数装饰器有什么作用?

```
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发并继续使用。
```

4. 生成器

4.1生成器、迭代器的区别?

```
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
发生器终结时,还会自动抛出 StopIteration 异常。
```

4.2 X是什么类型?

```
X = (for  i  in  ramg(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?

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

4.4Python中yield的用法?

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。

```
def createGenerator():
    mylist = range(3)
    for i in mylist:
    yield i*i
mygenerator = createGenerator() # create a generator
print(mygenerator) # mygenerator is an object!
<generator object createGenerator at 0xb7555c34>
for i in mygenerator:
    print(i)
# 输出结果: 0 1 4
```

5.试题
1 一行代码实现1–100之和

```
sum(range(0, 101))
```

2 如何在一个函数内部修改全局变量
利用global 修改全局变量

```
a = 20
def foo():
    global a
    a = 47
foo()
print(a)  # 4
```

3 列出5个python标准库
os：提供了不少与操作系统相关联的函数
sys: 通常用于命令行参数
re: 正则匹配
math: 数学运算
datetime:处理日期时间

4 字典如何删除键和合并两个字典
del和update方法

```
dic = {'name': 'kermit', 'age': 18}
del dic['name']  # 删除键
print(dic)  # {'age': 18}

dic2 = {'name': 'jack'}
dic.update(dic2)
print(dic)  # {'age': 18, 'name': 'jack'}
```

5、谈下python的GIL

```
GIL 是python的全局解释器锁，同一进程中假如有多个线程运行，一个线程在运行python程序的时候会霸占python解释器（加了一把锁即GIL），
使该进程内的其他线程无法运行，等该线程运行完后其他线程才能运行。如果线程运行过程中遇到耗时操作，则解释器锁解开，使其他线程运行。
所以在多线程中，线程的运行仍是有先后顺序的，并不是同时进行。
多进程中因为每个进程都能被系统分配资源，相当于每个进程有了一个python解释器，所以多进程可以实现多个进程的同时运行，缺点是进程系统
资源开销大
```

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?

```
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，也没有开辟新的出地址，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?

```
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

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?

将对象私有化，通过共有方法提供一个读取数据的接口。

```
class person:
    def __init__(self,x):
        self.__age = 10;
    def age(self):
        return self.__age;
t = person(22)
# t.__age = 100
print(t.age())
```

最好的方法

```
class MyCls(object):
    __weight = 50
@property  #以访问属性的方式来访问weight方法
def weight(self):
    return self.__weight
if __name__ == '__main__':
    obj = MyCls()
    print(obj.weight)
    obj.weight = 12

# 报错信息
Traceback (most recent call last):
50
File "C:/PythonTest/test.py", line 11, in <module>
obj.weight = 12
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?

```
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?

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

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

```
<.*>是贪婪匹配，会从第一个“<”开始匹配，直到最后一个“>”中间所有的字符都会匹配到，中间可能会包含
“<>”。
<.*?>是非贪婪匹配，从第一个“<”开始往后，遇到第一个“>”结束匹配，这中间的字符串都会匹配到，但是
不会有“<>”。
```

七．系统编程

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：

```
import osfrom multiprocessing import Process
import time
def pro_func(name, age, **kwargs):
    for i in range(5):
    print("子进程正在运行中,name=%s, age=%d, pid=%d" %(name, age, os.getpid()))
    print(kwargs)
    time.sleep(0.2)
if __name__ == '__main__':
    # 创建Process对象
    p = Process(target=pro_func, args=('小明',18), kwargs={'m': 20})
    # 启动进程
    p.start()

time.sleep(1) # 1秒钟之后，立刻结束子进程 17.
p.terminate()
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：

```
from multiprocessing import Process, Queueimport os, time, random # 写数据进程执行的代码:def write(q):
for value in ['A', 'B', 'C']:
    print('Put %s to queue...' % value)
    q.put(value)
    time.sleep(random.random())
```

读数据进程执行的代码:def read(q):

```
while True:
if not q.empty():
    value = q.get(True)
    print('Get %s from queue.' % value)
    time.sleep(random.random())
else:
    break
if __name__=='__main__':
# 父进程创建Queue，并传给各个子进程：
    q = Queue()
    pw = Process(target=write, args=(q,))
    pr = Process(target=read, args=(q,))
    # 启动子进程pw，写入：
    pw.start()

    # 等待pw结束：
    pw.join()

    # 启动子进程pr，读取：
    pr.start()
    pr.join()
    # pr进程里是死循环，无法等待其结束，只能强行终止：
    print('')
    print('所有数据都写入并且读完') 进程池Pool
```

```
from multiprocessing import Poolimport os, time, random
def worker(msg):
    t_start = time.time()
    print("%s开始执行,进程号为%d" % (msg,os.getpid()))
    # random.random()随机生成0~1之间的浮点数 7.      time.sleep(random.random()*2)
    t_stop = time.time()
    print(msg,"执行完毕，耗时%0.2f" % (t_stop-t_start))
    po = Pool(3)  # 定义一个进程池，最大进程数3
    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,))
    # 每次循环将会用空闲出来的子进程去调用目标     po.apply_async(worker,(i,))
    print("----start----")
    po.close()  # 关闭进程池，关闭后po不再接收新的请求 po.join()  # 等待po中所有子进程执行完成，必须放在close 语句之后 print("-----e
nd-----")
```

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.
from multiprocessing import Manager,Poolimport os,time,random
def reader(q):
    print("reader启动(%s),父进程为(%s)" % (os.getpid(), os.getppid()))
    for i in range(q.qsize()):
    print("reader从Queue获取到消息：%s" % q.get(True))
def writer(q):
    print("writer启动(%s),父进程为(%s)" % (os.getpid(), os.getppid()))
    for i in "itcast":
    q.put(i)
if __name__=="__main__":
    print("(%s) start" % os.getpid())
    q = Manager().Queue()  # 使用Manager中的Queue
```

```
    po = Pool()
    po.apply_async(writer, (q,))
    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 po.apply_async(reader, (q,))
    po.close()
    po.join()
    print("(%s) End" % os.getpid())
```

谈谈你对多进程，多线程，以及协程的理解，项目是否用?

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。
```

什么是多线程竞争?

```
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。
```

解释一下什么是锁，有哪几种锁?

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

什么是死锁呢?

```
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。
```

什么是线程安全，什么是互斥锁?

```
每个对象都对应于一个可称为" 互斥锁"的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
同一个进程中的多线程之间是共享系统资源的，多个线程同时对一个对象进行操作，一个线程操作尚未结束，
另一个线程已经对其进行操作，导致最终结果出现错误，此时需要对被操作对象添加互斥锁，保证每个线程对该对象的操作都得到正确的结果。
```

说说下面几个概念：同步，异步，阻塞，非阻塞?

```
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。
```

什么是僵尸进程和孤儿进程?怎么避免僵尸进程?

孤儿进程：父进程退出，子进程还在运行的这些子进程都是孤儿进程，孤儿进程将被 init 进程(进程号为 1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：进程使用fork创建子进程，如果子进程退出，而父进程并没有调用 wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中的这些进程是僵尸进程。

避免僵尸进程的方法：
-fork两次用孙子进程去完成子进程的任务；
-用wait()函数使父进程阻塞；
-使用信号量，在 signal handler 中调用waitpid，这样父进程不用阻塞。

Python中的进程与线程的使用场景?

多进程适合在 CPU 密集型操作(cpu 操作指令比较多，如位数多的浮点运算)。
多线程适合在 IO 密集型操作(读写数据操作较多的，比如爬虫)。

线程是并发还是并行，进程是并发还是并行?

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

并行（parallel）和并发（concurrency）?

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

CPU运算量大的程序程序，使用并行会更好。
IO密集型和CPU密集型区别?
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

```
import socket
def main():
    # 1、创建udp套接字
    # socket.AF_INET   表示IPv4协议   AF_INET6 表示IPv6协议
    # socket.SOCK_DGRAM   数据报套接字，只要用于udp协议
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # 2、准备接收方的地址
    # 元组类型   ip是字符串类型    端口号是整型
    dest_addr = ('192.168.113.111', 8888)
    # 要发送的数据
    send_data = "我是要发送的数据"
    # 3、发送数据
    udp_socket.sendto(send_data.encode("utf-8"), dest_addr)
    # 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
    # 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)
    # 1024   表示本次接收的最大字节数
    recv_data, addr = udp_socket.recvfrom(1024)
    # 5、关闭套接字
    udp_socket.close()
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

```
import socket
def main():
    # 1、创建udp套接字
    # socket.AF_INET  表示IPv4协议  AF_INET6 表示IPv6协议
    # socket.SOCK_DGRAM  数据报套接字，只要用于udp协议
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # 2、绑定端口
    # 元组类型  ip一般不写 表示本机的任何的一个ip
    local_addr = ('', 7777)
    udp_socket.bind(local_addr)
    # 3、准备接收方的地址
    # 元组类型  ip是字符串类型   端口号是整型
    dest_addr = ('192.168.113.111', 8888)
    # 要发送的数据
    send_data = "我是要发送的数据"
    # 4、发送数据
    udp_socket.sendto(send_data.encode("utf-8"), dest_addr)
    # 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
    # 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)
    # 1024  表示本次接收的最大字节数
    recv_data, addr = udp_socket.recvfrom(1024)
    # 6、关闭套接字
    udp_socket.close()
if __name__ == '__main__':
    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

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

```
import socket
def main():
    # 1、创建tcp服务端的socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 2、绑定
    server_socket.bind(('', 8888))
    # 3、listen使套接字变为被动套接字
    server_socket.listen(128)
    # 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
    # client_socket用来为这个客户端服务
    # tcp_server_socket就可以省下来专门等待其他新客户端的链接
    client_socket, client_addr = server_socket.accept()
    # 5、接收客户端发来的消息
    recv_data = client_socket.recv(1024)
    print("接收到客户端%s的数据：%s" % (str(client_addr), recv_data.decode('gbk')))
    # 6、回复数据给客户端
    client_socket.send("收到消息".encode('gbk'))
    # 7、关闭套接字
    client_socket.close()
    server_socket.close()
if __name__ == '__main__':
    main()
```

4. 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？

```
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

```
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？

GET请求：

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

POST请求：

```
POST请求会把请求的数据放置在HTTP请求包的包体中。上面的item=bandsaw就
是实际的传输数据。
因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。
传输数据的大小：
- 在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对
于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会
```

```
受到URL长度的限制。
- 对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST
提交数据大小进行限制，Apache、IIS都有各自的配置。
安全性：
- POST的安全性比GET的高。这里的安全是指真正的安全，而不同于上面GET提到的安全方法
中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过GET请求，
用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的
历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据
还可能会造成Cross-site request frogery攻击。
```

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?

```
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?

```
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）。
```

10. 请简单说一下三次握手和四次挥手?

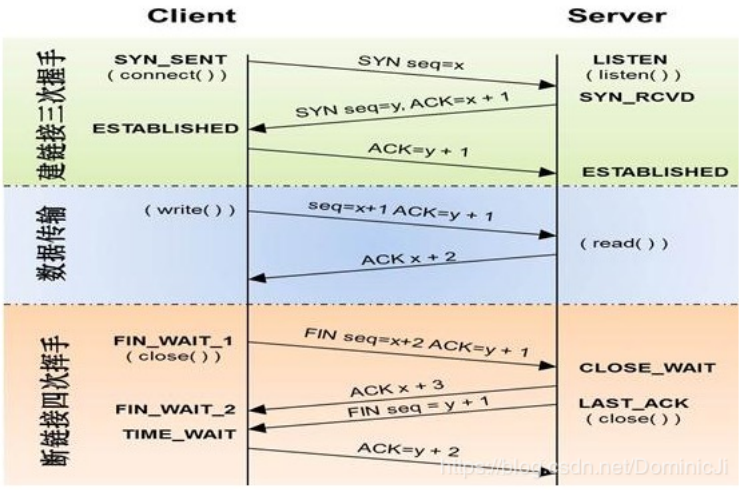
三次握手过程：

```
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?

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o

12. 为什么客户端在TIME-WAIT状态必须等待2MSL的时间?

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?

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?

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?

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？

```
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?

```
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?

```
IP： 网络层 TCP/UDP： 传输层 HTTP、RTSP、FTP： 应用层协议
```

19. url的形式?

```
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:8000/file/part01/1.2.html
```

第四章 前端

一．Html

head内常用标签:
https://www.cnblogs.com/Dominic-Ji/p/9085037.html
body内常用标签:
https://www.cnblogs.com/Dominic-Ji/p/9085099.html

二．Css

CSS选择器:
https://www.cnblogs.com/Dominic-Ji/p/9091130.html
属性相关:
https://www.cnblogs.com/Dominic-Ji/p/9100443.html

1. 什么是CSS初始化？有什么好处?

```
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?

```
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

参考博客:
https://www.cnblogs.com/Dominic-Ji/p/9111021.html

```
1. AJAX试什么？如何使用AJAX?
ajax(异步的javascript 和xml) 能够刷新局部网页数据而不是重新加载整个网页。
第一步，创建xmlhttprequest对象，var xmlhttp =new XMLHttpRequest ();XMLHttpRequest
对象用来和服务器交换数据。
第二步，使用xmlhttprequest对象的open()和send()方法发送资源请求给服务器。
第三步，使用xmlhttprequest对象的responseText或responseXML属性获得服务器的响应。
第四步，onreadystatechange函数，当发送请求到服务器，我们想要服务器响应执行一些功能就
需要使用onreadystatechange函数，每次xmlhttprequest对象的readyState发生改变都会触发
onreadystatechange函数。
```

第五章 Web

一．Flask

参考博客(写的很好，很全，很牛逼的~):https://www.cnblogs.com/Dominic-Ji/p/9505608.html

1. Flask 中正则 URL 的实现?

app.route('')中 URL 显式支持 string、int、float、path uuid any 6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?

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器交互的数据。
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?

```
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*用 Binds 操作多个数据库*。
SQLALCHEMY_ECHO	如果设置为Ture， 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get_debug_queries()。
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。

4. 常用的SQLAlchemy查询过滤器?

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

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')
```

字段对象 说明
FieldList 一组指定类型的字段
WTForms常用验证函数
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。

6. Flask项目中如何实现 session 信息的写入?

Flask中有三个 session：
第一个：数据库中的 session，例如:db.session.add()
第二个：在 flask_session 扩展中的 session，使用：from flask_session importSession，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。

7. 项目接口实现后路由访问不到怎么办?

1.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。
2.断点调试

8. Flask中url_for函数?

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('/')
```

```
def index():
    login_url = url_for('login')
    return redirect(login_url)
    return u'这是首页'

@app.route('/login/')
def login():
    return  u'这是登陆页面'

@app.route('/question/<is_login>/')
def question(is_login):
    if is_login == '1':
        return  u'这是发布问答的页面'
    else:
        return  redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
```

9. Flask中请求钩子的理解和应用?

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

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

10. 一个变量后写多个过滤器是如何执行的?

{{ expression | filter1 | filter2 | … }} 即表达式(expression)使用filter1 过滤后再将filter1的结果去使用 filter2 过滤。

11. 如何把整个数据库导出来，再导入指定数据库中?

导出：
mysqldump [-h 主机] -u 用户名 -p 数据库名 > 导出的数据库名.sql
导入指定的数据库中:
第一种方法：
mysqldump [-h 主机] -u 用户名 -p 数据库名 < 导出的数据库名.sql
第二种方法：
先创建好数据库，因为导出的文件里没有创建数据库的语句，如果数据库已经建好，则不用再创建。
create database example charset=utf8;（数据库名可以不一样）
切换数据库：
use example;
导入指定 sql 文件：
mysql>source /path/example.sql;

12. Flask和Django路由映射的区别?

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。

13. 跨站请求伪造和跨站请求保护的实现?

图中Browse是浏览器，WebServerA是受信任网站/被攻击网站A，WebServerB是恶意网站/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。

14. Flask(__name__)中的__name__可以传入哪些值?

可以传入的参数：

1，字符串：‘hello’，

但是‘abc’,不行，因为abc是python内置的模块

2，__name__，约定俗成

不可以插入的参数

1，python内置的模块，re,urllib,abc等

2，数字

二．Django

1. Django ORM查询中select_related和prefetch_related的区别？？

参考博客:https://www.cnblogs.com/Dominic-Ji/p/9213887.html

2. Django ORM详解?

参考博客:https://www.cnblogs.com/Dominic-Ji/p/9209341.html

3. Django创建项目的命令?

django-admin startproject 项目名称

python manage.py startapp 应用 app 名

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

项目文件夹下的组成部分：

manage.py 是项目运行的入口，指定配置文件路径。

与项目同名的目录，包含项目的配置文件。

_init.py 是一个空文件，作用是这个目录可以被当作包使用,也可以做一些初始化操作。

settings.py 是项目的整体配置文件。

urls.py 是项目的 URL 配置文件。

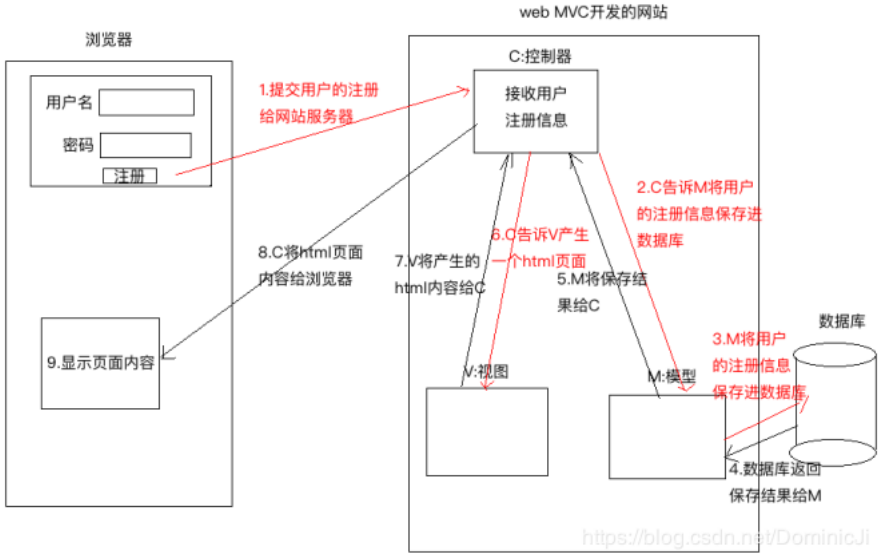
wsgi.py 是项目与 WSGI 兼容的 Web 服务器。

5. 对 MVC,MVT解读的理解?

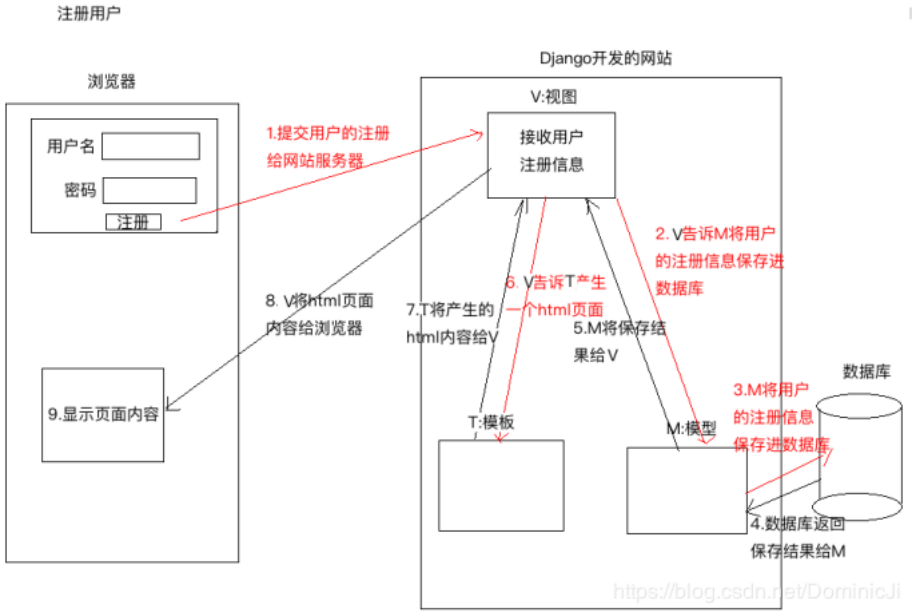
M：Model，模型，和数据库进行交互

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。

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

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(btitle__contains='传')
```

空查询 where 字段名 isnull

```
1. BookInfo.objects.filter(btitle__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。

7. django中间件的使用?

面试官问你Django中间件的时候，我们不应该只是局限于面试官的问题，而应做到举一反三。
面试之前准备一些白纸，在问到一些问题的时候应该用画图的形式展示出来
比如这里问到Django的中间件，我们应该给面试官画出Django的生命周期整体流程图，把中间件作为一部分的回答内容，
这样的好处在于，即展示了你对Django从前到后的流程都很熟悉又回答了面试官的问题，还顺带秀了一把其他技能，一举两得。
中间件介绍:作为Django的门户，一切请求都会先经过中间件才会到达Django后端，所以中间件可以用来做全局方面的一些功能
详细:给我们定义了五个方法

```
1.def process_request(request):
    pass
2.def process_view(request):
    pass
3.def process_template_response(request):
    pass
4.def process_exception(request):
    pass
5.def process_response(request):
    pass
```

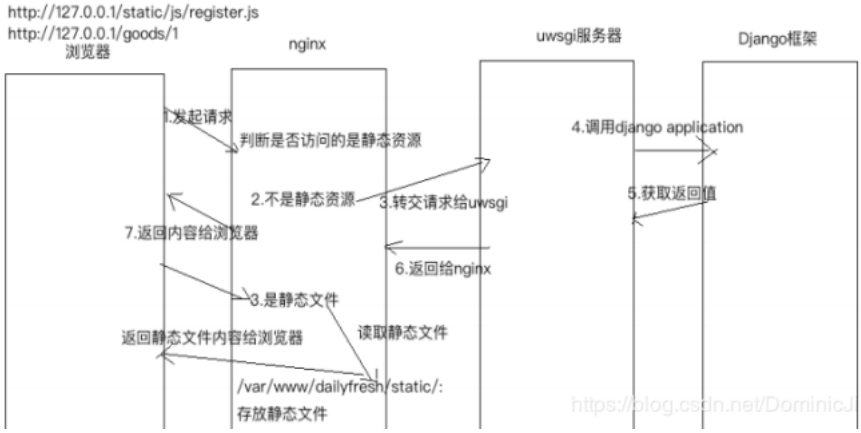
这些内容应该做到快速回答，不要”慢条斯理”的，搞IT的都很忙好吧，知识点一定要掌握牢固，脱口而出
一定要记住你是需要在有限的时间内将自己的看家本领不遗余力的倾囊而出

8. 谈一下你对uWSGI和 nginx的理解?

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。

9. 说说nginx和uWISG 服务器之间如何配合工作的?

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。



10. django开发中数据库做过什么优化?

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。

11. 验证码过期时间怎么设置?

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。

12. Python中三大框架各自的应用场景?

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。 得利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。

13. django如何提升性能（高并发）?

对一个后端开发程序员来说，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。

14. 什么是restful api，谈谈你的理解?

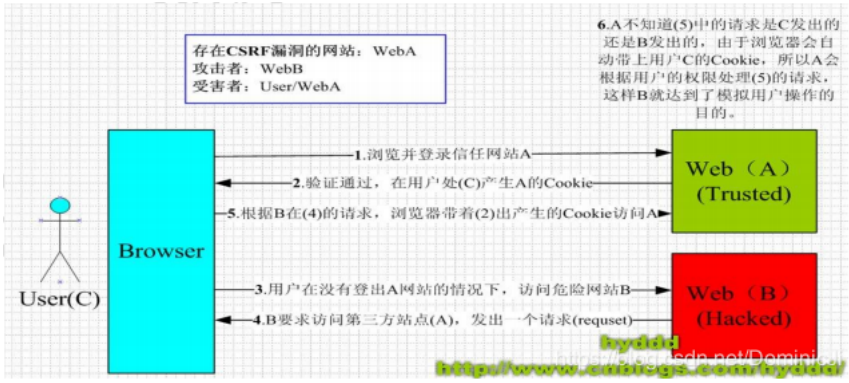
上来先给面试官扔出一手Django的restgramework源码(这一块知识课下一定要自己看着源码走三遍做到烂熟于心，看着面试官的眼睛快速自信的说出。这一手源码扔出来之后，面试已经成功一半)
REST:Representational State Transfer的缩写，翻译：“具象状态传输”。一般解释为“表现层状态转换”。
REST是设计风格而不是标准。是指客户端和服务器的交互形式。我们需要关注的重点是如何设计REST风格的网络接口。
REST的特点：
1.具象的。一般指表现层，要表现的对象就是资源。比如，客户端访问服务器，获取的数据就是资源。比如文字、图片、音视频等。
2.表现：资源的表现形式。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-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：客户端和服务器交互的过程。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.客户端和服务器之间，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。

14.1 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？



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：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?

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?

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?

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?

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?

设置Cookie

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

```
    response = HttpResponse("<h1>设置Cookie，请查看响应报文头</h1>")
    response.set_cookie('h1', 'hello django')
    return response
```

读取Cookie

```
def cookie_get(request):
    response = HttpResponse("读取Cookie，数据如下：<br>")
    if request.COOKIES.has_key('h1'):
        response.write('<h1>' + request.COOKIES['h1'] + '</h1>')
    return response
```

以键值对的格式写会话。

```
request.session['键']=值
```

根据键读取值。

```
request.session.get('键',默认值)
```

清除所有会话，在存储中删除值部分。

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

清除会话数据，在存储中删除会话的整条数据。

```
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。

```
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，那么会话在两周后过期。

```
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. 有用过Django REST framework 吗?

面试就喜欢面试官问这种问题，前面就已经说过了，这种问题一提出来，我们内心是高兴的一笔的，正好将我们学的Django restframework源码带面试官走一波，之后可以再补充一点Django的其他源码，比如ORM源码，settings源码，
admin源码…最后一定要记住，你要展示出你不仅阅读过源码还基于源码在自己的实际项目中参考借鉴过。如果把面试比作考试题满分100的话，这一题就是送分的30分大题！！！

20. Celery分布式任务队列?

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。
  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。

21. 简述Django下的（内建的）缓存机制?

Django提供6种缓存方式:
开发调试
内存
文件
数据库
Memcache缓存(python-memcached模块)
Memcache缓存(pylibmc模块)
除此之外还可使用redis缓存
由于Django是动态网站，所有每次请求均会去数据进行相应的操作，当程序访问量大时，耗时必然会更加明显，
最简单解决方式是使用：缓存，缓存将一个某个views的返回值保存至内存或者memcache中，5分钟内(默认配置)再有人来访问时，
则不再去执行view中的操作，而是直接从内存或者Redis中之前缓存的内容拿到，并返回。
这里可以向面试官介绍一下前面提到的Django中间件配合缓存协同工作的机制

22. 对cookie与session的了解？他们能单独用吗?

首先需要搞清楚的是session是存储在服务器上的，cookie是存储在客户端浏览器上的两者是相辅相成的用户首次访问服务器，服务器会为每个用户单独创建一个session对象(HttpSession)，
并为每个session分配唯一一个id(sessionId)，sessionId通过cookie保存到用户端，当用户再次访问服务器时，需将对应的sessionId携带给服务器，服务器通过这个唯一sessionId就可以找到用户对应的session对象，从而达到管理用户状态

23. Django里QuerySet的get和filter方法的区别?

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

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

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

1.安装django-cors-headers，之后在settings.py中配置

```
pip install django-cors-headers
```

```
INSTALLED_APPS = [
 ...
 'corsheaders',
 ...
 ]

MIDDLEWARE_CLASSES = (
 ...
 'corsheaders.middleware.CorsMiddleware',
 'django.middleware.common.CommonMiddleware', # 注意顺序
 ...
)
```

```
#跨域增加忽略
CORS_ALLOW_CREDENTIALS = True
CORS_ORIGIN_ALLOW_ALL = True
CORS_ORIGIN_WHITELIST = (
 '*'
)

CORS_ALLOW_METHODS = (
 'DELETE',
 'GET',
 'OPTIONS',
 'PATCH',
 'POST',
 'PUT',
 'VIEW',
)

CORS_ALLOW_HEADERS = (
 'XMLHttpRequest',
 'X_FILENAME',
 'accept-encoding',
 'authorization',
 'content-type',
 'dnt',
 'origin',
 'user-agent',
 'x-csrftoken',
 'x-requested-with',
 'Pragma',
)
```

2.使用JSONP

使用Ajax获取json数据时，存在跨域的限制。不过，在Web页面上调用js的script脚本文件时却不受跨域的影响，JSONP就是利用这个来实现跨域的传输。因此，我们需要将Ajax调用中的dataType从JSON改为JSONP（相应的API也需要支持JSONP）格式。

JSONP只能用于GET请求。

3.直接修改Django中的views.py文件

修改views.py中对应API的实现函数，允许其他域通过Ajax请求数据：

```
def myview(_request):
    response = HttpResponse(json.dumps({“key”: “value”, “key2”: “value”}))
    response[“Access-Control-Allow-Origin”] = “*”
    response[“Access-Control-Allow-Methods”] = “POST, GET, OPTIONS”
    response[“Access-Control-Max-Age”] = “1000”
    response[“Access-Control-Allow-Headers”] = “*”
return response
```

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

- 排序使用order_by()
- 降序需要在排序字段名前加-
- 查询字段大于某个值：使用filter(字段名_gt=值)
  更多骚操作:https://www.cnblogs.com/Dominic-Ji/p/9209341.html

27. 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
```

28.uWSGI与uwsgi区别

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。

29. 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。

30. 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
  (命令好记，实际操作就相对较难，平时应有意识地去锻炼使用git管理我们的代码仓库)

31. 什么是gitlab,github和gitlab的区别?

参考博客:https://blog.csdn.net/zhang_oracle/article/details/77317717

32. git中 .gitignore文件的作用?

参考博客:https://www.cnblogs.com/kevingrace/p/5690241.html

33. HttpRequest和HttpResponse是什么?干嘛用的?

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)

34. 什么是反向解析

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：\位置参数\

三．Tornado

1. Tornado的核是什么?

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

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

（考察对数据可的熟悉程度，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?

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

Re、json、jsonpath、BeautifulSoup、pyquery、lxml。

4. 爬取数据后使用哪个数据库存储数据的，为什么?

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MongDB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)Mongodb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 .monbgodb 有mapredurce功能，提供数据分析，redis 没有 ，这方面 mongodb优于redis 。

5. 写爬虫是用多进程好？还是多线程好？ 为什么?

一般情况下，在选择是使用多进程还是多线程时，主要考虑的业务到底是IO密集 型（多线程）还是计算密集型（多进程）。在爬虫中，请求的并发业务属于是网络 的IO类型业务，因此网络并发适宜使用多线程；但特殊需求下，比如使用 phantomjs或者chrome-headless来抓取的爬虫，应当是多进程的，因为每一个 phan/chro实例就是一个进程了，并发只能是多进程。。此外爬虫中还是数据处理 业务，如果数据处理业务是一个比较耗时的计算型操作，那么对数据处理部分应当 设为多进程，但更多可能会考虑将该部分数据处理操作和爬虫程序解耦，也就是先把数据抓取下来，事后单独运行另外的程序解析数据。

6. 常见的反爬虫和应对方法?

请求头、请求频率、IP地址、cookie等（持续更新请求头、cookie、用户cookie池、代理IP池、设置一定的延时）

7. 验证码的解决?

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（chromedirver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。
手动打码（有的验证码确实无解）

8. 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?

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。

9. cookie过期的处理问题?

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。

10. TTL，MSL，RTT?

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。

11. 代理 IP 里的“透明”“匿名”“高匿”分别是指?

透明代理的意思是客户端根本不需要知道有代理服务器的存在，但是它传送的仍然是真实的 IP。你要想隐藏的话，不要用这个。
普通匿名代理能隐藏客户机的真实 IP，但会改变我们的请求信息，服务器端有可能会认为我们使用了代理。不过使用此种代理时，虽然被访问的网站不能知道你的 ip 地址，但仍然可以知道你在使用代理，当然某些能够侦测 ip 的网页仍然可以查到你的 ip。 高匿名代理不改变客户机的请求，这样在服务器看来就像有个真正的客户浏览器在访问它，这时客户的真实 IP 是隐藏的，服务器端不会认为我们使用了代理。
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

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

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

13. 怎么获取加密的数据?

- Web端加密可尝试移动端（app） *解析加密，看能否破解
  *反爬手段层出不穷，js加密较多，只能具体问题具体分析
  分析前端js文件，找到加密解密数据的js代码，用Python代码实现或者利用js2py或pyexecjs等执行js代码

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

1. 5、6万条数据相对来说数据量比较小，线程数量不做强制要求(做除法得一个合理值即可)
2. 多线程使用代理，应保证不在同时一刻使用一个代理IP
3. 一般请求并发量主要考虑网站的反爬程度来定。

15. 怎么监控爬虫的状态

1. 使用python的STMP包将爬虫的状态信心发送到指定的邮箱
2. Scrapyd、pyspider
3. 引入日志
   集成日志处理平台来进行监控，如elk

二．Scrapy

1. 谈谈你对Scrapy的理解?

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。

2. 爬取下来的数据如何去重，说一下具体的算法依据

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掉就什么都没啦），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）。

3. Scrapy的优缺点?

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。

三．Scrapy-redis

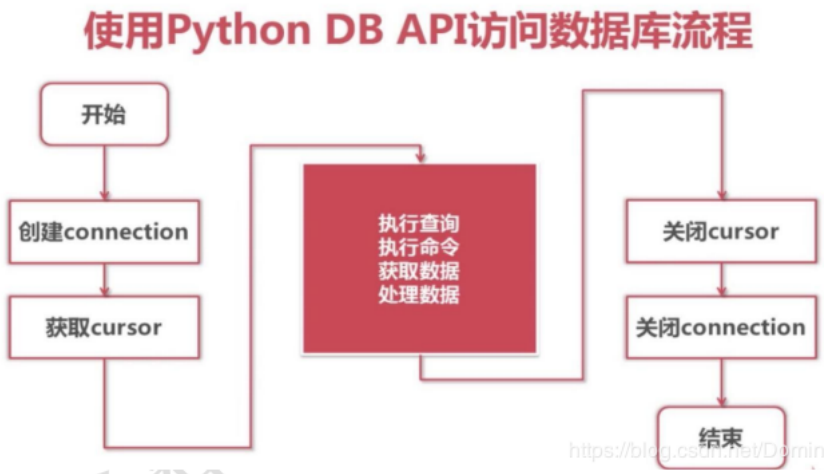
1. 什么是分布式存储?

传统定义：分布式存储系统是大量 PC 服务器通过 Internet 互联，对外提供一个整体的服务。
分布式存储系统具有以下的几个特性： 可扩展 ： 分布式存储系统可以扩展到几百台甚至几千台这样的一个集群规模，系统的 整体性能线性增长。
低成本 ： 分布式存储系统的自动容错、自动负载均衡的特性，允许分布式存储系统可 以构建在低成本的服务器上。另外，线性的扩展能力也使得增加、减少服务器的成本低， 实现分布式存储系统的自动运维。
高性能 ： 无论是针对单台服务器，还是针对整个分布式的存储集群，都要求分布式存 储系统具备高性能。
易用 ： 分布式存储系统需要对外提供方便易用的接口，另外，也需要具备完善的监 控、运维工具，并且可以方便的与其他的系统进行集成。 分布式存储系统的挑战主要在于数据和状态信息的持久化，要求在自动迁移、自动容 错和并发读写的过程中，保证数据的一致性。
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。

2. 你所知道的分布式爬虫方案有哪些?

三种分布式爬虫策略：
1.Slaver 端从 Master 端拿任务（Request/url/ID）进行数据抓取，在抓取数据的同时也 生成新任务，并将任务抛给 Master。Master 端只有一个 Redis 数据库，负责对 Slaver 提交 的任务进行去重、加入待爬队列。
优点： scrapy-redis 默认使用的就是这种策略，我们实现起来很简单，因为任务调度 等工作 scrapy-redis 都已经帮我们做好了，我们只需要继承 RedisSpider、指定 redis_key 就行了。
缺点： scrapy-redis 调度的任务是 Request 对象，里面信息量比较大（不仅包含 url， 还有 callback 函数、headers 等信息），导致的结果就是会降低爬虫速度、而且会占用 Redis 大量的存储空间。当然我们可以重写方法实现调度 url 或者用户 ID。
**2.Master 端跑一个程序去生成任务（Request/url/ID）。Master 端负责的是生产任务，
并把任务去重、加入到待爬队列。Slaver 只管从 Master 端拿任务去爬。**
优点： 将生成任务和抓取数据分开，分工明确，减少了 Master 和 Slaver 之间的数据 交流；Master 端生成任务还有一个好处就是：可以很方便地重写判重策略（当数据量大时 优化判重的性能和速度还是很重要的）。
缺点： 像 QQ 或者新浪微博这种网站，发送一个请求，返回的内容里面可能包含几十 个待爬的用户 ID，即几十个新爬虫任务。但有些网站一个请求只能得到一两个新任务，并 且返回的内容里也包含爬虫要抓取的目标信息，如果将生成任务和抓取任务分开反而会降 低爬虫抓取效率。毕竟带宽也是爬虫的一个瓶颈问题，我们要秉着发送尽量少的请求为原 则，同时也是为了减轻网站服务器的压力，要做一只有道德的 Crawler。所以，视情况而 定。
**3.Master 中只有一个集合，它只有查询的作用。Slaver 在遇到新任务时询问 Master 此
任务是否已爬，如果未爬则加入 Slaver 自己的待爬队列中，Master 把此任务记为已爬。它
和策略一比较像，但明显比策略一简单。策略一的简单是因为有 scrapy-redis 实现了
scheduler 中间件，它并不适用于非 scrapy 框架的爬虫。**
优点： 实现简单，非 scrapy 框架的爬虫也适用。Master 端压力比较小，Master 与 Slaver 的数据交流也不大。
缺点：“健壮性”不够，需要另外定时保存待爬队列以实现“断点续爬”功能。各 Slaver 的待爬任务不通用。 如果把 Slaver 比作工人，把 Master 比作工头。
策略一就是工人遇到新任务都上报给工头，需要干活的时候就去工头那里领任务；
策略二就是工头去找新任务，工人只管从工头那里领任务干活；
策略三就是工人遇到新任务时询问工头此任务是否有人做了，没有的话工人就将此任务加到自己的“行程表”。

第九章 数据库



一．Mysql

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
(2) JOIN
(3) ON
(4) WHERE
(5) GROUP BY
(6) WITH {CUBE | ROLLUP}
(7) HAVING
(8) SELECT
(9) DISTINCT
(9) ORDER BY
(10)
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

1. 常见SQL（必备）

详见武沛齐博客：https://www.cnblogs.com/wupeiqi/articles/5729934.html

2. 什么是事务，MySQL是如何支持事务的?

事务就是一段sql 语句的批处理，但是这个批处理是一个原子 ，不可分割，要么都执行，要么回滚（rollback）都不执行。
事务具体四大特性，也就是经常说的ACID ：
1.原子性（所有操作要么全部成功，要么全部失败回滚）
2.一致性（事务执行之前和执行之后都必须处于一致性状态。）
3.隔离性（数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离）
4.持久性（一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，
即使遭遇故障依然能够通过日志恢复最后一次更新在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务
MYSQL 事务处理主要有两种方法：
1、用 BEGIN, ROLLBACK, COMMIT来实现 BEGIN 开始一个事务 ROLLBACK 事务回滚 COMMIT 事务确认
2、直接用 SET 来改变 MySQL 的自动提交模式: SET AUTOCOMMIT=0 禁止自动提交 SET AUTOCOMMIT=1 开启自动提交

3. 说一下Mysql数据库存储的原理?

储存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. 数据库索引种类?

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度
MySQL索引的类型：

1. 普通索引：这是最基本的索引，它没有任何限制

2.唯一索引：索引列的值必须唯一，但允许有空值，如果是组合索引，则列值的组合必须唯一

3.全文索引：全文索引仅可用于 MyISAM 表，可以从CHAR、VARCHAR或TEXT列中作为CREATE TABLE语句的一部分被创建，或是随后使用ALTER TABLE 或CREATE INDEX被添加（切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法）

4. 单列索引、多列索引：多个单列索引与单个多列索引的查询效果不同，因为执行查询时，MySQL只能使用一个索引，会从多个索引中选择一个限制最为严格的索引。

5.组合索引（最左前缀）：简单的理解就是只从最左面的开始组合（实在单列索引的基础上进一步压榨索引效率的一种方式）

5. 索引在什么情况下遵循最左前缀的规则?

mysql在使用组合索引查询的时候需要遵循“最左前缀”规则

6. MySQL常见的函数?

聚合函数：

AVG(col)返回指定列的平均值
COUNT(col)返回指定列中非NULL值的个数
MIN(col)返回指定列的最小值
MAX(col)返回指定列的最大值
SUM(col)返回指定列的所有值之和
GROUP_CONCAT(col) 返回由属于一组的列值连接组合而成的结果

数学函数：

ABS(x) 返回x的绝对值
BIN(x) 返回x的二进制（OCT返回八进制，HEX返回十六进制）

7. 如何开启慢日志查询?

1 执行 SHOW VARIABLES LIKE “%slow%”，获知 mysql 是否开启慢查询 slow_query_log 慢查询开启状态 OFF 未开启 ON 为开启 slow_query_log_file 慢查询日志存放的位置（这个目录需要MySQL的运行帐号的可写权限，一般设置为MySQL的数据存放目录）
2 修改配置文件（ 放在[mysqld]下），重启 long_query_time 查询超过多少秒才记录
3 测试是否成功
4 慢查询日志文件的信息格式

8. 数据库导入导出命令（结构+数据）?

1.导出整个数据库
mysqldump -u用户名 -p密码 数据库名 > 导出的文件名
例如：C:\Users\jack> mysqldump -uroot -pmysql sva_rec > e:\sva_rec.sql
2.导出一个表，包括表结构和数据
mysqldump -u用户名 -p 密码 数据库名 表名> 导出的文件名
例如：C:\Users\jack> mysqldump -uroot -pmysql sva_rec date_rec_drv> e:\date_rec_drv.sql
3.导出一个数据库结构
例如：C:\Users\jack> mysqldump -uroot -pmysql -d sva_rec > e:\sva_rec.sql
4.导出一个表，只有表结构
mysqldump -u用户名 -p 密码 -d数据库名 表名> 导出的文件名
例如：C:\Users\jack> mysqldump -uroot -pmysql -d sva_rec date_rec_drv> e:\date_rec_drv.sql
5.导入数据库
常用source 命令
进入mysql数据库控制台，
如mysql -u root -p　　mysql>use 数据库
然后使用source命令，后面参数为脚本文件(如这里用到的.sql)
mysql>source d:wcnc_db.sql

9. 数据库怎么优化查询效率?

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。

10. Mysql集群的优缺点?

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

11. 你用的Mysql是哪个引擎，各引擎之间有什么区别?

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%’

12. 数据库的优化?

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

13. Mysql数据库如何分区、分表?

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。

14. Sql注入是如何产生的，如何防止?

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，避免直接暴漏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。

15. NoSQL和关系数据库的区别?

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。

16. 简述触发器、函数、视图、存储过程?

触发器：触发器是一个特殊的存储过程，它是MySQL在insert、update、delete的时候自动执行的代码块。

```
create trigger trigger_name

    after/before insert /update/delete on 表名

    for each row

    begin

    sql语句：（触发的语句一句或多句）

    end
```

函数：MySQL中提供了许多内置函数，还可以自定义函数（实现程序员需要sql逻辑处理）

```
自定义函数创建语法：

    创建：CREATE FUNCTION 函数名称(参数列表)

            RETURNS 返回值类型    函数体

    修改：ALTER FUNCTION 函数名称 [characteristic ...]

    删除：DROP FUNCTION [IF EXISTS] 函数名称

    调用：SELECT 函数名称(参数列表)
```

视图：视图是由查询结果形成的一张虚拟表，是表通过某种运算得到的一个投影

```
create view view_name as select 语句
```

存储过程：把一段代码封装起来，当要执行这一段代码的时候，
可以通过调用该存储过程来实现（经过第一次编译后再次调用不需要再次编译，比一个个执行sql语句效率高）

```
 create procedure 存储过程名(参数,参数,…)

    begin

    //代码

    end
```

17. 列举 创建索引但是无法命中索引的8种情况。

- 1、如果条件中有or，即使其中有条件带索引也不会使用(这也是为什么尽量少用or的原因)
- 2、对于多列索引，不是使用的第一部分(第一个)，则不会使用索引
- 3、like查询是以%开头
- 4、如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引
- 5、如果mysql估计使用全表扫描要比使用索引快,则不使用索引
- 6 对小表查询
- 7 提示不使用索引
- 8 统计数据不真实
- 9.单独引用复合索引里非第一位置的索引列.

18. 优化数据库？提高数据库的性能

1. 对语句的优化
   ①用程序中，保证在实现功能的基础上，尽量减少对数据库的访问次数；
   通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担；
   ②能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用SQL时，尽量把使用的索引放在选择的首列；算法的结构尽量简单；
   ③在查询时，不要过多地使用通配符如 SELECT * FROM T1 语句，要用到几列就选择几列如：
   SELECT COL1,COL2 FROM T1；

④在可能的情况下尽量限制尽量结果集行数如：SELECT TOP 300 COL1,COL2,COL3 FROM T1,因为某些情况下用户是不需要那么多的数据的。
⑤不要在应用中使用数据库游标，游标是非常有用的工具，但比使用常规的、面向集的SQL语句需要更大的开销；按照特定顺序提取数据的查找。

2. 避免使用不兼容的数据类型
   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。

19. 数据库负载均衡

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

1、 实现原理

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

2、 实现多据库数据同步

对于负载均衡，最重要的就是所有服务器的数据都是实时同步的。这是一个集群所必需的，因为，如果数不据实时、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。

20. 数据库三大范式?

什么是范式：简言之就是，数据库设计对数据的存储性能，还有开发人员对数据的操作都有莫大的关系。所以建立科学的，规范的的数据库是需要满足一些规范的来优化数据数据存储方式。在关系型数据库中这些规范就可以称为范式。

什么是三大范式：

第一范式：当关系模式R的所有属性都不能在分解为更基本的数据单位时，称R是满足第一范式的，简记为1NF。满足第一范式是关系模式规范化的最低要求，否则，将有很多基本操作在这样的关系模式中实现不了。

第二范式：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式，简记为2NF。

第三范式：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，简记为3NF.
注：关系实质上是一张二维表，其中每一行是一个元组，每一列是一个属性

21. 简述数据库设计中一对多和多对多的应用场景?

一对多：学生与班级—一个学生只能属于一个班级，一个班级可以有多个学生
多对多：学生与课程—一个学生可以选择多个课程，一个课程也可以被多个学生选择

22. 如何基于数据库实现商城商品计数器?

创建一个商城表—包含（id，商品名，每一个商品对应数量）

```
create table product
   (id primary key auto_increment,
     pname varchar(64),
      pcount int);
```

23. char和varchar的区别?

char：定长，char的存取数度相对快
varchar：不定长，存取速度相对慢
到底如何取舍可以根据一下几个方面考虑：
1、对于MyISAM表，尽量使用Char，对于那些经常需要修改而容易形成碎片的myisam和isam数据表就更是如此，它的缺点就是占用磁盘空间；
2、对于InnoDB表，因为它的数据行内部存储格式对固定长度的数据行和可变长度的数据行不加区分（所有数据行共用一个表头部分，这个标头部分存放着指向各有关数据列的指针），所以使用char类型不见得会比使用varchar类型好。事实上，因为char类型通常要比varchar类型占用更多的空间，所以从减少空间占用量和减少磁盘i/o的角度，使用varchar类型反而更有利；
3、存储很短的信息，比如门牌号码101，201……这样很短的信息应该用char，因为varchar还要占个byte用于存储信息长度，本来打算节约存储的现在得不偿失。
4、固定长度的。比如使用uuid作为主键，那用char应该更合适。因为他固定长度，varchar动态根据长度的特性就消失了，而且还要占个长度信息。
5、十分频繁改变的column。因为varchar每次存储都要有额外的计算，得到长度等工作，如果一个非常频繁改变的，那就要有很多的精力用于计算，而这些对于char来说是不需要的。

24. 在对name做了唯一索引前提下，简述以下区别：

```
select * from tb where name = ‘Oldboy’ -------------查找到tb表中所有name = ‘Oldboy’的数据

select * from tb where name = ‘Oldboy’ limit 1------查找到tb表中所有name = ‘Oldboy’的数据只取其中的第一条
```

二．MongoDB

参考博客

MongoDB:https://www.cnblogs.com/Dominic-Ji/p/9331076.html
MongoDB基本数据库操作:https://www.cnblogs.com/Dominic-Ji/p/9524079.html
Pymongo的使用: https://www.cnblogs.com/Dominic-Ji/p/9531899.html

1. MongoDB

MongoDB是一个面向文档的数据库系统。使用C++编写，不支持SQL，但有自己功能强大的查询语法。
MongoDB使用BSON作为数据存储和传输的格式。BSON是一种类似JSON的二进制序列化文档，支持嵌套对象和数组。
MongoDB很像MySQL，document对应MySQL的row，collection对应MySQL的table
应用场景：
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

2. MySQL与MongoDB本质之间最基本的差别是什么

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，所占空间会莫明其妙地增大，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。

3. 使用MongoDB的优点

```
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

三．Redis

三篇博客带你全方位了解熟悉Redis:

Redis简介:https://www.cnblogs.com/Dominic-Ji/p/9180006.html
Redis数据类型及常用命令:https://www.cnblogs.com/Dominic-Ji/p/9185078.html
Redis必备知识:https://www.cnblogs.com/Dominic-Ji/p/9206991.html
熟读这三篇博客之后，关于redis的相关问题，相信你都能形成初步的概念与合理的答案

1. Redis五大数据类型及对应使用场所。

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，可以用来做微博涨粉，点赞关注数变化。
2. 常用命令: set,get,decr,incr,mget等。
Hash
1. Hash 类型可以看成是一个key/value都是 String的Map容器。 通常用来存储对象数据类型
2. 常用命令：hget,hset,hgetall 等。
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片段。可用来做微信朋友圈按时间顺序加载
2. 常用命令：lpush,rpush,lpop,rpop,lrange 等
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。可用来做共同好友，共同关注等
2. 常用命令：sadd,spop,smembers,sunion 等。
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。可用来做各类排行榜应用
2. 常用命令：zadd,zrange,zrem,zcard等。

2. 怎样解决数据库高并发的问题?

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:
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中的sentinel(哨兵模式)的作用?

在哨兵模式下，如果主机宕机了，会在从机里面投票选出一个从机当主机，之后如果原来的主机又回来了，在较短的时间内还没有被哨兵模式监控到的时候，回来的主机就是自己一个人单独一套体系自己是光杆司令，但是一会儿功夫，哨兵模式监控到了这个重启的主机后，哨兵模式会告诉这个新来的主机，已经换老大了，你需要跟着新老大混，这个时候新来的就会自动变为从机依附于前面投票选出来的主机

4. Redis的并发竞争问题怎么解决?

方案一：可以使用独占锁的方式，类似操作系统的mutex机制，不过实现相对复杂，成本较高。
方案二：使用乐观锁的方式进行解决（成本较低，非阻塞，性能较高）
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. Redis的事务?

Redis 事务命令

下表列出了 redis 事务的相关命令：

序号	命令及描述
1	DISCARD 取消事务，放弃执行事务块内的所有命令。
2	EXEC 执行所有事务块内的命令。
3	MULTI 标记一个事务块的开始。
4	UNWATCH 取消 WATCH 命令对所有 key 的监视。
5	WATCH key [key ....] 监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。

Redis的事务具有以下特点:
事务之全体连坐
在创建事务的时候，如果事务内有一行代码在创建的时候就已经报错，那么整个事务都不会被执行
事务之冤头债主
在创建事务的时候，代码都没有报错，但是在执行阶段有几个语句无法操作，比如你的语句是让值加一，然而值是字符串类型，没法加一，这个时候所有不报错的语句都会执行成功，只有那些执行阶段报错的语句不会执行成功原因:redis是部分支持事务，不像传统的数据库对事务是强一致性的要求

7. Redis 的使用场景有哪些?

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

8. Redis 默认端口，默认过期时间，Value 最多可以容纳的数据 长度?

1. 默认端口：6379 (补充:大家应该知道常用数据库的默认端口都是多少Redis:6379,MySQL:3306，MongoDB:27017)
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。

9. Redis 有多少个库?

Redis自带有16个库，默认情况下是0库，可通过select选择相同的库，一般情况下都只使用0号库，其他库一般都用不到

第十一章 数据结构与算法

1. 算法的特征?

- 有穷性： 一个算法必须保证执行有限步骤之后结束；
- 确切性： 算法的每一步骤必须有确切的定义；
- 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；
- 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；
- 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. 快速排序的思想?

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（n2） ,最好的时间复杂度是(nlogn),空间复杂度为O(logn)

```
def quick_sort(alist, start, end):
    """快速排序"""
    # 递归的退出条件
    if start >= end:
        return
    # 设定起始元素为要寻找位置的基准元素
    mid = alist[start]
    # low为序列左边的由左向右移动的游标
    low = start
    # high为序列右边的由右向左移动的游标
    high = end
    while low < high:
        # 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
        while low < high and alist[high] >= mid:
            high -= 1
        # 将high指向的元素放到low的位置上
        alist[low] = alist[high]
        # 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
        while low < high and alist[low] < mid:
            low += 1
        # 将low指向的元素放到high的位置上
        alist[high] = alist[low]
    # 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
    # 将基准元素放到该位置
    alist[low] = mid
    # 对基准元素左边的子序列进行快速排序
    quick_sort(alist, start, low-1)
    # 对基准元素右边的子序列进行快速排序
    quick_sort(alist, low+1, end)
alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
quick_sort(alist, 0, len(alist)-1)
print(alist)
```

3. 如何判断单向链表中是否有环?

对于这个问题我们可以采用“快慢指针”的方法。就是有两个指针fast和slow，开始的时候两个指针都指向链表头head，然后在每一步操作中slow向前走一步即：slow = slow->next，而fast每一步向前两步即：fast =fast->next->next。
由于fast要比slow移动的快，如果有环，fast一定会先进入环，而slow后进入环。当两个指针都进入环之后，经过一定步的操作之后二者一定能够在环上相遇，并且此时slow还没有绕环一圈，也就是说一定是在slow走完第一圈之前相遇

![](https://img-blog.csdnimg.cn/20181108114708924.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L0RvbWluaWNkaQ==,size_16,color_FFFFFF,t
拓展:https://www.cnblogs.com/dancingrain/p/3405197.html

4. 基本的算法有哪些，怎么评价一个算法的好坏?

基本的算法有: 二分，冒泡，快排，插入，堆排序，希尔排序
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)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。

5. 斐波那契数列

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

```
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
```

6. 二叉树如何求两个叶节点的最近公共祖先?

二叉树是搜索二叉树： 1、原理：二叉搜索树是排序过的 ，位于左子树的结点都比父结点小，位 于右子树的结点都比父结点大，我们只需从根节点开始和两个输入的结点进行 比较，如果当前节点的值比两个结点的值都大，那么最低的公共祖先结点一定 在该结点的左子树中，下一步开遍历当前结点的左子树。如果当前节点的值比 两个结点的值都小，那么最低的公共祖先结点一定在该结点的右子树中，下一 步开遍历当前结点的右子树。这样从上到下找到第一个在两个输入结点的值之 间的结点。
2、实现代码：

```
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
```

7. 找出二叉树中最远结点的距离?

计算一个二叉树的最大距离有两个情况。
情况A: 路径经过左子树的最深节点，通过根节点，再到右子树的最深节点。
情况B: 路径不穿过根节点，而是左子树或右子树的最大距离路径，取其大者。 只需要计算这两个情况的路径距离，并取其大者，就是该二叉树的最大距离。

8. 写一个二叉树

```
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)
```

```
            print root.data
            inorder(root.right)

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

9. set 用 in 时间复杂度是多少，为什么?

O(1)，因为set是键值相同的一个数据结构，键做了hash处理。

10. 深度优先遍历和广度优先遍历的区别?

1） 二叉树的深度优先遍历的非递归的通用做法是采用栈，广度优先遍历的非递归的通用做法是采用队列。
2） 深度优先遍历：对每一个可能的分支路径深入到不能再深入为止，而且每个结点只能访问一次。
要特别注意的是，二叉树的深度优先遍历比较特殊，可以细分为先序遍历、中序遍历、后序遍历。具体说明如下：
先序遍历：对任一子树，先访问根，然后遍历其左子树，最后遍历其右子树。
中序遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树。
后序遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访问根。
广度优先遍历：又叫层次遍历，从上往下对每一层依次访问，在每一层中，从左往右（也可以从右往左）访问结点，访问完一层就进入下一层，直到没有结点可以访问为止。
3）深度优先搜素算法：不全部保留结点，占用空间少；有回溯操作(即有入栈、出栈操作)，运行速度慢。
广度优先搜索算法：保留全部结点，占用空间大； 无回溯操作(即无入栈、出栈操作)，运行速度快。 通常，深度优先搜索法不全部保留结点，扩展完的结点从数据库中弹出删去，这样，一般在数据库中存储的结点数就是深度值，因此它占用空间较少。所以，当搜索树的结点较多，用其它方法易产生内存溢出时，深度优先搜索不失为一种有效的求解方法。
广度优先搜索算法，一般需存储产生的所有结点，占用的存储空间要比深度优先搜索大得多，因此，程序设计中，必须考虑溢出和节省内存空间的问题。但广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索要快些。

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

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

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

class ListNode:
    def __init__(self, x):
   self.val=x
   self.next=None
def recurse(head, newhead): #递归，head 为原链表的头结点，newhead 为
反转后链表的头结点
   if head is None:
       return
   if head.next is None:
       newhead=head
   else :
       newhead=recurse(head.next, newhead)
       head.next.next=head
       head.next=None
    return newhead
```

12. 青蛙跳台阶问题

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3：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
```

13. 删除元素 Remove Element

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:
```

```
            m+=1
    for i in range(n+1):
        A.pop()
    return A
```

14. 用两个队列如何实现一个栈，用两个栈如何实现一个队列?

两个栈实现一个队列:
栈的特性是先进后出（FILO），队列的特性是先进先出（FIFO），在实现 delete时，将一个栈中的数据依次拿出来压入到另一个为空的栈，另一个栈中数据的顺序恰好是先压入栈 1 的元素此时在栈 2 的上面，为了实现效率的提升，在delete 时，判断栈 2 是否有数据，如果有的话，直接删除栈顶元素，在栈 2 为空时才将栈 1 的数据压入到栈 2 中，从而提高程序的运行效率，实现过程可以分为
下面几个步骤:
1、push 操作时，一直将数据压入到栈 2 中
2、delete 操作时，首先判断栈 2 是否为空，不为空的情况直接删除栈 2 栈顶元素，为空的话将栈 1 的数据压入到栈 2 中，再将栈 2 栈顶元素删除。
两个队列实现一个栈：
因为队列是先进先出，所以要拿到队列中最后压入的数据，只能每次将队列中数据 pop 到只剩一个，此时这个数据为最后压入队列的数据，在每次 pop 时，将数据压入到另一个队列中。每次执行 delete 操作时，循环往复。

15. 爬楼梯 Climbing Stairs

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int[3]。于是你就会发现，这其实就是一个裴波拉契数列问题。

```
class Solution:
    # @param n, an integer
    # @return an integer
    def climbStairs(self, n):
        if n <= 1:
            return 1
        arr = [1, 1, 0]      # look here, arr[0] = 1, arr[1] = 2
        for i in range(2, n + 1):
            arr[2] = arr[0] + arr[1]
            arr[0], arr[1] = arr[1], arr[2]
         return arr[2]
```

16. 落单的数Single Number

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。

```
class Solution:
    # @param A, a list of integer
    # @return an integer
    def singleNumber(self, A):
        len_A = len(A)
        if 0 == len_A:
            return 0
        elif 1 == len_A:
            return A[0]
         else:
             result = A[0]
             for i in range(1, len_A):
                 result ^= A[i]
         return result
```

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表中的一样的数据)?

```
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?

```
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?

```
1. lxml
```

5. Redis里面list内容的长度?

```
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?

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?

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?

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 orign():
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。

```
def new_func(func):
    def wrappedfun(*args,**kwargs):
        if args:
            counts = len(args)
            print('本系统包含 ',end='')
            for arg in args:
                print(arg,' ',end='')
            print('等',counts,'部分')
            if kwargs:
                 for k in kwargs:
                     v= kwargs[k]
                     print(k,'为: ',v)
             return func()
         else:
             if kwargs:
                 for kwarg in kwargs:
                     print(kwarg)
                     k,v = kwarg
                     print(k,'为: ',v)
             return func()
     return wrappedfun

@new_func
def orign():
     print('开始执行函数')

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?

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?

会打印出字符型的所有的内置方法。

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__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?

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?

```
new_str = old_str[::-1]
```

5. 交换变量a,b的值?

```
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?

```
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?

```
def index(list, key):
    if key < list[0]:
        position = 0
    elif key > list[-1]:
        position = len(list)
    else:
        for i in range(len(list)):
            if key>list[i] and list[i+1]>key:
                position = i+1
    return position
```

8. 正则表达式贪婪与非贪婪模式的区别?

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?

```
^[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?

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?

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，浏览器想该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。

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。

```
a= ["a","b",["1","3",["4","haha"]]]
for b in a :
  for c in b :
    for d in c :
      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?

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:

1. 最简单的方法：直接使用Timer类

```
import time
import os

while True:
    os.system("scrapy crawl News")
time.sleep(86400)  #每隔一天运行一次 24*60*60=86400s
```

2. 使用sched

```
import sched
#初始化sched模块的scheduler类
```

```
#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。
schedule = sched.scheduler ( time.time, time.sleep )

#被周期性调度触发的函数
def func():
    os.system("scrapy crawl News")
def perform1(inc):
    schedule.enter(inc,0,perform1,(inc,))
    func()    # 需要周期执行的函数
def mymain():
    schedule.enter(0,0,perform1,(86400,))

if __name__=="__main__":
    mymain()
    schedule.run()  # 开始运行，直到计划时间队列变成空为止
```

15. 什么是scrapy-redis中的指纹,是如何去重的?

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进制的转义符字
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：urll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?

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）

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

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

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）

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

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

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?

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)
```

6. 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）

```
1. session = requests.session()
2. response = session.get(url,headers)
```

7. 正则表达式操作

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?

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. 生成一个斐波那契数列?

```
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)
```

3. 说明一下os.path 和sys.path 分别代表什么?

os.path 主要是用于用户对系统路径文件的操作。
sys.path 主要用户对 Python 解释器的系统环境参数的操作。

4. 什么是lambda函数? 有什么好处?

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)

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)

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)

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）

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）

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)

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)

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

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，但该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. MonggoDB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}});

6. 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。

7. 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(nlogn).此外在最坏、最佳、平均情况下归并排序时间复杂度均为o(nlogn).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logn+1层。所以总的时间代价为cn*(logn+1).时间复杂度是o(nlogn)。

七、北京号外科技爬虫面试题

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。

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?

```
import os
import glob
from PIL import Image

def thumbnail_pic(path):
    a=glob.glob(r'./*.jpg')
    for x in a:
        name=os.path.join(path,x)
        im=Image.open(name)
         im.thumbnail((80,80))
         print(im.format,im.size,im.mode)
         im.save(name,'JPEG')
     print('Done!')

if __name__=='__main__':
     path='.'
     thumbnail_pic(path)
```

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) Beautiful Soup: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~~）。

```
import requests
s =requests.session()
login_data={"account":"   ","password":"  "}
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器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。

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。第一个文件的名字以表的名字开始，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,其执行数度比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

八、首信Python研发面试

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. 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()
```

3. 介绍一下Python的异常处理机制和自己开发过程中的体会?

1. 默认的异常处理器
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。

4. jQuery库中$()是什么？网上有5个元素，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

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。

5. 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。

6. 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。

九、微影时代

1. 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

2. 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。

十、斯沃创智

1. 简述Python 中is和= =的区别

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. 简述read,readline和readlines的区别

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

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

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

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。

```
#!/usr/bin/python
#coding=utf-8
'''
Created on 2012-2-22
Q: 给定一个列表，去掉其重复的元素，并输出
'''
def distFunc1():
    a=[1,2,4,2,4,5,6,5,7,8,9,0]
    b={}
     b=b.fromkeys(a)
     print b
     #print b.keys()
     a=list(b.keys())
     print a
def distFunc2():
    a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
    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
    print a
def distFunc3():
    #可以先把list重新排序，然后从list的最后开始扫描，代码如下：
    List=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
    if List:
        List.sort()
        #print List
        last = List[-1]
        #print last
        for i in range(len(List)-2, -1, -1):
            if last==List[i]:
                del List[i]
            else: last=List[i]

if __name__ == '__main__':
    distFunc1()
    distFunc2()
distFunc3()
```

6. Python代码中(*args, **kwargs)是什么意思

*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?

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

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?

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?

并发。

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?

7层从上到下分别是

- 应用层
- 表示层
- 会话层
- 传输层
- 网络层
- 数据链路层
- 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）主要面向通过网络的端到端的数据流。

HTTP属于应用层。

2. 什么是HTTP协议？HTTP请求有哪几种?

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服务器通迅的格式。
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?

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?

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

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。

十三、成安

1. Python的logging模块常用的几个等级?

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

2. 在HTTP1.1中常见的状态码有哪些，如何设置状态码?

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

3. Python如何处理上传文件?

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：

```
import requests #需要安装requests
with open('test.txt', 'rb') as f:
requests.get('http://服务器IP 地址:端口', data=f)
服务端代码:
var http = require('http');
var fs = require('fs');
var server = http.createServer(function(req, res){
    //console.log(req);
    var recData = "";
     req.on('data', function(data){
         recData += data;
     })
     req.on('end', function(data){
         recData += data;
         fs.writeFile('recData.txt', recData, function(err){
         console.log('file received');
         })
     })
     res.end('hello');
})
server.listen(端口);
```

十四、博派通达

1. 请列举你使用过的Python代码检测工具
2. 移动应用自动化测试 Appium
3. OpenStack 集成测试 Tempest
4. 自动化测试框架 STAF
5. 自动化测试平台 TestMaker
6. JavaScript内存泄露检测工具 Leak Finder
7. Python的Web应用验收测试 Splinter
8. 即插即用设备调试工具 UPnP-Inspector

1. 简述Python垃圾回收机制和如何解决循环引用

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被任何人使用了，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有点，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，使得侠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。

2. 请简述如何编写清晰可读的的代码

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，一般来说有一下两种方式：
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。

5. 请列出MySQL数据库查询的技巧

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。
8. 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。
9. 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。

10. 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。

8. 请列出常见的HTTP头及其作用

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

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

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
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

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 functool.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

type()
isinstance()

2. Python里面如何生成随机数

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

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）

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

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

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，ist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. 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]

```
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]

```
1. b1 =[1,2,3]
```

```
b2=[2,3,4]
b3 = []
b4 =[]
#方法一
for val in b2 :
    if  val in b1:
        b3.append(val)
print(b3)
for val in b2 :
    if val not in b1:
        b4.append(val)
for  val2 in b1:
    if val2 not in b2:
        b4.append(val2)
print(b4)
#方法二
t1 = set(b1)
t2 = set(b2)
print(t1&t2)
list1 = list(t1-t2)
list1.extend(list(t2-t1))
print(list1)
```

9. 写一段Python代码实现list里排a=[1,2,4,2,45,7,10,5,5,7,8,9,0,3]

```
a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
print(sorted(a,reverse=False))#返回新的列表
print(a)
a.sort()#注意sort没有返回值在原列表上修改
print(a)
```

10. D= [i**2 for i in range(1,10)]请写出D的输出结果

[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

```
nums = range(2,20)
for i in nums:
    nums = filter(lambda x :x == i or x % i,nums)
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。

13. 谈一下对于对线程的理解，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concurrrent.futures?

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/Dominic-Ji/p/8944723.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)

```
def sub_sort(array,low,high):
    pivotkey=array[low]
    while low<high :
        while low<high and array[high]>=pivotkey:
            high -= 1
        array[low]=array[high]
        while low<high and array[low]<=pivotkey:
            low += 1
        array[high]=array[low]
    array[low]=pivotkey
    return low

def quick_sort(array,low,high):
    if low < high :
        pivoloc=sub_sort(array,low,high)
        quick_sort(array,low,pivoloc-1)
        quick_sort(array,pivoloc+1,high)

if __name__=="__main__":
    array=[49,38,65,97,76,13,27]
    print array
    quick_sort(array,0,len(array)-1)
    print array
```

2. 简述Python在异常处理中，else和finally的作用分别是什么?

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

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

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

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。

6. HTTPS网站如何爬取

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.bbzhh.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?

```
def  add(a,s_list=[]):
     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

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

```
1. class Person:
2.      name =" aaa"
3. pl =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

```
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

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

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

```
1. list2 =["2","3","4"]
2. t = type(list2)
```

4. Python里面如何拷贝一个对象

```
1. list2 =["2","3","4"]
2. q=list2.copy()
3. print(q)
```

5. 写一段except的函数

```
a=10
b=0
try:
    c=a/b
    print(c)
except Exception as e:
    print(e)
print("done")
```

6. Pyhon里面pass语句的作用是什么?

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?

isinstance(变量名，类型);
type();

8. Python中range()函数的用法

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

```
import  random
random.randint()
```

11. 如何在function里面设置一个全局变量

```
 globals()

 globals()   返回包含当前作用域全局变量的字典。
 global 变量    设置使用全局变量
```

13. Python的传参是传值还是传址

Python是传对象引用。

14. with 语句的作用,写一段代码

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。

```
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

```
a='是一个test字符串'
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

```
a = [1,2,4,3,2,2,3,4]
t =set(a)
print(list(t))
```

3. 有一个文件test.txt里面有数据

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。

```
a=["2012-04-18",
"2012-04-19",
"2012-04-18",
"2012-04-17",
"2012-04-16"]
import datetime
def date_sort3(x):
    ls=list(x)
   #用了冒泡排序来排序，其他方法效果一样
    for j in range(len(ls)-1):
        for i in range(len(ls)-j-1):
            lower=datetime.datetime.strptime(ls[i], '%Y-%m-%d')
            upper=datetime.datetime.strptime(ls[i+1], '%Y-%m-%d')
            if lower>upper:
                ls[i],ls[i+1]=ls[i+1],ls[i]
    return tuple(ls)
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?

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?

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。

```
>>> def createGenerator():
...     mylist = range(3)
...     for i in mylist:
...         yield i*i
...
>>> mygenerator = createGenerator() # create a generator
>>> print(mygenerator) # mygenerator is an object!
<generator object createGenerator at 0xb7555c34>
```

```
>>> for i in mygenerator:
...     print(i)
0
1
4
```

3. 深拷贝，浅拷贝的区别?

在Python中对象的赋值其实就是对象的引用。当创建一个对象，把它赋值给另一个变量的时候，python并没有拷贝这个对象，只是拷贝了这个对象的引用而已。
浅拷贝：拷贝了最外围的对象本身，内部的元素都只是拷贝了一个引用而已。也就是，把对象复制一遍，但是该对象中引用的其他对象我不复制
深拷贝：外围和内部元素都进行了拷贝对象本身，而不是引用。也就是，把对象复制一遍，并且该对象中引用的其他对象我也复制。

4. 简化代码?

```
l = []
for i in range(10):
 l.append(i**2)
print l
```

简化后的来：

```
print([x**2 for x in range(10)])
```

6. 这两个参数什么意思 *args **kwargs，我们为什么要使用他们?

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?

```
内连接：inner join on ；
左连接：left join on ；
右连接：right join on ；
```

二十二、浙江从泰

2. 反爬虫措施?

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?

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starts_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在是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

```
1. str1 = "k:1|k1:2|k2:3|k3:4"
2. def str2dict(str1):
3.       dict1 = {}
4.       for iterms in str1.split('|'):
5.           key, value = iterms.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?

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

3. 写一个装饰器?

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?

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

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

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

```
1. t =(1,5,8)
2. l = list(t)
```

4. Python里面range和xrange的区别

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

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]

```
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

```
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

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

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

```
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?

```
b1 =[1,2,3]
b2 =[2,3,4]
t1 = set(b1)
t2 =set(b2)
t3 =t1&t2
t4 = t1-t2
t5 = t2-t1
print(list(t3))
print(list(t4))
print(list(t5))
```

12. 请写出一段代码求出1到100的和?

```
i =1
su=0
while i <=100:
    su =su+i
    i+=1
print(su)
```

13. Python中正则表达式提取出字符串中的数字

```
s ='12j33jk12ksdjfkj23jk4h1k23h'
import  re
b=re.findall("\d",s)
b="".join(b)
print(b)
```

14. 补全下列代码

```
def deco(func):
    pass“补全代码”
@deco
def myfunc(a,b):
    print("myfunc(%s,%s) called" %(a,b))
    return a+b
```

补充后:

```
def deco(func):
 def inner(a,b):
  return func(a,b)
 return inner

@deco
def myfunc(a,b):
 print("myfunc(%s,%s) called" %(a,b))
 return a+b
```

二十五、名企片爬虫面试题

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

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

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，进程的目的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

```
1. CREATE INDEX name ON table_name (person)
```

4. *args and **kwargs的区别

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

```
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

1）通过Headers反爬虫
从用户请求的Headers反爬虫是最常见的反爬虫策略。很多网站都会对Headers的User-Agent进行检测，还有一部分网站会对Referer进行检测（一些资源网站的防盗链就是检测Referer）。如果遇到了这类反爬虫机制，可以直接在爬虫中添加Headers，将浏览器的User-Agent复制到爬虫的Headers中；或者将Referer值修改为目标网站域名。对于检测Headers的反爬虫，在爬虫中修改或者添加Headers就能很好的绕过。
2）基于用户行为反爬虫
还有一部分网站是通过检测用户行为，例如同一IP短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。 大多数网站都是前一种情况，对于这种情况，使用IP代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理ip，检测后全部保存起来。这样的代理ip爬虫经常会用到，最好自己准备一

个。有了大量代理ip后可以每请求几次更换一个ip，这在requests或者urllib2中很容易做到，这样就能很容易的绕过第一种反爬虫。 对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。
3）动态页面的反爬虫
上述的几种情况大多都是出现在静态页面，还有一部分网站，我们需要爬取的数据是通过ajax请求得到，或者通过JavaScript生成的。首先用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

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

8. 排序算法有哪些用Python写一种排序算法

冒泡排序:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import random
unsortedList=[]
# generate an unsorted list
def generateUnsortedList(num):
    for i in range(0,num):
        unsortedList.append(random.randint(0,100))
    print unsortedList
# 冒泡排序
def bubbleSort(unsortedList):
    list_length=len(unsortedList)
    for i in range(0,list_length-1):
        for j in range(0,list_length-i-1):
            if unsortedList[j]>unsortedList[j+1]:
                unsortedList[j],unsortedList[j+1]=unsortedList[j+1],unsortedList[j]
    return unsortedList
generateUnsortedList(20)
print bubbleSort(unsortedList)
```

选择排序

```
def selectionSort(unsortedList):
    list_length=len(unsortedList)
    for i in range(0,list_length-1):
        for j in range(i+1,list_length):
            if unsortedList[i]>unsortedList[j]:
                unsortedList[i],unsortedList[j]=unsortedList[j],unsortedList[i]
    return unsortedList
```

快排:

```
def quickSort(unsortedList):
    if len(unsortedList)<2:
        return unsortedList
    less=[]
    greater=[]
    middle=unsortedList.pop(0)
    for item in unsortedList:
        if item<middle:
            less.append(item)
        else:
            greater.append(item)
    return quickSort(less)+[middle]+quickSort(greater)
```

归并排序：

```
def mergeSort(unsortedList):
    if len(unsortedList)<2:
        return unsortedList
    sortedList=[]
    left=mergeSort(unsortedList[:len(unsortedList)/2])
    right=mergeSort(unsortedList[len(unsortedList)/2:])
    while len(left)>0 and len(right)>0:
        if left[0]<right[0]:
            sortedList.append(left.pop(0))
        else:
            sortedList.append(right.pop(0))
    if len(left)>0:
        sortedList.extend(mergeSort(left))
    else:
        sortedList.extend(mergeSort(right))
    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

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

PIL、pytesser、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

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RFPDupeFilter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？

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

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
.....

二十八、傲盾

1. Python线程和进程的区别?

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?

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

```
# 方法一，使用zip函数
a = ['hello', 'world', '1', '2']
b = dict(zip(a[0::2], a[1::2]))
print(b)

# 方法二，利用循环
b = {}
for i in range(0, len(a), 2):
    b[a[i]] = a[i+1]
print(b)

# 使用enumerate 函数生成index实现
my_dict = {}
for index, item in enumerate(a):
    if index % 2 == 0:
        my_dict[item] = a[index+1]
print(b)
```

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?

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

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=KWJHUG6JJM65HS2K6 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Sesion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

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。

三十、大会信统Python工程师

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。

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

方法二：使用字典函数。

```
a=[1,2,4,2,4,5,6,5,7,8,9,0]
  b={}
  b=b.fromkeys(a)
  c=list(b.keys())
  print('去重后的list为：',c)
```

方法三：用append方法实现。

```
def delList(L):
    L1 = []
    for i in L:
        if i not in L1:
            L1.append(i)

    return L1

 print(delList([1,2,2,3,3,4,5]))
print(delList([1,8,8,3,9,3,3,3,3,6,3]))
```

换成列表推导式更简单：

```
l1 = ['b','c','d','b','c','a','a']
l2 = []
[l2.append(i) for i in l1 if not i in l2]
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。

```
    List=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
    if List:
        List.sort()
        #print List
        last = List[-1]
        #print last
        for i in range(len(List)-2, -1, -1):
            if last==List[i]:
                del List[i]
             else:
                 last=List[i]
                 #print(list)
     print(List)
```

2. 编程用sort进行排序，然后从最后一个元素开始判断

```
a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
a.sort()
last=a[-1]
for i in range(len(a)-2,-1,-1):
    if last==a[i]:
       del a[i]
```

```
    else:
        last=a[i]
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。

```
>>>print(re.match('super', 'superstition').span())

(0, 5)

>>>print(re.match('super', 'insuperable'))

None

>>>print(re.search('super', 'superstition').span())

(0, 5)

>>>print(re.search('super', 'insuperable').span())

(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<=x<=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

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 而抽类象不是。
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。

4. 请用代码实现一个冒泡排序?

```
def bubbleSort(nums):
    for i in range(len(nums)-1):    # 这个循环负责设置冒泡排序进行的次数
        for j in range(len(nums)-i-1):  # j 为列表下标
            if nums[j] > nums[j+1]:
                nums[j], nums[j+1] = nums[j+1], nums[j]
    return nums
nums = [5,2,45,6,8,2,1]
print （bubbleSort(nums)
```

5. 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：

```
 class FibIterator(object):
    """斐波那契数列迭代器"""
    def __init__(self, n):
        """
        :param n: int, 指明生成数列的前n个数
        """
        self.n = n
        # current用来保存当前生成到数列中的第几个数了
        self.current = 0
         # num1用来保存前前一个数，初始值为数列中的第一个数0
         self.num1 = 0
         # num2用来保存前一个数，初始值为数列中的第二个数1
         self.num2 = 1

     def __next__(self):
        """被next()函数调用来获取下一个数"""
        if self.current < self.n:
            num = self.num1
            self.num1, self.num2 = self.num2, self.num1+self.num2
            self.current += 1
            return num
        else:
            raise StopIteration

     def __iter__(self):
        """迭代器的__iter__返回自身即可"""
        return self
```

迭代器实现：

```
      def fib(n):
....:     current = 0
....:     num1, num2 = 0, 1
....:     while current < n:
....:         num = num1
....:         num1, num2 = num2, num1+num2
....:         current += 1
....:         yield num
....:     return 'done'
```

6. 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

7. 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)。

8. 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、上海行知道教育Python程序员笔试题

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。

```
# mysingle.py
class MySingle:
  def foo(self):
    pass

sinleton = MySingle()

将上面的代码保存在文件mysingle.py 中，然后这样使用：
from mysingle import sinleton
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，

```
class Singleton(object):
    def __new__(cls):
        # 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
        if not hasattr(cls, 'instance'):
            cls.instance = super(Singleton, cls).__new__(cls)
        return cls.instance

obj1 = Singleton()
obj2 = Singleton()

obj1.attr1 = 'value1'
print obj1.attr1, obj2.attr1
print obj1 is obj2

输出结果：
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

```
def singleton(cls):
    instances = {}
    def getinstance(*args,**kwargs):
        if cls not in instances:
            instances[cls] = cls(*args,**kwargs)
        return instances[cls]
    return getinstance

@singleton
class MyClass:
    a = 1

c1 = MyClass()
c2 = MyClass()
print(c1 == c2) # True


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，
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，
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

```
class Singleton2(type):
    def __init__(self, *args, **kwargs):
        self.__instance = None
        super(Singleton2,self).__init__(*args, **kwargs)

    def __call__(self, *args, **kwargs):
        if self.__instance is None:
            self.__instance = super(Singleton2,self).__call__(*args, **kwargs)
        return self.__instance


class Foo(object):
    __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。


foo1 = Foo()
foo2 = Foo()
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）
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

```
def sum(x,y):
return x+y
```

用lambda来实现：

```
p = lambda x,y:x+y
print(4,6)
```

例2：传入一个参数的lambda函数

```
a=lambda x:x*x
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：

```
a = lambda x,y,z:(x+8)*y-z
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。

```
class FloatRange:

    def __init__(self, start, end, step):
        self.start = start
        self.end = end
        self.step = step

    # 正向迭代
    def __iter__(self):

        t = self.start
        while round(t, 2) <= round(self.end, 2):
            yield t
            t += self.step
```

```
    # 反向迭代
    def __reversed__(self):

        t = self.end
        while round(t, 2) >= round(self.start, 2):
            yield t
            t -= self.step


if __name__ == "__main__":

    for x in FloatRange(3.0, 4.0, 0.2):
        print x

    print ""

    for x in reversed(FloatRange(3.0, 4.0, 0.2)):
        print x
```