python-interview

[最新版本链接地址](https://github.com/wangyitao/python-interview)

##### 为什么学python

[为什么学python](https://blog.csdn.net/Darkman_EX/article/details/81101232)

* 答题路线：a、python的优点，b、python的应用领域广
* 具体：
  + 优点
    - 1、python语法非常优雅，简单易学
    - 2、免费开源
    - 3、跨平台，可以自由移植
    - 4、可扩展，可嵌入性强
    - 5、第三方库丰富
  + 应用领域
    - 1、在系统编程中应用广泛，比如说shell工具。
    - 2、在网络爬虫方面功能非常强大，常用的库如scrapy，request等
    - 3、在web开发中使用也很广泛，如很多大型网站都用python开发的，如ins，youtube等，常用的框架如django，flask等
    - 4、python在系统运维中应用广泛，尤其在linux运维方面，基本上都是自动化运维。
    - 5、在人工智能，云计算，金融等方面也应用非常广泛。

##### 通过什么途径学习python

* 通过看官方文档
* 通过哔哩哔哩上的视频教程
* 通过百度查资料
* 通过买python相关方面的书

##### 公司线上和开发环境使用的什么系统

* 线上用的centos和Ubuntu系统
* 开发环境用的windows，mac，还有Ubuntu。

##### python和java、php、c、c#、c++ 等其他语言对比？

* Java C# PHP Python (开发效率高)，这些语言本身不需要你去管理内存了。它们都有自己的虚拟机，对于开辟内存、释放内存都由这个虚拟机处理。
* C 和 Python、Java、C#等C语言： 代码编译得到 机器码 ，机器码在处理器上直接执行，每一条指令控制CPU工作其他语言： 代码编译得到 字节码 ，虚拟机执行字节码并转换成机器码再后在处理器上执行Python 和 C Python这门语言是由C开发而来
* 对于使用：Python的类库齐全并且使用简洁，如果要实现同样的功能，Python 10行代码可以解决，C可能就需要100行甚至更多.
* 对于速度：Python的运行速度相较与C，绝逼是慢了Python 和 Java、C#等
* 对于使用：Linux原装Python，其他语言没有；以上几门语言都有非常丰富的类库支持
* 对于速度：Python在速度上可能稍显逊色所以，Python和其他语言没有什么本质区别，其他区别在于：擅长某领域、人才丰富、先入为主

##### 简述解释型和编译型编程语言

* 解释型语言编写的程序不需要编译，在执行的时候，专门有一个解释器能够将VB语言翻译成机器语言，每个语句都是执行的时候才翻译。这样解释型语言每执行一次就要翻译一次，效率比较低。
* 用编译型语言写的程序执行之前，需要一个专门的编译过程，通过编译系统，把源高级程序编译成为机器语言文件，翻译只做了一次，运行时不需要翻译，所以编译型语言的程序执行效率高，但也不能一概而论，部分解释型语言的解释器通过在运行时动态优化代码，甚至能够使解释型语言的性能超过编译型语言。

##### python解释器种类以及特点

* CPython
  + c语言开发的 使用最广的解释器
* IPython
  + 基于cpython之上的一个交互式计时器 交互方式增强 功能和cpython一样
* PyPy
  + 目标是执行效率 采用JIT技术 对python代码进行动态编译，提高执行效率
* JPython
  + 运行在Java上的解释器 直接把python代码编译成Java字节码执行
* IronPython
  + 运行在微软 .NET 平台上的解释器，把python编译成. NET 的字节码

##### 位和字节的关系

* 8位=一字节

##### b、B、kB、MB、GB的关系

1. 1B=8b
2. 1kB=1024B
3. 1MB=1024kB
4. 1GB=1024MB

##### 请列出至少5个PEP8规范

[PEP8规范](https://www.jianshu.com/p/e132bea1d2c9) \* 每个缩进级别使用4个空格 \* 每行代码的最大长度限制为79个字符 \* 若是导入多个库函数，应该分开依次导入 \* 道路应按照以下顺序导入 a、标准库导入 b、相关的第三方库导入 c、本地应用程序的库导入 \* 在表达式中避免无关的空格 \* 在括号或者大括号内 \* 在尾随逗号和后面的右括号之间 \* 在逗号，分号或者冒号前面 \* 函数名的与后面的参数的括号之间 \* 代码更改时，相应的注释也要随之更改 \* 命名要规范，通俗易懂

##### or 和 and

* 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
* 结果：
  + v1 = 1
  + v2 = 3
  + v3 = 0
  + v4 = 1
  + v5 = 1
  + v6 = False
* 基本运算规律
  1. 在不加括号时候, and优先级大于or
  2. x or y 的值只可能是x或y. x为真就是x, x为假就是y
  3. x and y 的值只可能是x或y. x为真就是y, x为假就是x

##### ascii、Unicode、utf-8、gbk的区别

* ascii 是最早美国用的标准信息交换码，把所有的字母的大小写，各种符号用 二进制来表示，共有256中，加入些拉丁文等字符，1bytes代表一个字符
* Unicode是为了统一世界各国语言的不用，统一用2个bytes代表一个字符，可以表达2^16=65556个，称为万国语言，特点：速度快，但浪费空间
* utf-8 为了改变Unicode的这种缺点，规定1个英文字符用1个字节表示，1个中文字符用3个字节表示，特点；节省空间，速度慢，用在硬盘数据传输，网络数据传输，相比硬盘和网络速度，体现不出来的
* gbk  是中文的字符编码，用2个字节代表一个字符

##### 字节码和机器码的区别

* 机器码是电脑CPU直接读取运行的机器指令，运行速度最快，但是非常晦涩难懂，也比较难编写，一般从业人员接触不到。
* 字节码是一种中间状态（中间码）的二进制代码（文件）。需要直译器转译后才能成为机器码。

##### 三元运算编写格式

* 表达式1 if 布尔表达式2 else 表达式3
* 例如：a=3 if 3 > 4 else 5

##### 列举你所了解的所有Python2和Python3的区别

[python2和python3的区别](https://blog.csdn.net/weixin_41819299/article/details/81259721) 1. python2没有nonlocal关键字，要修改临时变量只能将其改成可变数据类型，如数组。b=[a] 2. print()函数代替print语句 3.  Python3加入 Unicode 字符串，用以编码存储字符串。比如用 utf-8可以用来输入中文 4.  Python3去掉long类型，新增了bytes。 5.  Python 3整数之间做除法可以得到浮点数的结果，不需要进行数据格式转换1/2=0.5 Python 2整数int间除法结果为把运算结果去尾的整数1/2=0，3/2.0=1.5 6.   Python3 中 range()，Python2 中 xrange()。 7.   python2中的不等于号可以是!=或者<>，python3只能是!= 8.   python2中raw\_input()用来输入字符串，而python3中使用input()来输入字符串

##### py2项目如何迁移成py3

1. 先备份原文件，然后使用python3自带工具2to3.py将py2文件转换位py3文件
2. 手动将不兼容的代码改写成兼容py3的代码

##### 用一行代码实现数值交换

* a=1
* b=2
* 答案：a,b=b,a

##### python3和python2中int和long的区别

* python2中有long类型，python3中没有long类型，只有int类型。python3中的int类型包括了long类型。

##### xrange和range的区别

* xrange和range用法相同，但是xrange是一个生成器，range结果是一个列表。xrange做循环的时候性能比range好。

##### 如何实现字符串的反转？如：name=felix，反转成name=xilef

name = "felix"  
 # 方法一：  
 name=name[::-1]  
 # 方法二：  
 name2=list(name)  
 name2.reverse()  
 name=''.join(name2)  
 # 方法三：  
 from functools import reduce  
 name=reduce(lambda x, y: y+x, name)

##### 文件操作时，xreadlines和readlines的区别

* xreadlines返回的是一个生成器
* readlines返回的是一个列表

##### 请列举布尔值位False的常见值

* 0、''、[]、{}、tuple()、None、set()

##### 列举字符串、列表、元组、字典每个常用的5个方法

* 字符串---[字符串方法总结](https://www.cnblogs.com/chendai21/p/8137285.html)
  1. strip() ->去掉字符串两端的空白符
  2. split() ->对字符串进行分割，默认按照空格分割
  3. join() ->字符串连接
  4. startwith(),endwith() ->判断是否以啥开头或者结尾
  5. replace() -> 字符串替换
  6. find() -> 查找字符串，存在返回第一个索引，不存在返回-1
* 列表---[列表方法总结](https://www.cnblogs.com/smelond/p/7857701.html)
  1. count() ->统计在列表中出现的个数
  2. apped() ->在列表末尾添加值
  3. pop() ->删除一个对象，默认最后一个
  4. remove() ->删除指定的第一个匹配项
  5. insert() ->插入对象
  6. index() ->获取索引
* 元组
  1. count() ->统计在元组中出现的个数
  2. index() ->获取索引
* 字典
  1. keys() ->获取所有的键
  2. pop() ->删除指定的键的键值对
  3. popitem() ->随机删除一个键值对
  4. update() ->更新字典，参数为一个字典，如果键已存在，则更改，不存在则添加
  5. setdefault() ->如果键存在则，返回该键对应的值，如果不存在，设置该键为设置的默认值，然后返回该键对应的值
  6. get() ->返回键对应的值
  7. fromkeys() ->创建字典，第一个参数为可迭代对象，每个值变成字典的键，第二个参数为每个键的默认值

##### is和==的区别

* is比较的是两个对象的id是否相同
* ==比较的是两个对象的值是否相同

##### 1，2，3，4，5能组成多少个互不相同且不重复的三位数？

* 排列组合问题： 5\*4\*3=60个

##### 什么是反射，以及应用场景

[什么是反射的解释](https://www.cnblogs.com/IT-Scavenger/p/9394306.html)

* 反射就是通过字符串的形式，导入模块；通过字符串的形式，去模块寻找指定函数，并执行。利用字符串的形式去对象（模块）中操作（查找/获取/删除/添加）成员，一种基于字符串的事件驱动！
* 应用场景：当我们动态的输入一个模块名的时候就可以使用到反射。
* 通过hasattr，getattr，delattr，setattr等四个函数来操作

##### 简述python的深浅拷贝

* 浅拷贝只是对另外一个变量的内存地址的拷贝，这两个变量指向同一个内存地址的变量值。
  + 浅拷贝的特点：
    - 共用一个值
    - 这两个变量的内存地址一样
    - 对其中一个变量的值改变，另外一个变量的值也会改变
* 深拷贝是一个变量对另外一个变量的值拷贝
  + 深拷贝的特点：
    - 两个变量的内存地址不同
    - 两个变量各有自己的值，且互不影响
    - 对其任意一个变量的值的改变不会影响另外一个
* 如果是不可变类型，则深浅拷贝只拷贝引用，如果是可变类型，浅拷贝只拷贝第一层引用，深拷贝无论多少层引用都拷贝

##### python的垃圾回收机制

[python垃圾回收机制详解](https://testerhome.com/topics/16556)

* 概述：python采用的是引用计数机制为主，标记-清除和分代收集两种机制为辅的策略
* 引用计数：
  + 每当新的引用指向该对象时，引用计数加1，当对该对象的引用失效时，引用计数减1，当对象的引用计数为0时，对象被回收。缺点是，需要额外的空间来维护引用计数，并且无法解决对象的循环引用。
* 分代回收：（具体见上面链接）
  + 以时间换空间的回收方式
* 标记清除：
  + 活动对象会被打上标记，会把那些没有被打上标记的非活动对象进行回收。

##### python的可变类型和不可变类型的区别

* 可变类型有：列表，字典等
* 不可变类型有：数字，字符串，元组等
* 这里的可变不可变是指内存中的那块内容是否可以被改变。

##### 求下面代码结果

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

* 结果： {'k1': [666], 'k2': [666]} {'k1': 777, 'k2': [666]}

##### 一行代码实现删除列表中的所有的重复的值

lis=[1,1,2,1,22,5]  
lis=list(set(lis))

##### 如何实现"1.2.3"变成['1','2','3']?

s="1,2,3"  
s=s.split(',')

##### 如何实现['1','2','3']变成[1,2,3]

ss=['1', '2', '3']  
ss=[int(i) for i in ss]

##### 比较：a=[1,2,3]和b=[(1),(2),(3)]以及c=[(1,),(2,),(3,)]的区别

* a和b的结果相同，列表里面的值相同，类型也相同
* c中的列表里面的值是元组类型的

##### 如何用一行代码生成[1,4,9,16,25,36,49,64,81,100]?

lis=[i\*\*2 for i in range(1,11)]

##### 常用字符串格式化有哪几种？

1. 使用百分号

* print('hello %s and %s'%('friend','another friend'))

1. 使用format

* print('hello {first} and {second}'.format(first='friend',second='another friend'))

##### 什么是断言(assert)?应用场景？

[断言的参考](https://blog.csdn.net/shujuanyaning/article/details/47184541)

* assert是用来检查一个条件，如果它为真，就不做任何事。如果它为假，则会抛出AssertError并且包含错误信息。
* 应用场景：
  1. 防御型编程
  2. 运行时检查程序逻辑
  3. 检查约定
  4. 程序常量
  5. 检查文档

##### 有两个字符串列表a和b，每个字符串是由逗号隔开的一些字符

a=[  
 'a,1',  
 'b,3,22',  
 'c,3,4',  
]  
b=[  
 'a,2',  
 'b,1',  
 'd,5',  
]  
# 按照a，b中每个字符串的第一个值，合并成c如下：  
c=[  
 'a,1,2',  
 'b,3,22,1',  
 'c,3,4',  
 'd,5'  
]

# 解法：  
  
a=[  
 'a,1',  
 'b,3,22',  
 'c,3,4',  
]  
b=[  
 'a,2',  
 'b,1',  
 'd,5',  
]  
a\_dic={}  
for s in a:  
 k,v = s.split(',',1)  
 a\_dic[k]=v  
b\_dic={}  
for s in b:  
 k,v = s.split(',',1)  
 b\_dic[k]=v  
c\_dic=a\_dic  
for k,v in b\_dic.items():  
 if k in c\_dic:  
 c\_dic[k]=','.join([c\_dic[k],v])  
 else:  
 c\_dic[k]=v  
c=[','.join([k,c\_dic[k]]) for k in c\_dic]  
print(c)

##### 有一个多层嵌套的列表A=[1,2,3,[4,1,['j1',1,[1,2,3,'aa']]]],请写一段代码将A中的元素全部打印出来

A=[1,2,3,[4,1,['j1',1,[1,2,3,'aa']]]]  
def my\_print(lis):  
 for i in lis:  
 if type(i)==list:  
 my\_print(i)  
 else:  
 print(i)  
my\_print(A)

##### a=range(10),则a[::-3]的值是？

* [9,6,3,0] 或者 range(9,-1,-3)

##### 将下面列表中的元素根据位数合并成字典：

lst = [1,2,4,8,16,32,64,128,256,512,1024,32769,65536,4294967296]  
  
# 结果  
{1: [1, 2, 4, 8], 2: [16, 32, 64], 3: [128, 256, 512], 4: [1024], 5: [32769, 65536], 10: [4294967296]}

lst = [1,2,4,8,16,32,64,128,256,512,1024,32769,65536,4294967296]  
dic={}  
for i in lst:  
 len\_i=len(str(i))  
 dic.setdefault(len\_i,[]).append(i)  
print(dic)

##### 用尽量简洁的方法将二维数组合并成一维数组

lst = [[1,2,3],[4,5,6],[7,8,9]]  
ll=[]  
for l in lst:  
 # ll+=l  
 ll.extend(l)  
print(ll)

##### 将列表按照下列规则排序：

1. 正数在前，负数在后
2. 正数从小到大
3. 负数从大到小

* 例子：
  + 排序前：[7,-8,5,4,0,-2,-5]
  + 排序后：[0, 4, 5, 7, -2, -5, -8]
  + lis = [7,-8,5,4,0,-2,-5]  
    lis=sorted(lis,key=lambda x:(x<0,abs(x))) # 这里排序条件返回元组，先比较第一个，再第二个值  
    print(lis)

##### 解决哈希冲突的算法有哪几种？分别有什么特点？

[哈希冲突参考](https://blog.csdn.net/seulzz/article/details/77163878) 1. 开放定址法 2. 再哈希法 3. 链地址法 4. 建立公共溢出区

##### 简述python字符串的驻留机制

[python字符串驻留机制参考文档](https://www.cnblogs.com/asmer-stone/p/4802800.html)

* 相同对象的引用都指向内存中的同一个位置，这个也叫python的字符串驻留机制
* python的引用计数机制，并不是对所有的数字，字符串，他只对”[0-9][a-z][A-Z] 和"\_"(下划线)  ”有效“，当字符串中由其他字符比如“！ @ # ￥ % -”时字符驻留机制是不起作用的。

##### 以下代码输出什么？

lis=['a','b','c','d','e']  
print(lis[10:])

* 答案：[]

##### python哪些类型的数据才能作为字典的key？

* 可哈希的类型

##### 有如下代码：

import copy  
a=[1,2,3,[4,5],6]  
b=a  
c=copy.copy(a)  
d=copy.deepcopy(a)  
b.append(10)  
c[3].append(11)  
d[3].append(12)

* 求a，b，c，d
* 答案： > a：[1, 2, 3, [4, 5, 11], 6, 10] > b：[1, 2, 3, [4, 5, 11], 6, 10] > c：[1, 2, 3, [4, 5, 11], 6] > d：[1, 2, 3, [4, 5, 12], 6]

##### 对字典d={'a':30,'g':17,'b':25,'c':18,'d':50,'e':36,'f':57,'h':25}按照value字段进行排序

d={'a':30,'g':17,'b':25,'c':18,'d':50,'e':36,'f':57,'h':25}  
dd=sorted(d.items(),key=lambda x:x[1])  
print(dd)

##### 找出两个列表中相同的元素和不同的元素

list1=[1,2,3,5,8,7,11,10]  
list2=[5,15,25,10]  
sim=[i for i in list1 if i in list2]  
diff=[i for i in list1+list2 if i not in sim]  
print(sim)  
print(diff)

##### 二叉树是非线性结构，栈和队列以及线性表都是线性结构，对吗？

* 对的

##### 从0-99这100个数中随机取出10个，要求不能重复

import random  
lis=random.sample(range(0,100),10)  
print(lis)

##### 有一个列表lis=['This','is','a','Man','B','!']，对它进行大小写无关的排序

lis=['This','is','a','Man','B','!']  
lis=sorted(lis,key=str.lower)  
print(lis)

##### 描述以下字典的items()方法和iteritems()方法有啥不同？

* 字典的items方法作用：是可以将字典中的所有项，以列表方式返回。因为字典是无序的，所以用items方法返回字典的所有项，也是没有顺序的。
* 字典的iteritems方法作用：与items方法相比作用大致相同，只是它的返回值不是列表，而是一个迭代器

##### 请列举你所知道的python代码检测工具以及他们之间的区别

* pylint --- 源代码分析器，可以分析python代码中的错误
* pyflakes --- 检查源文件错误的简单程序，不会检查代码风格。
* pep8 --- 检查代码规范的工具

##### 介绍一下try except的用法和作用？

* 主要用来处理异常
* 完整用法如下：
* try:  
       Normal execution block  
  except A:  
       Exception A handle  
  except B:  
       Exception B handle  
  except:  
       Other exception handle  
  else:  
       if no exception,get here  
  finally:  
       print("finally")

##### 阅读以下代码，写输出结果

lis = [2,4,5,6,7]  
for i in lis:  
 if i % 2==0:  
 lis.remove(i)  
print(lis)

* 结果：[4, 5, 7]

##### 对列表[3,1,-4,-2]按照绝对值排序

lis=[3,1,-4,-2]  
lis=sorted(lis,key=lambda x:abs(x))  
print(lis)

##### 获取python解释器版本的方法

* 终端执行python -V

##### 现有mydict和变量onekey，请写出从mydict中取出onekey的值的方法

方法一：mydict[onekey] 这种方法，如果mydict中键不存在的时候程序会报错 方法二：mydict.get(onekey) 这种方法，如果存在，返回值，不存在返回None 方法三：mydict.setdefault(onekey,[]) 这种方法：存在的话返回值，不存在的时候创建一个值，值得内容为第二个参数+

##### 列表中保留顺序和不保留顺序去重 [参考文章](https://blog.csdn.net/Jerry_1126/article/details/84677212)

* 不保留顺序
* lis=[3, 1, 4, 2, 3]  
  print(list(set(lis)))
* 保留顺序 ```python lis=[3, 1, 4, 2, 3] T=[][T.append(i) for i in lis if i not in T]) print(T)

# 或者

T=sorted(set(lis), key=lis.index) print(T) ```

##### 在什么情况下y!=x-(x-y)会成立？

* x，y是两个不相等的非空集合

##### 实现99乘法表（使用两种方法）

print('\n'.join(['\t'.join(['{}\*{}={}'.format(x,y,x\*y) for x in range(1,y+1)]) for y in range(1,10)]))

for i in range(1,10):  
 for j in range(1,i+1):  
 print('%s\*%s=%s'%(i,j,i\*j),end='\t')  
 else:  
 print()

##### 判断dict中有没有某个key。

* key in dict.keys() 判断

##### a = dict(zip(('a','b','c','d','e'),(1,2,3,4,5))) 请问a是什么？

* {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

##### 在python中如何拷贝一个对象，并说明他们之间的区别

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（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deep.deepcopy()函数）

##### python中进制转换

进制转换以十进制为媒介 十六进制前面加上0x，八进制加上0o，二进制前面加上0b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 二进制 | 八进制 | 十进制 | 十六进制 |
| 二进制 |  | bin(int(x, 8)） | bin(int(x, 10)) | bin(int(x, 16)) |
| 八进制 | oct(int(x, 2)) |  | oct(int(x, 10)) | oct(int(x, 16)) |
| 十进制 | int(x, 2) | int(x, 8) |  | int(x, 16) |
| 十六进制 | hex(int(x, 2)) | hex(int(x, 8)) | hex(int(x, 10)) |  |

##### 将列表alist=[{'name':'a','age':25},{'name':'b','age':30},{'name':'c','age':20}]，按照age的值从大到小排列。

alist=[{'name':'a','age':25},{'name':'b','age':30},{'name':'c','age':20}]  
blist=sorted(alist,key=lambda x:x['age'],reverse=True)  
print(blist)

##### 关于Python程序的运行方面，有什么手段能提升性能？

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

##### python是如何进行内存管理的？python的程序会内存泄漏吗？说说有没有什么方面阻止或者检测内存泄漏？

[整体参考文章](https://www.jianshu.com/p/2b683cb5837c) \* python是如何进行内存管理的[参考文章](https://blog.csdn.net/u010967872/article/details/80301633) 1. 引用计数 \* Python内部使用引用计数，来保持追踪内存中的对象，Python内部记录了对象有多少个引用，就是引用计数，当对象被创建时就创建了一个引用计数，当对象不再需要的时候，这个对象的引用计数为0时，他被垃圾回收。 2. 垃圾回收 \* 当内存中有不再使用的部分时，垃圾收集器就会把他们清理掉。他会去检查那些引用计数为0的对象，然后清除其在内存中的空间。当然除了引用计数为0的会被清除，还有一种情况也会被垃圾收集器清掉，当两个对象相互引用时，他们本身其他引用已经为0了。 3. 内存池机制 \* Python提供了对内存的垃圾收集机制，但是他将不用的内存放到内存池而不是反回给操作系统。 \* python的程序会内存泄漏吗？ \* 会发生内存泄漏，在Python程序里，内存泄漏是由于一个长期持有的对象不断的往一个dict或者list对象里添加新的对象, 而又没有即时释放，就会导致这些对象占用的内存越来越多，从而造成内存泄漏。另外，对象的交叉引用也会造成内存无法释放的问题。 \* 说说有没有什么方面阻止或者检测内存泄漏？ 1. 程序员管理好每个python对象的引用，尽量在不需要使用对象的时候，断开所有引用 2. 尽量少通过循环引用组织数据，可以改用weakref做弱引用或者用id之类的句柄访问对象 3. 通过gc模块的接口可以检查出每次垃圾回收有哪些对象不能自动处理，再逐个逐个处理

##### 一个大小为100G的文件etl\_log.txt，要读取文件的内容，写出具体过程代码

with open("etl\_log.txt",'r',encoding='utf8') as f:  
 for line in f:  
 print(line,end='')

##### python代码如何获取命令行参数

[获取命令行参数的方法参考](https://www.cnblogs.com/ouyangpeng/p/8537616.html) 1.使用sys模块 \* 通过sys.argv来获取 2. 使用getopt模块

##### 写代码：如何由tuple1=('a','b','c','d','e')，和tuple2=(1,2,3,4,5)得到res={'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

tuple1=('a','b','c','d','e')  
tuple2=(1,2,3,4,5)  
res=dict(zip(tuple1,tuple2))  
print(res)

##### 1<(2==2)和1<2==2的结果分别是什么？

* 第一个为False，第二个为True，暂时按照第一个类型进行相应的转换

##### 如何打乱一个排好序的列表

* 使用random.shuffle()
* import random  
  alist=[1,2,3,4,5,6]  
  random.shuffle(alist)  
  print(alist)

##### 如何查找一个字符串中特定的字符？find和index的差异？

* 使用find和index方法查找

1. find()方法：查找子字符串，若找到返回从0开始的下标值，若找不到返回-1
2. index()方法：python 的index方法是在字符串里查找子串第一次出现的位置，类似字符串的find方法，不过比find方法更好的是，如果查找不到子串，会抛出异常，而不是返回-1

##### 把a='aaabbcccdddde'这种形式的字符串，压缩成a3b2c3d4e1这种形式。

a='aaabbcccdddde'  
aa=''  
for i in sorted(list(set(a)),key=a.index):  
 aa=aa+i+str(a.count(i))  
print(aa)

##### 一个数如果恰好等于它的因子之和，这个数就称为‘完数’，比如6=1+2+3，编程找出1000以内的所有的完数。

wanshu=[]  
for i in range(1,1001):  
 s=0  
 for j in range(1,i//2+1):  
 if i % j ==0:  
 s+=j  
 else:  
 if i==s:  
 wanshu.append(i)  
print(wanshu)

##### 输入一个字符串，输出该字符串的字符的所有组合。如输入'abc',输出a,b,c,ab,ac,bc,abc.

def getC(s):  
 if not s:  
 return  
 len\_s=len(s)  
 ss=[]  
 for i in range(len\_s):  
 combination(s,0,i,ss)  
aaa=[]  
def combination(s,index,num,ss):  
 global aaa  
 if num==-1:  
 return  
 if index==len(s):  
 return  
 ss.append(s[index])  
 aaa.append(''.join(ss))  
 combination(s,index+1,num-1,ss)  
 ss.pop()  
 combination(s,index+1,num,ss)  
   
getC('123')  
print(aaa)  
print(sorted(set(aaa),key=lambda x:len(str(x))))

##### 给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。给定的字符串只含有小写英文字母，并且长度不超过10000。例如：'ababab',返回True，'ababa'，返回False

def solution(s):  
 ll=len(s)  
 for i in range(1,ll//2+1):  
 if ll % i == 0:  
 j=0  
 while s[:i]==s[j:j+i] and j<ll:  
 j=j+i  
 if j==ll:  
 return True  
 return False  
   
print(solution('abababa'))

##### python递归的最大层数？

* 1000

##### filter、map、reduce的作用。

1. filter() 相当于过滤器的作用 python s=[1,2,3,5,6,8,9,10,25,12,30] # 筛选出3的倍数 # 第一个参数为一个返回True或者False的函数，第二个参数为可迭代对象 # 该函数把可迭代对象依次传入第一个函数，如果为True，则筛选 d=filter(lambda x:True if x % 3 == 0 else False,s) print(list(d))
2. map()函数，

* # 第一个参数为函数，依次将后面的参数传给第一个函数，并执行函数  
  # 如果有多个参数则，依次将后面的对应传给参数  
  s=map(lambda x,y:x+y,range(10),range(10))  
  print(list(s))  
  ss=map(lambda x:x\*x,range(10))  
  print(list(ss))

1. reduce()函数

* from functools import reduce  
  # 开始的时候将可迭代对象的第一个数和第二个数当成x和y  
  # 然后将第一次函数的执行结果当成x，然后再传入一个数当成y  
  # 再执行函数  
  s=reduce(lambda x,y:x+y,range(101))  
  print(s) # 相当于0+1+2+……+99+100

##### 什么是闭包

* 在函数中可以（嵌套）定义另一个函数时，如果内部的函数引用了外部的函数的变量，则可能产生闭包。闭包可以用来在一个函数与一组“私有”变量之间创建关联关系。在给定函数被多次调用的过程中，这些私有变量能够保持其持久性。
* # 内部函数使用了外部函数的变量，就相当于闭包  
  def func1():  
  a=1  
  def inner():  
   return a  
  return inner  
  print(func1()())

##### 简述生成器，迭代器，装饰器以及应用场景

[参考链接](https://blog.csdn.net/weixin_39387409/article/details/85089652)

1. 迭代器对象实现了iter()方法
2. 迭代器实现了iter()和next()方法，迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束
3. 生成器是迭代器的一种，一个函数调用时返回一个迭代器，这个函数就叫生成器。通常带有yield
4. 装饰器是一个以函数作为参数，并返回一个替换函数的可执行函数，是闭包的一种应用。通常用来给一个函数添加功能

##### 使用生成器编写一个函数实现生成指定个数的斐波那契数列

def fib2(imax):  
 t,a,b=0,0,1  
 while t<imax:  
 yield b  
 a,b=b,a+b  
 t+=1  
   
for i in fib2(10):  
 print(i)

##### 一行代码通过filter和lambda函数输出alist=[1,22,2,33,23,32]中索引为奇数的值

alist=[1,22,2,33,23,32]  
ss=[x[1] for x in filter(lambda x:x[0]%2==1,enumerate(alist))]  
print(ss)

##### 编写一个函数实现十进制转62进制，分别用0-9A-Za-z,表示62位字母

import string  
print(string.ascii\_lowercase) # 小写字母  
print(string.ascii\_uppercase) # 大写字母  
print(string.digits) # 0-9  
  
s=string.digits+string.ascii\_uppercase+string.ascii\_lowercase  
def \_10\_to\_62(num):  
 ss=''  
 while True:  
 ss=s[num%62]+ss  
 if num//62==0:  
 break  
 num=num//62  
 return ss  
print(\_10\_to\_62(65))

##### 实现一个装饰器，限制该函数被调用的频率，如10秒一次

import time  
from functools import wraps  
def dec(func):  
 key=func.\_\_name\_\_  
 cache={key:None}  
 @wraps(func)  
 def inner(\*args,\*\*kwargs):  
 result=None  
 if cache.get(key) is None:  
 cache[key]=time.time()  
 result=func(\*args,\*\*kwargs)  
 print('执行函数中')  
 else:  
 now=time.time()  
 if now-cache[key]>10:  
 cache[key]=now  
 result=func(\*args,\*\*kwargs)  
 print('执行函数中')  
 else:  
 print('函数执行受限')  
 return result  
 return inner  
   
@dec  
def add(x,y):  
 print(x+y)  
   
add(1,2)  
add(1,3)  
time.sleep(10)  
add(3,4)

##### 实现一个装饰器，通过一次调用，使函数重复执行5次

from functools import wraps  
def dec(func):  
 @wraps(func)  
 def inner(\*args,\*\*kwargs):  
 result=[func(\*args,\*\*kwargs) for i in range(5)]  
 return result  
 return inner  
   
@dec  
def add(x,y):  
 return x+y  
print(add(1,2))

##### 生成器与函数的区别？

* 生成器和函数的主要区别在于函数 return a value，生成器 yield a value同时标记或记忆point of the yield 以便于在下次调用时从标记点恢复执行。 yield 使函数转换成生成器，而生成器反过来又返回迭代器。 ```python # 简单实现生成器 def dec(): n=0 for i in range(10): yield n n+=i

for i in dec(): print(i) ```

##### 列表推导式[i for i in range(10)]和生成式表达式(i for i in range(10))的区别

[参考文章](https://blog.csdn.net/qq_36523839/article/details/79807866)

1. 列表推导式的结果是一个列表。
2. 生成器表达式的结果是一个生成器，它和列表推导式类似，它一次处理一个对象，而不是一口气处理和构造整个数据结构，可以节约内存。

##### python如何定义函数时如何书写可变参数和关键字参数？

def func(a,\*args,b=1,\*\*kwargs):  
 pass

##### python中enumerate的意思是什么？

* 枚举的意思，同时得到可迭代对象，如列表和元组的索引和值，以元组形式返回

##### 描述以下dict的items和iteritems的区别

* python3中没有iteritems
* items和iteritems大致相同，只是items返回的是一个列表，iteritems返回的是一个迭代器。

##### 是否使用过functools中的函数？他的作用是什么？

1. functools.wraps()
   * 在装饰器中用过，如果不使用wraps，则原始函数的\_\_name\_\_和\_\_doc\_\_的值就会丢失
2. functools.reduce()
   * 第一个参数是一个函数，第二个参数是一个可迭代对象，代码如下：
   * # 下面代码相当于从1加到9  
     from functools import reduce  
     a=reduce(lambda x,y:x+y,range(10))  
     print(a)

##### 如何判断一个值是方法还是函数？

[参考链接](https://blog.csdn.net/amoscn/article/details/77074403) 1. 使用type()来判断，如果是method为方法，如果是function则是函数。 2. 与类和实例无绑定关系的function都属于函数（function） 3. 与类和实例有绑定关系的function都属于方法

##### 请编写一个函数将ip地址转换成一个整数。如10.3.9.12转换成00001010 00000011 00001001 00001100，然后转换成整数

def ip2int(ip):  
 nums=ip.split('.')  
 # zfill()函数是补0  
 to\_bin=[bin(int(i))[2:].zfill(8) for i in nums]  
 return int(''.join(to\_bin),2)  
i=ip2int('127.0.0.1')  
print(i)

##### lambda表达式格式以及应用场景？

* 格式：lambda 参数列表 : 返回表达式
* 应用场景：常见的在filter，reduce以及map中使用。

##### pass的使用

* 通常用来标记一个还未写的代码的位置，pass不做任何事情，一般用来做占位语句，保持程序结构的完整性

##### \*arg和\*\*kwargs的作用 \* 用来接收不确定个数的参数，\*args通常用来接收不确定个数的非关键字参数，而\*\*kwargs通常用来接收不确定个数的关键字参数

##### 如何在函数中设置一个全局变量？

* 在函数中使用global关键字定义变量

##### 求以下代码结果：

def num():  
 return [lambda x:i\*x for i in range(4)]  
print([m(2) for m in num()])

* 答案：[6,6,6,6]

##### yield from 和 yield 的区别

[简述yield和yield from](https://blog.csdn.net/lamusique/article/details/85845225)

# 下面a()和b()是等价的  
def a():  
 yield from [1,2,3,4,5]  
def b():  
 for i in [1,2,3,4,5]:  
 yield i  
for i in a():  
 print(i)  
for i in b():  
 print(i)

* yield将一个函数变成一个生成器
* yield 返回一个值
* yield from后面接可迭代对象，一个一个返回值。

##### 求以下代码的输出结果

collapse=True  
processFunc=collapse and (lambda s:' '.join(s.split())) or (lambda s:s)  
print(processFunc('i\tam\ntest\tproject!'))  
  
collapse=False  
processFunc=collapse and (lambda s:' '.join(s.split())) or (lambda s:s)  
print(processFunc('i\tam\ntest\tproject!'))

* 答案： >i am test project! >i am >test project!

##### 编写一个函数，找出数组中没有重复的值的和

def func(lis):  
 lis1=[]  
 del\_lis=[]  
 for i in lis:  
 if i not in lis1:  
 if i not in del\_lis:  
 lis1.append(i)  
 else:  
 del\_lis.append(i)  
 lis1.remove(i)  
 return sum(lis1)  
   
def func2(lis):  
 return sum([i for i in set(lis) if lis.count(i)==1])  
   
print(func2([3,4,1,2,5,6,6,5,4,3,3]))

##### 下面代码的执行结果是

a=1  
def bar():  
 a+=3  
   
bar()  
print(a)

* 答案：运行出错

##### 写一个函数，计算出以下字母所代表的数字，每个字母值不一样

for A in range(1,10):  
 for B in range(10):  
 if A==B:  
 continue  
 for C in range(1,10):  
 if C in [A,B]:  
 continue  
 for D in range(10):  
 if D in [A,B,C]:  
 continue  
 for E in range(1,10):  
 if E in [A,B,C,D]:  
 continue  
 for F in range(10):  
 if F in [A,B,C,D,E]:  
 continue  
 for G in range(1,10):  
 if G in [A,B,C,D,E,F]:  
 continue  
 for H in range(10):  
 if H in [A,B,C,D,E,F,G]:  
 continue  
 for P in range(1,10):  
 if P in [A,B,C,D,E,F,G,H]:  
 continue  
 if (A\*10+B)-(C\*10+D)==(E\*10+F) and (E\*10+F)+(G\*10+H)==(P\*100+P\*10+P):  
 print(A,B,C,D,E,F,G,H,P)

#### 写出如下代码的输出结果

[参考链接](https://www.cnblogs.com/z-x-y/p/9157238.html)

def decorator\_a(func):  
 print('Get in decorator\_a')  
 def inner\_a(\*args, \*\*kwargs):  
 print('Get in inner\_a')  
 return func(\*args, \*\*kwargs)  
 return inner\_a  
   
def decorator\_b(func):  
 print('Get in decorator\_b')  
 def inner\_b(\*args, \*\*kwargs):  
 print('Get in inner\_b')  
 return func(\*args, \*\*kwargs)  
 return inner\_b  
   
@decorator\_b #f=decorator\_b(f)  
@decorator\_a #f=decorator\_a(f)  
def f(x):  
 print('Get in f')  
 return x \* 2  
f(1)

* 答案 >Get in decorator\_a >Get in decorator\_b >Get in inner\_b >Get in inner\_a >Get in f
* 解释 >当我们对f传入参数1进行调用时，inner\_b被调用了，他会先打印Get in inner\_b,然后在inner\_b内部调用了inner\_a,所以会再打印Get in inner\_a,然后再inner\_a内部调用原来的f,并且将结果作为最终的返回总结：装饰器函数在被装饰函数定义好后立即执行从下往上执行函数调用时从上到下执行

##### 写出以下代码的输出结果：

def test():  
 try:  
 raise ValueError('something wrong')  
 except ValueError as e:  
 print('error occured')  
 return  
 finally:  
 print('ok')  
test()

* 结果(finally无论怎样都会执行) >error occured >ok

##### 求出以下代码的输出结果

mydict={'a':1,'b':2}  
def func(d):  
 d['a']=0  
 return d  
   
func(mydict)  
mydict['c']=2  
print(mydict)

* 结果 >{'a': 0, 'b': 2, 'c': 2}

##### 写个函数接收一个文件夹名称作为参数，显示文件夹中文件的路径，以及其中包含的文件夹中文件的如今

# 方法一  
import os  
def Test1(rootDir):  
 list\_dirs = os.walk(rootDir)  
 for root, dirs, files in list\_dirs:  
 for d in dirs:  
 print(os.path.join(root, d))  
 for f in files:  
 print(os.path.join(root, f))  
Test1(r'C:\Users\felix\Desktop\aaa')  
print('###################')  
# 方法二  
import os  
def Test2(rootDir):  
 paths=os.listdir(rootDir)  
 for lis in paths:  
 path=os.path.join(rootDir,lis)  
 print(path)  
 if os.path.isdir(path):  
 Test2(path)  
Test2(r'C:\Users\felix\Desktop\aaa')

##### re的match和search的区别

* match()函数是在string的开始位置匹配，如果不匹配，则返回None
* 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，

##### 什么是正则的贪婪匹配？贪婪模式和非贪婪模式的区别？

[参考文档](https://www.cnblogs.com/ILoke-Yang/p/8060003.html)

* 贪婪匹配:正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。
* 非贪婪匹配：就是匹配到结果就好，就少的匹配字符。
* 区别：默认是贪婪模式；在量词后面直接加上一个问号？就是非贪婪模式。

##### 如何使用python删除一个文件或者文件夹？

import os  
import shutil  
os.remove(path) # 删除文件  
os.removedirs(path) # 删除空文件夹  
shutil.rmtree(path) # 删除文件夹，可以为空也可以不为空

##### logging模块的作用以及应用场景

* logging模块定义的函数和类为应用程序和库的开发实现了一个灵活的事件日志系统。
* 记录日志

##### json序列化时可以处理的数据类型有哪些？如何定制支持datetime类型？序列化时，遇到中文转成unicode，如何保持中文形式？

1. 可以处理的数据类型是 string、int、list、tuple、dict、bool、null
2. 通过自定义时间序列化转换器

* import json  
  from json import JSONEncoder  
  from datetime import datetime  
  class ComplexEncoder(JSONEncoder):  
  def default(self, obj):  
   if isinstance(obj, datetime):  
   return obj.strftime(‘%Y-%m-%d %H:%M:%S‘)  
   else:  
   return super(ComplexEncoder,self).default(obj)  
  d = { ‘name‘:‘alex‘,‘data‘:datetime.now()}  
  print(json.dumps(d,cls=ComplexEncoder))  
  # {"name": "alex", "data": "2018-05-18 19:52:05"}

1. 使用ensure\_ascii=False参数

##### 写出邮箱的正则表达式

import re  
pp=re.compile('[a-zA-Z0-9\_-]+@[0-9A-Za-z]+(\.[0-9a-zA-Z]+)+')  
if pp.match('1403179190@qq.com'):  
 print('ok')

##### 写python爬虫分别用到了哪些模块？分别有什么用？

* 模块
  + request，发起请求
  + pyquery，解析html数据
  + beautifulsoup，解析html数据
  + aiohttp，异步发送请求
* 框架
  + pyspider，web界面的爬虫框架
  + scrapy，爬虫框架
  + selenium，模拟浏览器的爬虫框架

##### sys.path.append('xxx')的作用

* 添加搜索路径

##### 输入某年某月某日，判断这是这一年的第几天？

date=input('请输入某年某月某日，格式：xxxx.xx.xx')  
def get\_day(date):  
 days1=[31,28,31,30,31,30,31,31,30,31,30,31]  
 days2=[31,29,31,30,31,30,31,31,30,31,30,31]  
 year,month,day = [int(i) for i in date.split('.')]  
 if year % 400 ==0 or (year % 4==0 and year % 100!=0):  
 days=days2  
 else:  
 days=days1  
 return sum(days[:month-1])+day  
print(get\_day(date))

##### 简述面向对象的三大特性？

[参考链接](https://www.cnblogs.com/lfpython/p/7346385.html) \* 继承，封装和多态 \* 继承： \* 继承就是继承的类直接拥有被继承类的属性而不需要在自己的类体中重新再写一遍，其中被继承的类叫做父类、基类，继承的类叫做派生类、子类。 \* 封装： \* 封装就是把类中的属性和方法定义为私有的，方法就是在属性名或方法名前加双下划线，而一旦这样定义了属性或方法名后，python会自动将其转换为\_类名\_\_属性名（方法名）的格式，在类的内部调用还是用双下划线加属性名或方法名，在类的外部调用就要用\_类名\_\_属性名（方法名）。父类的私有属性和方法，子类无法对其进行修改。 \* 多态： \* 多态就是不同的对象可以调用相同的方法然后得到不同的结果，有点类似接口类的感觉，在python中处处体现着多态，比如不管你是列表还是字符串还是数字都可以使用+和\*。

##### 什么是鸭子模型？

* 鸭子类型（英语：duck typing）是动态类型的一种风格。在这种风格中，一个对象有效的语义，不是由继承自特定的类或实现特定的接口，而是由当前方法和属性的集合决定。

##### super的作用

* 当子类中的方法与父类中的方法重名时，子类中的方法会覆盖父类中的方法，那么，如果我们想实现同时调用父类和子类中的同名方法，就需要使用到super()这个函数，用法为super().函数名()

##### mro是什么？

* 对于支持继承的编程语言来说，其方法（属性）可能定义在当前类，也可能来自于基类，所以在方法调用时就需要对当前类和基类进行搜索以确定方法所在的位置。而搜索的顺序就是所谓的「方法解析顺序」（Method Resolution Order，或MRO）。

##### 什么是c3算法？

* c3算法是python新式类中用来产生mro顺序的一套算法。即多继承的查找规则。

##### 列举面向对象中带双下划线的特殊方法

\_\_new\_\_：可以调用其它类的构造方法或者直接返回别的对象来作为本类的实例。 \_\_init**： 负责类的实例化 \_\_call**：对象后边加括号，触发执行 \_\_str**：print打印一个对象时。 \_\_doc**：类的注释，该属性是无法继承的。 \_\_getattr**：在使用调用属性（方式、属性）不存在的时候触发 \_\_setattr**：添加/修改属性会触发它的执行 \_\_dellattr**：删除属性的时候会触发 \_\_delete**：采用del删除属性时，触发

##### 双下划线和单下划线的区别

* "单下划线" 开始的成员变量叫做保护变量，意思是只有类对象和子类对象自己能访问到这些变量；
* "双下划线" 开始的是私有成员，意思是只有类对象自己能访问，连子类对象也不能访问到这个数据。

##### 实例变量和类变量的区别

* 实例变量是对于每个实例都独有的数据
* 类变量是该类所有实例共享的属性和方法

##### 实例方法、静态方法和类方法的区别

* 实例方法：第一个参数必须是实例对象，通常为self。实例方法只能由实例对象调用。
* 类方法：使用装饰器@classmethod。第一个参数为当前类的对象，通常为cls。实例对象和类对象都可以调用类方法。
* 静态方法：使用装饰器@staticmethod。没有self和cls参数。方法体中不能使用类或者实例的任何属性和方法。实例对象和类对象都可以调用。

##### isinstance和type的作用

* 两者都用来判断对象的类型
* 对于一个类的之类对象的类型判断，type就不行了，而isinstance可以。 ```pyyhon class A(object): pass class B(A): pass

ba=B() ab=A() print(type(ba)==A) # False print(type(ab)==A) # True print(isinstance(ab,A)) # True print(isinstance(ba,A)) # True ```

##### 使用with语句的好处是什么

* 使用with后不管with中的代码出现什么错误，都会进行对当前对象进行清理工作。例如file的file.close()方法，无论with中出现任何错误，都会执行file.close（）方法
* 只有支持上下文管理器的对象才能使用with，即在对象内实现了两个方法：**enter**()和\_\_exit\_\_()

##### 写一个的支持with语句的类

[参考链接](https://www.cnblogs.com/yidashi110/p/10091991.html)

class W(object):  
 def \_\_init\_\_(self):  
 pass  
 def \_\_enter\_\_(self):  
 print('进入with语句')  
 return self  
   
 def \_\_exit\_\_(self,\*args,\*\*kwargs):  
 print('退出with语句')  
   
with W() as w:  
 print('之前')  
 print(w)  
 print('之后')

##### 实现一个单例模式。(尽可能多的方法)

[参考链接](https://www.cnblogs.com/huchong/p/8244279.html)

# 方法一：使用\_\_new\_\_()  
import threading  
class Singleton(object):  
 \_instance\_lock = threading.Lock()  
 def \_\_init\_\_(self):  
 pass  
   
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if not hasattr(Singleton, "\_instance"):  
 with Singleton.\_instance\_lock:  
 if not hasattr(Singleton, "\_instance"):  
 Singleton.\_instance = object.\_\_new\_\_(cls)  
 return Singleton.\_instance  
   
obj1 = Singleton()  
obj2 = Singleton()  
print(obj1 is obj2)  
  
# 方法二：使用元类来创建  
import threading  
  
class SingletonType(type):  
 \_instance\_lock = threading.Lock()  
 def \_\_call\_\_(cls, \*args, \*\*kwargs):  
 if not hasattr(cls, "\_instance"):  
 with SingletonType.\_instance\_lock:  
 if not hasattr(cls, "\_instance"):  
 cls.\_instance = super().\_\_call\_\_(\*args, \*\*kwargs)  
 return cls.\_instance  
   
class Singleton(metaclass=SingletonType):  
 def \_\_init\_\_(self):  
 pass  
   
obj1 = Singleton()  
obj2 = Singleton()  
print(obj1 is obj2)

##### 如何判断一个对象是否可调用？哪些对象是可调用对象？如何定义一个类，使其对象本身就是可调用对象？

* 使用callable函数判断。
* 可调用对象有7类：
  + 用户自定义函数
  + 内置函数
  + 内置方法
  + 方法(定义在类中的函数)
  + 类
  + 类实例(如果类中定义了\_\_call\_\_方法，那么这个类的实例就是可调用对象)
  + 生成器函数
* 在类中定义\_\_call\_\_方法，实例对象加()是即调用\_\_call\_\_的方法

##### 手写一个栈

#给一个点，我们能够根据这个点知道一些内容  
class Node(object):  
 def \_\_init\_\_(self,val): #定位的点的值和一个指向  
 self.val=val #指向元素的值,原队列第二元素  
 self.next=None #指向的指针  
   
class stack(object):  
 def \_\_init\_\_(self):  
 self.top=None #初始化最开始的位置  
   
 def push(self,n):#添加到栈中  
 n=Node(n) #实例化节点  
 n.next=self.top #顶端元素传值给一个指针  
 self.top=n  
 return n.val  
   
 def pop(self): #退出栈  
 if self.top == None:  
 return None  
 else:  
 tmp=self.top.val  
 self.top=self.top.next #下移一位，进行  
 return tmp  
   
if \_\_name\_\_=="\_\_main\_\_":  
 s=stack()  
 print(s.pop())  
 s.push(1)  
 print(s.pop())  
 s.push(2)  
 s.push(3)  
 print(s.pop())  
 s.push(3)  
 s.push(3)  
 s.push(3)  
 print(s.pop())  
 print(s.pop())  
 print(s.pop())  
 print(s.pop())

##### 使用两个队列实现一个栈

class Stack(object):  
 def \_\_init\_\_(self):  
 self.queueA=[]  
 self.queueB=[]  
 def push(self,node):  
 self.queueA.append(node)  
 def pop(self):  
 if len(self.queueA)==0:  
 return None  
 while len(self.queueA)!=1:  
 self.queueB.append(self.queueA.pop(0))  
 self.queueA,self.queueB=self.queueB,self.queueA  
 return self.queueB.pop()  
   
st=Stack()  
print(st.pop())  
st.push(1)  
print(st.pop())  
st.push(1)  
st.push(1)  
st.push(1)  
print(st.pop())  
print(st.pop())  
print(st.pop())

* 注意上面两个栈的实现方法，第一种的效率高，队列的这种方法效率低

##### 有如下链表类，请实现单链表逆置。

class ListNode:  
 def \_\_init\_\_(self,val):  
 self.val=val  
 self.next=None  
   
class Solution:  
 def reverseList(self,pHead):  
 if not pHead or not pHead.next:  
 return pHead  
 last=None  
 while pHead:  
 tmp=pHead.next  
 pHead.next=last  
 last=pHead  
 pHead=tmp  
 return last

##### 类的加载和实例化过程

1. 在堆内存中生成class对象, 把静态变量和静态方法加载到方法区, 这个堆内存中的class对象是方法区数据的入口
2. 静态变量默认初始化
3. 静态变量显式初始化
4. 执行静态代码块
5. 成员变量默认初始化, 显示初始化
6. 执行构造函数

##### 手写一个队列

class Queue(object):  
 def \_\_init\_\_(self,size):  
 self.queue=[]  
 self.size=size  
 def is\_empty(self):  
 return not bool(len(self.queue))  
 def is\_full(self):  
 return len(self.queue)==self.size  
 def enqueue(self,val):  
 if not self.is\_full():  
 self.queue.insert(0,val)  
 return True  
 return False  
 def dequeue(self):  
 if not self.is\_empty():  
 return self.queue.pop()  
 return None  
s=Queue(2)  
print(s.is\_empty)  
s.enqueue(1)  
s.enqueue(2)  
print(s.is\_full())  
print(s.dequeue())  
print(s.dequeue())  
print(s.is\_empty())

##### python的底层网络交互模块有哪些

* socket，urllib，requests，pycurl

##### 简述OSI七层协议

[参考链接](https://zhidao.baidu.com/question/568531489.html) \* 为了实现计算机系统的互连，OSI参考模型把整个网络的通信功能划分为7个层次，同时也定义了层次之间的相互关系以及各层所包括的服务及每层的功能。OSI的七层由低到高依次为：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层，下三层（物理层、数据链路层、网络层）面向数据通信，而上三层（会话层、表示层、应用层）则面向资源子网，而传输层则是七层中最为重要的一层。它位于上层和下层中间，起承上启下的作用。

##### 什么是C/S和B/S架构

[参考链接](https://www.cnblogs.com/1130136248wlxk/p/5192043.html) \* C/S 架构是一种典型的两层架构，其全称是Client/Server，即客户端服务器端架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是Socket服务器端，服务器端的程序通过Socket与客户端的程序通信。 \* B/S架构的全称为Browser/Server，即浏览器/服务器结构。Browser指的是Web浏览器，极少数事务逻辑在前端实现，但主要事务逻辑在服务器端实现，Browser客户端，WebApp服务器端和DB端构成所谓的三层架构。B/S架构的系统无须特别安装，只有Web浏览器即可。

##### 简述TCP三次握手，四次挥手的流程。

[参考链接](https://blog.csdn.net/qq_38950316/article/details/81087809)

* 三次握手
  + 第一次握手：客户端的应用进程主动打开，并向客户端发出请求报文段。其首部中：SYN=1,seq=x。
  + 第二次握手：服务器应用进程被动打开。若同意客户端的请求，则发回确认报文，其首部中：SYN=1,ACK=1,ack=x+1,seq=y。
  + 第三次握手：客户端收到确认报文之后，通知上层应用进程连接已建立，并向服务器发出确认报文，其首部：ACK=1,ack=y+1。当服务器收到客户端的确认报文之后，也通知其上层应用进程连接已建立。
* 四次挥手
  + 第一次挥手：数据传输结束以后，客户端的应用进程发出连接释放报文段，并停止发送数据，其首部：FIN=1,seq=u。
  + 服务器端收到连接释放报文段之后，发出确认报文，其首部：ack=u+1,seq=v。此时本次连接就进入了半关闭状态，客户端不再向服务器发送数据。而服务器端仍会继续发送。
  + 第三次挥手：若服务器已经没有要向客户端发送的数据，其应用进程就通知服务器释放TCP连接。这个阶段服务器所发出的最后一个报文的首部应为：FIN=1,ACK=1,seq=w,ack=u+1。
  + 第四次挥手：客户端收到连接释放报文段之后，必须发出确认：ACK=1,seq=u+1,ack=w+1。 再经过2MSL(最长报文端寿命)后，本次TCP连接真正结束，通信双方完成了他们的告别。

##### 如果已经建立了TCP连接，但是客户端突然出现故障了怎么办

* TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

##### 什么是arp协议

* ARP全称“Address Resolution Protocol”，地址解析协议。
* 实现局域网内通过IP地址获取主机的MAC地址。
* MAC地址48位主机的物理地址，局域网内唯一。
* ARP协议类似DNS服务，但不需要配置服务。
* ARP协议是三层协议。

##### TCP和UDP的区别

1. TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接
2. TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付
3. UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。
4. 每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信
5. TCP对系统资源要求较多，UDP对系统资源要求较少。

##### 为什么基于tcp协议的通信比基于udp协议的通信更可靠

* TCP是面向连接的传输协议，每次都需要建立一个可以相互信任的连接，中间有个三次握手过程。而UDP是面向无连接的传输协议，不需要建立安全的连接，

##### 什么是局域网和广域网

* 局域网（Local Area Network），简称LAN，是指在某一区域内由多台计算机互联成的计算机组。“某一区域”指的是同一办公室、同一建筑物、同一公司和同一学校等，一般是方圆几千米以内。局域网可以实现文件管理、应用软件共享、打印机共享、扫描仪共享、工作组内的日程安排、电子邮件和传真通信服务等功能。局域网是封闭型的，可以由办公室内的两台计算机组成，也可以由一个公司内的上千台计算机组成。
* 广域网（Wide Area Network），简称WAN，是一种跨越大的、地域性的计算机网络的集合。通常跨越省、市，甚至一个国家。广域网包括大大小小不同的子网，子网可以是局域网，也可以是小型的广域网。
* 两者区别：
  + 范围不同，广域网比局域网广
  + 接口类型不同
  + 速率不同
  + 协议不同

##### 什么是socket？简述基于tcp协议的socket通信流程？

* socket通常也称作"套接字"，用于描述IP地址和端口，是一个通信链的句柄。
* 通信流程：
  1. 服务端创建一个ServerSocket对象,指定端口号,ServerSocket对象等待客户端的连接请求。
  2. 客户端创建一个Socket对象,指定主机地址和端口号,向服务端发出连接请求。
  3. 服务端接收到客户端的连接请求,建立一条TCP连接,再创建一个Socket对象与客户端的Socket对象进行通信。
  4. 服务端和客户端分别创建字节输入流和字节输出流,通过字节输入流获得对方发来的数据,通过字节输出流向对方发送数据。
  5. 当一方决定结束通信时,向对方发送结束信息;另一方接收到结束信息后,双方分别关闭各自的TCP连接。
  6. ServerSocket对象停止等待客户端的连接请求。

##### 什么是粘包？出现粘包的原因？

* 粘包：多个数据包被连续存储于连续的缓存中，在对数据包进行读取时由于无法确定发生方的发送边界，而采用某一估测值大小来进行数据读出，若双方的size不一致时就会使指发送方发送的若干包数据到接收方接收时粘成一包，从接收缓冲区看，后一包数据的头紧接着前一包数据的尾。
* 出现粘包现象的原因是多方面的，它既可能由发送方造成，也可能由接收方造成。
  + 发送方引起的粘包是由TCP协议本身造成的，TCP为提高传输效率，发送方往往要收集到足够多的数据后才发送一包数据。若连续几次发送的数据都很少，通常TCP会根据优化算法把这些数据合成一包后一次发送出去，这样接收方就收到了粘包数据。
  + 接收方引起的粘包是由于接收方用户进程不及时接收数据，从而导致粘包现象。这是因为接收方先把收到的数据放在系统接收缓冲区，用户进程从该缓冲区取数据，若下一包数据到达时前一包数据尚未被用户进程取走，则下一包数据放到系统接收缓冲区时就接到前一包数据之后，而用户进程根据预先设定的缓冲区大小从系统接收缓冲区取数据，这样就一次取到了多包数据。

##### 发生粘包现象如何处理？

1. 对于发送方引起的粘包现象，用户可通过编程设置来避免，TCP提供了强制数据立即传送的操作指令push，TCP软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；
2. 对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；
3. 由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。

##### IO多路复用的作用？

* I/O多路复用是用于提升效率，单个进程可以同时监听多个网络连接IO。
* 与多进程和多线程技术相比，I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

##### 什么是防火墙？防火墙的作用是什么？

* 在互联网上防火墙是一种非常有效的网络安全模型，通过它可以隔离风险区域(即Internet或有一定风险的网络)与安全区域(局域网)的连接，同时不会妨碍人们对风险区域的访问。所以它一般连接在核心交换机与外网之间。
* 防火墙的作用：
  1. 过滤进出网络的数据
  2. 管理进出访问网络的行为
  3. 封堵某些禁止业务
  4. 记录通过防火墙信息内容和活动
  5. 对网络攻击检测和告警

##### select、poll、epoll模型的区别

[参考链接](https://www.cnblogs.com/jeakeven/p/5435916.html)

1. 支持一个进程所能打开的最大连接数
   * select的最大连接数大概32*32，或者32*64
   * poll本质和select没区别，但是它没有最大连接数限制
   * epoll大概10万左右(1G的机器)
2. FD剧增后带来的IO效率问题
   * select和poll每次调用都会对连接进行线性遍历，所以会随着FD的增加会造成遍历速度慢的“线性下降性能问题”
   * epoll没有前两个的线性下降的性能问题，但是当socket都很活跃的情况下，可能会有性能问题。
3. 消息传递方式
   * select和poll内核需要将消息传递到用户空间，都需要内核拷贝动作。
   * epoll通过内核和用户空间共享一块内存来实现

##### 简述进程，线程，协程的区别以及应用场景？

* 区别：
  1. 线程是资源分配的单位
  2. 线程是操作系统调度的单位
  3. 进程切换需要的资源很大，效率很低
  4. 线程切换需要的资源一般，效率一般(在不考虑GIL的情况下
  5. 协程切换任务资源很小，效率高
  6. 多进程，多线程根据cpu核数不一样可能是并行的，但是协程是在一个线程中，所以是并发。)
* 应用场景
  1. 协程：当程序中存在大量不需要cpu的操作时，适用协程
  2. 计算密集型，用进程。IO密集型，用线程。

##### 什么时GIL锁

* 即全局解释器锁，
* 一个时间点只有一个线程处于执行状态。

##### python中如何使用进程池和线程池

from concurrent.futures import ThreadPoolExecutor,ProcessPoolExecutor  
import os,time,random  
from multiprocessing import Pool  
  
def task(n):  
 print('%s is runing' %os.getpid())  
 time.sleep(random.randint(1,3))  
 return n\*\*2  
   
if \_\_name\_\_ == '\_\_main\_\_':  
 # 多进程方式一  
 pool2=Pool()  
 pool2.map(task,range(10))  
   
 # 多进程方式二，下面这种多进程和多线程的用法一模一样  
 executor=ThreadPoolExecutor(max\_workers=3)  
 futures=[]  
 for i in range(11):  
 future=executor.submit(task,i)  
 futures.append(future)  
 executor.shutdown(True)  
 print('+++>')  
 for future in futures:  
 print(future.result())

##### threading.local的作用

* 为每个线程创建一个独立的空间，使得线程对自己的空间中的数据进行操作(数据隔离)。

##### 进程之间如何进行通信？

1. 共享内存
   * 通过mmap模块实现
2. 信号
3. 通过Queue队列
4. 通过Pipe管道
5. 通过socket

##### 什么是并发和并行

* 并发:指应用能够交替执行不同的任务,其实并发有点类似于多线程的原理,多线程并非是同时执行多个任务,如果你开两个线程执行,就是在你几乎不可能察觉到的速度不断去切换这两个任务,已达到"同时执行效果",其实并不是的,只是计算机的速度太快,我们无法察觉到而已.
* 并行:指应用能够同时执行不同的任务,
* 并发是多个事件在同一时间段执行，并行是多个事件在统一时间点执行。

##### 解释什么是异步非阻塞

* 同步和异步：
  + 同步：就是在发出一个功能调用时，在没有得到结果之前，该调用就不返回。
  + 异步：当一个异步过程调用发出后，调用者不会立刻得到结果。实际处理这个调用的部件是在调用发出后，通过状态、通知来通知调用者，或通过回调函数处理这个调用。
* 阻塞和非阻塞
  + 阻塞：阻塞调用是指调用结果返回之前，当前线程会被挂起。函数只有在得到结果之后才会返回。
  + 非阻塞：指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。
* 阻塞，非阻塞：进程/线程要访问的数据是否就绪，进程/线程是否需要等待；
* 同步，异步：访问数据的方式，同步需要主动读写数据，在读写数据的过程中还是会阻塞；异步只需要I/O操作完成的通知，并不主动读写数据，由操作系统内核完成数据的读写。

##### 路由器和交换机的区别

[参考链接](https://www.kanzhun.com/k-mianshiwenti/1466326.html)

* 交换机是一种用于电信号转发的网络设备。路由器是链接因特网中各局域网和广域网的设备。
* 区别
  1. 交换机工作在第二层，数据链路层，路由器工作在第三层，网络层。
  2. 路由器提供防火墙服务。
  3. 传统交换机只能风格冲突域，不能分割广播域，二路由器可以分割广播域。

##### 什么是域名解析

* 域名解析是把域名指向网站空间IP，让人们通过注册的域名可以方便地访问到网站一种服务。IP地址是网络上标识站点的数字地址，为方便记忆，采用域名来代替IP地址标识站点地址。域名解析就是域名到IP地址的转换过程。

##### 如何修改本地hosts文件

* 进入c:32

##### 生产者消费者模型的应用场景

* 说明
  + 生产者只在仓库未满时进行生产，仓库满时生产者进程被阻塞；消费者只在仓库非空时进行消费，仓库为空时消费者进程被阻塞；
* 应用场景：处理数据比较消耗时间，线程独占，生产数据不需要即时的反馈等。比如说写入日志，将多线程产生的日志放在队列中，然后写入。

##### 什么是cdn

* cdn全称是内容分发网络。其目的是让用户能够更快速的得到请求的数据。
* cdn就是用来加速的，他能让用户就近访问数据，这样就更更快的获取到需要的数据。

##### traceroute使用哪种网络协议

traceroute (Windows 系统下是tracert) 命令利用ICMP 协议定位您的计算机和目标计算机之间的所有路由器。

##### 守护线程，守护进程是什么

* 主进程创建守护进程
  1. 守护进程会在主进程代码运行结束的情况下，立即挂掉。
  2. 守护进程本身就是一个子进程。
  3. 主进程在其代码结束后就已经算运行完毕了（守护进程在此时就被回收）,然后主进程会一直等非守护的子进程都运行完毕后回收子进程的资源(否则会产生僵尸进程)，才会结束，
* 守护线程
  1. 守护线程会在"该进程内所有非守护线程全部都运行完毕后,守护线程才会挂掉"。并不是主线程运行完毕后守护线程挂掉。这一点是和守护进程的区别之处！
  2. 守护线程守护的是：当前进程内所有的子线程！
  3. 主线程在其他非守护线程运行完毕后才算运行完毕（守护线程在此时就被回收）。因为主线程的结束意味着进程的结束，进程整体的资源都将被回收，而进程必须保证非守护线程都运行完毕后才能结束。

##### 简述多进程开发中join和deamon的区别

* join：当子线程调用join时，主线程会被阻塞，当子线程结束后，主线程才能继续执行。
* deamon：当子进程被设置为守护进程时，主进程结束，不管子进程是否执行完毕，都会随着主进程的结束而结束。

##### GIL锁对python性能的影响

* 会降低多线程的效率。可以说python就是个单线程的程序。
* 如何避免：
  + 用多进程代替多线程
  + 使用其他解释器

##### 使用yield实现一个协程

def consumer():  
 r = ''  
 while True:  
 n = yield r  
 if n is None:  
 return  
 print('[CONSUMER] Consuming %s...' % n)  
 r = '200 OK'  
  
def produce(c):  
 c.send(None)  
 n = 0  
 while n < 5:  
 n = n + 1  
 print('[PRODUCER] Producing %s...' % n)  
 r = c.send(n)  
 print('[PRODUCER] Consumer return: %s' % r)  
 c.close()  
  
c = consumer()  
produce(c)

##### 使用async语法实现一个协程

import asyncio  
import time  
  
now = lambda : time.time()  
  
async def hello():  
 print("hello")  
 await asyncio.sleep(2)  
 return "done"  
   
start = now()  
# 协程对象  
h1 = hello()  
h2 = hello()  
h3 = hello()  
  
# 创建一个事件loop  
loop = asyncio.get\_event\_loop()  
# 任务（task）对象  
tasks = [  
asyncio.ensure\_future(h1),  
asyncio.ensure\_future(h2),  
asyncio.ensure\_future(h3),  
]  
  
# 将协程加入到事件循环loop  
loop.run\_until\_complete(asyncio.wait(tasks))  
for task in tasks:  
 print(task.result())  
   
print(now()-start)

##### 简述线程死锁是怎么造成的。如何避免？

* 死锁的产生原因？
  + 系统资源的竞争
  + 进程运行推进顺序不当
* 解决死锁
  + 加锁顺序：线程按照一定的顺序加锁
  + 加锁时限：线程尝试获取锁的时候加上一定的时限，超过时限，则放弃对该锁的请求，并释放自己占有的锁。
  + 死锁检测

##### 什么是asyncio

* asyncio是并发的一种方式，是一个协程相关的库。也叫异步IO

##### 什么是gevent

* gevent是一个pythn网络框架，它为各种并发和网络相关的任务提供了整洁的API

##### 什么是twisted框架

* twisted是用python实现的基于事件驱动的网络引擎框架。

##### 什么是LVS

* LVS是linux虚拟服务器，是一个虚拟的linux集群系统。

##### 什么是Nginx

* nginx是一个高性能的HTTP和反向代理服务，也是一个IMAP/POP3/SMTP服务

##### 什么是keepalived

* Keepalived是Linux下一个轻量级别的高可用解决方案

##### 什么是haproxy？

* HAProxy是一个使用C语言编写的自由及开放源代码软件，其提供高可用性、负载均衡，以及基于TCP和HTTP的应用程序代理

##### 什么是负载均衡

* 负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。
* 负载均衡其意思就是分摊到多个操作单元上进行执行。

##### 什么是rpc

# 远程过程调用 (RPC) 是一种协议，程序可使用这种协议向网络中的另一台计算机上的程序请求服务

# 1.RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是一个服务器。

# 2.首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。

# 2.在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，

# 3.最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。

##### 什么是正向代理和反向代理？

* 正向代理
  + 正向代理类似一个跳板机，代理访问外部资源。
  + 正向代理 是一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端必须要进行一些特别的设置才能使用正向代理。
  + 正向代理作用：
    - 访问原来无法访问的资源，如google
    - 可以做缓存，加速访问资源
    - 对客户端访问授权，上网进行认证
    - 代理可以记录用户访问记录（上网行为管理），对外隐藏用户信息
* 反向代理
  + 反向代理（Reverse Proxy）实际运行方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个服务器
  + 反向代理的作用：
    - 保证内网的安全，可以使用反向代理提供WAF功能，阻止web攻击
    - 负载均衡，通过反向代理服务器来优化网站的负载

##### 列举常见的关系型数据库和非关系型数据库。

* 关系型数据库：Oracle、DB2、Microsoft SQL Server、Microsoft Access、MySQL
* 非关系型数据库：mongodb，redis，elasticsearch，NoSql

##### Mysql常见数据库引擎及区别

[参考链接](https://www.cnblogs.com/yangjinwang/p/6433515.html)

* InnoDB：用于事务处理应用程序，具有众多特性，包括ACID事务支持。(提供行级锁)
* MyISAM：默认的MySQL插件式存储引擎，它是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。注意，通过更改STORAGE\_ENGINE配置变量，能够方便地更改MySQL服务器的默认存储引擎。
* Memory：将所有数据保存再RAM中

##### 简述事务及其特性

* 事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位
* 事务具有4个特性：原子性、一致性、隔离性和持续性。
  + 原子性：事务是数据库的逻辑工作单位，事务中包括的诸操作要么都做，要么都不做。
  + 一致性：事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。
  + 隔离性：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。
  + 持续性：持续性也称永久性，指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

##### 简述触发器、函数、视图和存储过程

* 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
* 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）
* 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
* 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）

##### mysql索引种类

[参考链接](https://blog.csdn.net/liutong123987/article/details/79384395)

* 普通索引：仅加速查询
* 唯一索引：加速查询 + 列值唯一（可以有null）
* 主键索引：加速查询 + 列值唯一（不可以有null）+ 表中只有一个
* 组合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并
* 全文索引：对文本的内容进行分词，进行搜索

##### 索引再什么情况下遵循最左前缀的规则？

[参考链接](http://www.itdaan.com/blog/2018/07/09/65f5a78ed14d0e3374ba1ab8286a08af.html)

* 在多字段进行索引的时候，会遵循以上原则

##### Mysql常见的函数

[参考链接](https://blog.csdn.net/qq646040754/article/details/82721801)

* ABS(x)：返回x的绝对值
* ROUND(x)：返回参数x的四舍五入的一个整数
* TRIM(str)：去除字符串两边的空白
* COUNT()：返回值的个数
* AVG()：返回平均值
* SUM()：求和

##### 列举创建索引但是无法命中索引的情况

[参考链接](http://www.cnblogs.com/zhuifeng-mayi/p/9285724.html)

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

##### 数据库的导入与导出命令

* 导出(mysqldump)
  + 导出数据和表结构
    - mysqldump -uroot -p dbname > dbname .sql
  + 只导出表结构
    - mysqldump -uroot -p -d dbname > dbname .sql
* 导入
  + mysql -u用户名 -p密码 数据库名 < 数据库名.sql

##### 你了解哪些数据库优化方案

[参考链接](https://blog.csdn.net/zhushuai1221/article/details/51740846/)

1. 减少数据访问-创建并正确使用索引
2. 返回更少的数据
   1. 数据分页处理
   2. 只返回需要的字段
3. 减少交互次数
   1. 使用存储过程
   2. 优化业务逻辑
4. 减少服务器的cpu运算
   1. 使用绑定变量
   2. 合理使用排序
   3. 减少比较操作
   4. 大量复杂运算在客户端处理
5. 利用更多资源
   1. 客户端多进程访问
   2. 数据库并行处理

##### char和varchar的区别

1. 保存方式不一样
2. char有固定的长度，而varchar属于可变长的字符类型
3. char的效率比varchar高

##### MySQL执行计划的作用和使用方法

[参考链接](https://blog.csdn.net/heng_yan/article/details/78324176)

* 作用：用来进行查询分析，比如整个查询涉及多少防，使用哪些索引，运行时间等
* 使用方法：使用explain关键字，如explain select xxx from xxx;

##### 为什么数据很大的时候使用limit offset分页时，越往后翻速度越慢，如何优化？

* 使用limit分页时，比如limit 10000，20的意思是扫描满足条件的10020行，扔掉前面的10000行，最后返回20行，问题就出在这，当数据量大的时候，大量时间花在了扫描上面了。
* 优化方法：
  + 当一个数据库表过于庞大，LIMIT offset, length中的offset值过大，则SQL查询语句会非常缓慢，你需增加order by，并且order by字段需要建立索引。
  + 如果使用子查询去优化LIMIT的话，则子查询必须是连续的，某种意义来讲，子查询不应该有where条件，where会过滤数据，使数据失去连续性

##### 什么是索引合并

1. 索引合并是把几个索引的范围扫描合并成一个索引。
2. 索引合并的时候，会对索引进行并集，交集或者先交集再并集操作，以便合并成一个索引。
3. 这些需要合并的索引只能是一个表的。不能对多表进行索引合并。

##### 什么是覆盖索引

* 覆盖索引又可以称为索引覆盖。
  1. 解释一： 就是select的数据列只用从索引中就能够取得，不必从数据表中读取，换句话说查询列要被所使用的索引覆盖。
  2. 解释二： 索引是高效找到行的一个方法，当能通过检索索引就可以读取想要的数据，那就不需要再到数据表中读取行了。如果一个索引包含了（或覆盖了）满足查询语句中字段与条件的数据就叫做覆盖索引。
  3. 解释三： 是非聚集组合索引的一种形式，它包括在查询里的Select、Join和Where子句用到的所有列（即建立索引的字段正好是覆盖查询语句[select子句]与查询条件[Where子句]中所涉及的字段，也即，索引包含了查询正在查找的所有数据）。

##### 简述数据库的读写分离

[参考链接](https://www.cnblogs.com/0zcl/p/7141459.html)

* 读写分离就是在主服务器上修改，数据会同步到从服务器，从服务器只能提供读取数据，不能写入，实现备份的同时也实现了数据库性能的优化，以及提升了服务器安全。

##### 简述数据库分库分表

[参考链接](https://www.cnblogs.com/butterfly100/p/9034281.html)

* 垂直切分
  + 垂直分库：
    - 就是根据业务耦合性，将关联度低的不同表存储在不同的数据库
  + 垂直分表：
    - 基于数据库中的"列"进行，某个表字段较多，可以新建一张扩展表，将不经常用或字段长度较大的字段拆分出去到扩展表中。
* 横向切分：
  + 水平切分分为库内分表和分库分表，是根据表内数据内在的逻辑关系，将同一个表按不同的条件分散到多个数据库或多个表中，每个表中只包含一部分数据，从而使得单个表的数据量变小，达到分布式的效果。

##### 数据库锁的作用

[参考链接](https://blog.csdn.net/zhangzhetaojj/article/details/81559583)

* 锁分为三种：乐观锁，悲观锁和共享锁
* 数据库和操作系统一样，是一个多用户使用的共享资源。当多个用户并发地存取数据 时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。加锁是实现数据库并 发控制的一个非常重要的技术。在实际应用中经常会遇到的与锁相关的异常情况，当两个事务需要一组有冲突的锁，而不能将事务继续下去的话，就会出现死锁，严 重影响应用的正常执行。

##### MySQL的半同步复制原理

* 半同步复制，介于异步复制和全同步复制之间，主库在执行完客户端提交的事务后不是立刻返回给客户端，而是等待至少一个从库接收到并写到relay log中才返回给客户端。相对于异步复制，半同步复制牺牲了一定的性能，提高了数据的安全性。
* 异步复制，MySQL默认的复制是异步的，主库在执行完客户端提交的事务后会立即将结果返给给客户端，并不关心从库是否已经接收并处理。原理最简单，性能最好，但是主从之间数据不一致的概率很大。
* 全同步复制，指当主库执行完一个事务，所有的从库都执行了该事务才返回给客户端。因为需要等待所有从库执行完该事务才能返回，所以全同步复制的性能必然会收到严重的影响。

##### MySQL的增删改查

[参考链接](https://www.cnblogs.com/heyangblog/p/7624645.html)

* 增
  + 指定字段名
    - 语法：INSERT INTO 表名（字段名1，字段名2，…）VALUES（值1，值2，…）；
    - 举例：INSERT INTO student(id,name,grade) VALUES(1,'zhangshan',98);
  + 不指定字段名
    - 语法：INSERT INTO 表名 VALUES(值11，值2，…）；
    - 举例：INSERT INTO student VALUES (2,'lisi',62);
  + 其他写法
    - 语法：INSERT INTO 表名 SET 字段名1=值1[,字段名2=值2，…]
    - 举例：INSERT INTO student SET id=4，name='zhaoliu',grade=72;
  + 同时添加多条数据
    - 语法：INSERT INTO 表名[(字段名1，字段名2，…)] VALUES （值1，值2，…），（值1，值2，…），…（值1，值2，…）
    - 举例：INSERT INTO student VALUES (5，‘lilei’,99), (6,'hanmeimei',87), (8,'poly',76);
* 删
  + 删除部分数据
    - 语法：DELETE FROM 表名 [WHERE 条件表达式]
    - 命令：DELETE  FROM student WHERE id=7;
  + 删除全部数据
    - 语法：DELETE FROM 表名
    - 命令：DELETE FROM student；
  + 推荐的删除全部数据
    - 语法：TRUNCTE [TABLE ] 表名
    - 举例：TRUNCATE TABLE student
* 改
  + 更新部分数据
    - 语法：UPDATE 表名 SET 字段名1=值1，[ ，字段名2=值2，…] [ WHERE 条件表达式 ]
    - 命令：UPDATE student SET name=‘caocao’,grade=50 WHERE id=1;
  + 更新全部数据
    - 语法：UPDATE 表名 SET 字段名=值
    - 命令：UPDATE student SET grade=80；
* 查
  + 查询所有字段
    - 语法：SELECT 字段名1，字段名2，… FROM 表名 （该语法也可以查询部分字段）
    - 语法：SELECT \* FROM 表名；
  + 按条件查询
    - 语法：SELECT 字段名1，字段名2，… FROM 表名 WHERE 条件表达式
    - 命令：SELECT id，name FROM student2  WHERE id=4;
  + 带IN关键字的查询
    - 语法：SELECT \* | 字段名1，字段名2，… FROM 表名 WHERE 字段名 [ NOT ]  IN （元素1，元素2，…）
    - 命令：SELECT \* FROM student2 WHERE id IN （1,2,3）；
  + 带 BETWEEN AND  关键字的查询
    - 语法：SELECT \* | { 字段名1，字段名2，… } FROM  表名 WHERE 字段名 [ NOT ] BETWEEN  值1  AND  值2；
    - 命令：SELECT id,name FROM students WHERE id BETWEEN 2 AND 5;
  + 空值查询
    - 语法：SELECT \* | 字段名1，字段名2，… FROM 表名 WHERE 字段名 IS [ NOT ] NULL
    - 命令：SELECT \* FROM student2 WHERE gender IS NULL;
  + 带 DISTINCT 关键字的查询
    - 语法：SELECT DISTINCT 字段名 FROM 表名；
    - 命令：SELECT DISTINCT gender FROM student2;
  + 带 LIKE 关键字的查询
    - 语法：SELECT \* | 字段名1，字段名2，… FROM 表名 WHERE 字段名 [ NOT ] LIKE ‘匹配字符串’;
    - 注意：%表示匹配任意长度的字符串，\_表示匹配单个字符串
    - 命令：SELECT id,name FROM student2  WHERE name LIKE "S%";
    - 命令：SELECT id,name FROM student2 WHERE name LIKE 'w%g';
    - 命令：SELECT id,name FROM student2 WHERE name NOT LIKE '%y%';
    - 命令：SELECT \* FROM student2 WHERE name LIKE 'wu\_ong';
  + 带 AND 关键字的多条件查询
    - 语法：SELECT \* | 字段名1，字段名2，… FROM 表名 WHERE 条件表达式1 AND 条件表达式2 [ … AND 条件表达式 n ];
    - 命令：SELECT id,name FROM student2 WHERE id<5 AND gender='女';
  + 带 OR 关键字的多条件查询
    - 语法：SELECT \* | 字段名1，字段名2，… FROM 表名 WHERE 条件表达式1 OR 条件表达式2 [ … OR 条件表达式 n ];
    - 命令：SELECT id,name ,gender FROM student2 WHERE id<3 OR gender='女';
  + AND和OR一起使用时，AND的优先级高于OR
  + 聚合函数
    - COUNT()函数：统计记录的条数
      * 语法：SELECT COUNT(\*) FROM 表名举例：
      * 命令：SELECT COUNT(\*) FROM student2;
    - SUM()函数：求出表中某个字段所有值的总和
      * 语法：SELECT  SUM(字段名) FROM 表名；
      * 命令：SELECT SUM(grade) FROM student2;
    - AVG()函数：求出表中某个字段所有值的平均值
      * 语法：SELECT AVG(字段名) FROM 表名；
      * 命令：SELECT AVG(grade) FROM student2;
    - MAX()函数：求出表中某个字段所有值的最大值
      * 语法：SELECT MAX(字段名) FROM 表名；
      * 命令：SELECT MAX(grade) FROM student2;
    - MIN()函数：求出表中某个字段所有值的最小值
      * 语法：SELECT MIN(字段名) FROM 表名；
      * 命令：SELECT MIN(grade) FROM student2;
  + 对查询结果进行排序
    - 语法：SELECT 字段名1，字段名2，… FROM 表名 ORDER BY 字段名1 [ ASC | DESC ],字段名2 [ ASC | DESC ]
    - (升序)命令：SELECT \* FROM student2 ORDER BY grade;
    - (降序)命令：命令：SELECT \* FROM student2 ORDER BY grade DESC;
  + 分组查询
    - 语法：SELECT  字段名1，字段名2，… FROM 表名 GROUP BY 字段名1，字段名2，… [ HAVING 条件表达式 ];
    - 单独使用 GROUP BY 进行分组
      * 命令：SELECT \* FROM student2 GROUP BY gender;
    - GROUP BY 和聚合函数一起使用
      * 命令：SELECT COUNT(\*) ,gender FROM student2 GROUP BY gender;
    - GROUP BY 和 HAVING 关键字一起使用
      * 命令：SELECT sum(grade),gender FROM student2 GROUP BY gender HAVING SUM(grade) < 300;
  + 使用 LIMIT 限制查询结果的数量
    - 语法：SELECT 字段名2，字段名2，… FROM 表名 LIMIT [ OFFSET ,] 记录数
    - (从0开始的4条)命令：SELECT \* FROM student LIMIT 4;
    - (从第五条开始的4条)命令：SELECT \* FROM student2 ORDER BY grade DESC LIMIT 4,4;
  + 为表和字段取别名
    - 语法：SELECT \* FROM 表名 [ AS ] 别名；
    - 命令：SELECT \* FROM student2 AS s WHERE s.gender='女';
  + 为字段取别名
    - 语法：SELECT 字段名 [ AS ] 别名 [ ,字段名 [AS] 别名，…]  FROM 表名 ；
    - 命令：SELECT name AS stu\_name,gender AS stu\_gender FROM student2;

##### MySQL的建表语句

#创建表，例子  
#所谓的建表就是声明列的过程,所以要首先分析列  
create table member(  
 id int unsigned auto\_increment primary key,  
 username varchar(20) not null default '',  
 gender char(1) not null default '',  
 weight tinyint unsigned not null default 0,  
 birth date not null default '0000-00-00',  
 salary decimal(8,2) not null default 0.00,  
 lastlogin int unsigned not null default 0  
)engine myisam charset utf8;

##### MySQL如何创建索引

* ALTER TABLE
  + ALTER TABLE用来创建普通索引、UNIQUE索引或PRIMARY KEY索引。
    - ALTER TABLE table\_name ADD INDEX index\_name (column\_list)
    - ALTER TABLE table\_name ADD UNIQUE (column\_list)
    - ALTER TABLE table\_name ADD PRIMARY KEY (column\_list)
* CREATE INDEX
  + CREATE INDEX可对表增加普通索引或UNIQUE索引。
    - CREATE INDEX index\_name ON table\_name (column\_list)
    - CREATE UNIQUE INDEX index\_name ON table\_name (column\_list)

##### 简述SQL注入原理，以及如何在代码层面房子sql注入

通俗点讲，SQL注入的根本原因是: "用户输入数据"意外变成了代码被执行。 "用户输入数据"我这里可以指Web前端$\_POST,$\_GET获取的数据，也可以指从数据库获取的数据，当然也不排除程序猿无意中使用的特殊字符串。 在SQL语句的拼接中，一些含特殊字符的变量在拼接时破坏了SQL语句的结构，导致"用户输入数据"意外变成了代码被执行。

规避方法： 1. 理语句法  (解析协议层面上完全规避SQL注入) 2. 字符串转义（不要在sql中拼接字符） # query作为sql模板，args为将要传入的参数 execute(query, args=None)

##### 使用python将数据库的student表中的数据写入db.txt

import pymysql  
connect=pymysql.Connect(  
 host='',  
 port=,  
 user='',  
 passwd='',  
 db='',  
 charset='',  
)  
  
cursor=connect.cursor()  
sql='select \* from student'  
cursor.execute(sql)  
students=cursor.fetchall()  
  
with open('db.txt','w') as f:  
 for student in students:  
 f.write(student)  
   
cursor.close()  
connect.close()

##### 简述left join和right join的区别

* 左连接和右连接很相似，只是左右表位置的不同罢了。
* left join(左连接) 返回包括左表中的所有记录和右表中连接字段相等的记录
* right join(右连接) 返回包括右表中的所有记录和左表中连接字段相等的记录

##### 索引有什么作用，有哪些分类，有什么好处和坏处？

* 作用：为了增加查询速度，提高系统性能
* 分类：
  + 唯一索引：不允许其中任何两行具有相同索引值的索引。
  + 非唯一索引：允许其中任何两行具有相同索引值的索引。
  + 主键索引：有一列或列组合，其值唯一标识表中的每一行。
  + 聚集索引：表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。
* 好处：
  + 帮助用户提高查询速度
  + 利用索引的唯一性来控制记录的唯一性
  + 可以加速表与表之间的连接
  + 降低查询中分组和排序的时间
* 坏处：
  + 存储索引占用磁盘空间
  + 执行数据修改操作(INSERT、UPDATE、DELETE)产生索引维护

##### mysql慢日志

* MySQL的慢查询日志是MySQL提供的一种日志记录，它用来记录在MySQL中响应时间超过阀值的语句，具体指运行时间超过long\_query\_time值的SQL，则会被记录到慢查询日志中。

##### redis和memcached的区别

1. 存储方式：
   * Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。Redis有部份存在硬盘上，这样能保证数据的持久性。
2. 数据支持类型
   * Memcache对数据类型支持相对简单。Redis有复杂的数据类型。
3. 使用底层模型不同
   * 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。
4. value大小
   * redis最大可以达到1GB，而memcache只有1MB

##### 如何高效的找到redis中所有以felix开头的key

* scan 0 match felix\* count 5
* 表示从游标0开始查询felix开头的key，每次返回5条，但是这个5条不一定

##### 什么是一致性哈希

[参考链接](https://www.jianshu.com/p/49e3fbf41b9b)

* 一致性哈希简称DHT,是麻省理工学院提出的一种算法，目前主要应用于分布式缓存当中。一致性哈希可以有效地解决分布式存储结构下动态增加和删除节点所带来的问题。

##### redis是单进程单线程的吗？

* Redis采用的是基于内存的采用的是单进程单线程模型的KV数据库

##### redis默认多少个db

* 默认有16个数据库

##### 如果redis中的某个列表中的数据量非常大，如何实现循环显示每一个值？

* 使用生成器一个一个取

##### redis如何实现主从复制？以及数据同步机制？

* redis从主从结构可以采用一主多从或者级联结构，主从复制可以根据是否全量分为全量同步和增量同步。
* 全量同步：redis全量复制一般发生在slave初始化阶段，这时slave需要将master上的所有数据都复制一份步骤如下：
  + 从服务器连接主服务器，发送SYNC命令；
  + 主服务器接收到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执行的所有写命令；
  + 主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的写命令；
  + 从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；
  + 主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；
  + 从服务器完成对快照的载入，开始接收命令请求，并执行来自主服务器缓冲区的写命令；
* 增量同步：
  + Redis增量复制是指Slave初始化后开始正常工作时主服务器发生的写操作同步到从服务器的过程。 增量复制的过程主要是主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务器接收并执行收到的写命令。
* Redis主从同步策略
  + 主从刚刚连接的时候，进行全量同步；全同步结束后，进行增量同步。当然，如果有需要，slave 在任何时候都可以发起全量同步。redis 策略是，无论如何，首先会尝试进行增量同步，如不成功，要求从机进行全量同步

##### redis中sentinel的作用

[参考链接](https://blog.csdn.net/qq2430/article/details/80679439)

* Redis Sentinel是一个用来监控redis集群中节点的状态，不用来存储数据。当集群中的某个节点有故障时，可以自动的进行故障转移的操作。通常为了保证sentinel的高可用，sentinel也会部署多个

##### 如何实现redis集群

[参考链接](https://blog.csdn.net/truelove12358/article/details/79612954)

1. Twitter开发的twemproxy
2. 豌豆荚开发的codis
3. redis官方的redis-cluster

##### redis中默认有多少个哈希槽

* 2^14个
* Redis 集群没有使用一致性hash, 而是引入了哈希槽的概念。

##### 简述redis的有几种持久化策略以及比较？

* RDB 持久化可以在指定的时间间隔内生成数据集的时间点快照。
* AOF 持久化记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。 AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被追加到文件的末尾。 Redis 还可以在后台对 AOF 文件进行重写(rewrite)，使得 AOF 文件的体积不会超出保存数据集状态所需的实际大小。
* 区别：
  + RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。
  + AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。

##### 列举redis支持的过期策略

* 定时删除
  + 在设置key的过期时间的同时，为该key创建一个定时器，让定时器在key的过期时间来临时，对key进行删除
* 惰性删除
  + key过期的时候不删除，每次从数据库获取key的时候去检查是否过期，若过期，则删除，返回null。
* 定期删除
  + 每隔一段时间执行一次删除(在redis.conf配置文件设置hz，1s刷新的频率)过期key操作

##### 如何保证redis中的数据都是热点数据

* redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。redis 提供 6种数据淘汰策略：
  + volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
  + volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
  + volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
  + allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
  + allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰
  + no-enviction（驱逐）：禁止驱逐数据

##### 如何基于redis实现发布和订阅

[参考链接](https://www.jianshu.com/p/c4d7255425da)

# 发布者  
#coding:utf-8import time  
import redis  
  
number\_list = ['300033', '300032', '300031', '300030']  
signal = ['1', '-1', '1', '-1']  
  
rc = redis.StrictRedis(host='\*\*\*', port='6379', db=3, password='\*\*\*\*\*\*\*\*')  
for i in range(len(number\_list)):  
 value\_new = str(number\_list[i]) + ' ' + str(signal[i])  
 rc.publish("liao", value\_new) #发布消息到liao

# 订阅者  
#coding:utf-8import time  
import redis  
  
rc = redis.StrictRedis(host='\*\*\*\*', port='6379', db=3, password='\*\*\*\*\*\*')  
ps = rc.pubsub()  
ps.subscribe('liao') #从liao订阅消息for item in ps.listen(): #监听状态：有消息发布了就拿过来  
 if item['type'] == 'message':  
 print item['channel']  
 print item['data']

##### 什么是codis

* Codis 是一个分布式 Redis 解决方案, 对于上层的应用来说, 连接到 Codis Proxy 和连接原生的 Redis Server 没有明显的区别 (有一些命令不支持), 上层应用可以像使用单机的 Redis 一样使用, Codis 底层会处理请求的转发, 不停机的数据迁移等工作, 所有后边的一切事情, 对于前面的客户端来说是透明的, 可以简单的认为后边连接的是一个内存无限大的 Redis 服务，当然，前段时间redis官方的3.0出了稳定版，3.0支持集群功能，codis的实现原理和3.0的集群功能差不多。

##### 什么是Twemproxy

* Twemproxy是一种代理分片机制，由Twitter开源。Twemproxy作为代理，可接受来自多个程序的访问，按照路由规则，转发给后台的各个Redis服务器，再原路返回。该方案很好的解决了单个Redis实例承载能力的问题。当然，Twemproxy本身也是单点，需要用Keepalived做高可用方案。通过Twemproxy可以使用多台服务器来水平扩张redis服务，可以有效的避免单点故障问题。虽然使用Twemproxy需要更多的硬件资源和在redis性能有一定的损失（twitter测试约20%），但是能够提高整个系统的HA也是相当划算的。不熟悉twemproxy的同学，如果玩过nginx反向代理或者mysql proxy，那么你肯定也懂twemproxy了。其实twemproxy不光实现了redis协议

##### redis如何实现事务

[参考链接](https://blog.csdn.net/hxpjava1/article/details/79553073)

##### redis中watch的作用

watch 用于在进行事务操作的最后一步也就是在执行exec 之前对某个key进行监视 如果这个被监视的key被改动，那么事务就被取消，否则事务正常执行. 一般在MULTI 命令前就用watch命令对某个key进行监控.如果想让key取消被监控，可以用unwatch命令

##### [redis常见的面试问题参考](https://www.cnblogs.com/Survivalist/p/8119891.html)

##### DNS域名解析过程

1. 浏览器检查缓存中有没有这个域名对应的解析后的IP地址，如果缓存中有，解析过程结束。缓存大小、时间都有限制，时间由TTL属性决定；
2. 如果浏览器缓存中么有，浏览器会查找操作系统缓存中有无这个域名DNS解析后的结果。操作系统也有一个域名解析的过程，windows通过C:32，浏览器会优先使用这个解析结果（Win7已将hosts设置为只读），linux系统中/etc/named.conf。目前为止都是在本机完成，如果未完成，才会真正请求域名服务器解析域名。
3. “网络配置”中都会有“DNX服务器地址”，操作系统会把域名发送给这个LDNS，本地区的域名服务器，通常都会提供一个本地互联网接入的DNS解析服务。就在你所在城市的某个角落，通过ipconfig可以看到。
4. 如果LDNS仍然没有命中，则向RootServer域名服务器请求解析。
5. 根域名服务器向本地域名服务器返回一个所查询域的主域名服务器（gTLD Server）。国际顶级域名服务器（.com、.cn、.org等），全球13台。
6. 本地域名服务器（Local DNS Server）再向上一步返回的gTLD发送请求。
7. gTLD返回域名对应NameServer域名服务器地址，通常由你购买域名的服务商提供。
8. NameServer服务器查询域名与IP映射关系表，返回目标IP记录和TTL值给DNS Server域名服务器。
9. Local DNS Server根据TTL缓存该IP解析。
10. 缓存结果返回给用户，用户根据TTL缓存到本地操作系统中，域名解析过程结束。

##### 了解过Hbase，DB2，SQLServer，Access吗

* Hbase：HBase是一个分布式的、面向列的开源数据库
* DB2：一套关系型数据库管理系统，
* SQLServer：SQL Server是由Microsoft开发和推广的关系数据库管理系统
* Sccess：Access是由微软发布的关系数据库管理系统。

##### JavaScript(或者jQuery)如何选择一个id为main的容器

jquery：$('#id') JavaScript：document.getElementById("id"))

##### css如何隐藏一个元素

* display：none

##### 前后端分离的基本原理

[参考链接](https://www.cnblogs.com/Kevin-ZhangCG/p/9501635.html)

* 前后端分离并非仅仅只是一种开发模式，而是一种架构模式（前后端分离架构）。前端项目与后端项目是两个项目，放在两个不同的服务器，需要独立部署，两个不同的工程，两个不同的代码库，不同的开发人员。前后端工程师需要约定交互接口，实现并行开发，开发结束后需要进行独立部署，前端通过Ajax来调用HTTP请求调用后端的restful api。前端只需要关注页面的样式与动态数据的解析&渲染，而后端专注于具体业务逻辑。

##### 如何保证api调用时数据的安全性

1. 通信使用https
2. 请求签名，防止参数被篡改
3. 身份确认机制，每次请求都要验证是否合法
4. app中使用ssl pinning防止抓包操作
5. 对所有的请求和响应都进行加解密操作

##### 如何实现响应式布局

1. 流体布局
   * 其实就是度量单位的改变。在响应式设计的布局中，不在把像素(px)作为唯一的单位，而是采用%或者是混合%、px为单位，设计出自己想要的布局方式。
2. 媒体查询
   * 媒体查询可以在你根据特定的环境下查询到各种属性---------比如设备类型，分辨率、屏幕物理尺寸以及色彩等。通过使用媒体查询，可以获得设备的一些特性，以及响应式的布局方案。
3. 弹性图片
   * 其实在做响应式布局时，大多用到的是弹性盒子进行布局，那么在设置图片的地方也应该具有一些变化以适应布局的变化。出了图片外，像图标啦！视频啦也应做一些调整用以适应布局的变化。

##### 曾经使用过哪些前端框架

* react，vue，bootstrap，elementUI，Echarts

##### 什么是ajax请求？手写一个ajax请求

* ajax（异步JavaScript和XML）是指一种创建交付式网页应用的网页开发技术。可以在不重新加载整个网页的情况下，对网页的某部分进行更新。 ```javascript // 不使用第三方 var xhr = new XMLHttpRequest(); xhr.open("GET", url, false); xhr.onreadtstatechange = function () { if (xhr.readystate == 4) { //响应内容解析完成，可以在客户端调用了 if (xhr.status == 200) { //客户端的请求成功了 alert(xhr.responseText); } } } xhr.send(null);

// 使用ajax

$.ajax({ type:"GET", url:"", dataType:"json", success:function(data){ }, error:function(jqXHR){ } });

```

##### 什么是轮询和长轮询

* 轮询是在特定的的时间间隔（如每1秒），由浏览器对服务器发出HTTP request，然后由服务器返回最新的数据给客户端的浏览器。这种传统的HTTP request 的模式带来很明显的缺点 – 浏览器需要不断的向服务器发出请求，然而HTTP request 的header是非常长的，里面包含的有用数据可能只是一个很小的值，这样会占用很多的带宽。
* var xhr = new XMLHttpRequest();   
  setInterval(function(){  
   xhr.open('GET','/user');   
   xhr.onreadystatechange = function(){ };   
   xhr.send();  
  },1000)
* 长轮询是ajax实现:在发送ajax后,服务器端会阻塞请求直到有数据传递或超时才返回。 客户端JavaScript响应处理函数会在处理完服务器返回的信息后，再次发出请求，重新建立连接。

function ajax(){  
 var xhr = new XMLHttpRequest();  
 xhr.open('GET','/user');  
 xhr.onreadystatechange = function(){  
 ajax();  
 };  
 xhr.send();  
 }

##### vuex的作用

1. 组件之间的数据通信
2. 使用单向数据流的方式进行数据的去中心化管理

##### vue中的路由拦截器的作用

* 当某些页面需要访问权限时，可以使用路由拦截器对用户权限进行判断

##### axios的作用

* axios是基于promise的用于浏览器和nodejs的HTTP客户端，本身有以下特征：

1. 从浏览器中创建XMLHttpRequest；
2. 从nodejs发出http请求
3. 支持promiseAPI
4. 拦截 请求和响应
5. 转换请求和响应数据
6. 取消请求
7. 自动转换JSON数据
8. 客户端支持防止CSRF/XSRF攻击

##### 简述jsonp及其原理

* JSONP是JSON with Padding的略称。它是一个非官方的协议，它允许在服务器端集成Script tags返回至客户端，通过javascript callback的形式实现跨域访问（这仅仅是JSONP简单的实现形式）
* 原理：
* 标签的src属性并不被同源策略所约束，所以可以获取任何服务器上脚本并执行。

javascript <script type="text/javascript" src="http://localhost:20002/test.js"></script>

##### 简述http协议以及常用请求头

* HTTP(超文本传输协议)是一个应用层协议，由请求和相应构成，是一个标准的客户端服务器模型。HTTP通常承载于TCP协议之上，有时也承载于TLS或SSL协议层之上，这个时候，就成了常说的HTTPS。默认HTTP的端口号为80，HTTPS的端口号为443。
* 常用请求头
* Accept：向服务器申明客户端（浏览器）可以接受的媒体类型（MIME）的资源
* Accept-encoding：向服务器申明客户端（浏览器）接收的编码方法，通常为压缩方法
* Accept-Language：向服务器申明客户端（浏览器）接收的语言
* Cookie：告诉服务器关于 Session 的信息，存储让服务器辨识用户身份的信息。
* Refer：告诉服务器该页面从哪个页面链接的。
* User-agent：向服务器发送浏览器的版本、系统、应用程序的信息。

##### 列举常用的http请求方法

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。

##### 列举常用的状态码

1xx：指示信息–表示请求已接收，继续处理 2xx：成功–表示请求已被成功接收、理解、接受 3xx：重定向–要完成请求必须进行更进一步的操作 4xx：客户端错误–请求有语法错误或请求无法实现 5xx：服务器端错误–服务器未能实现合法的请求 常见状态代码、状态描述、说明： 200 OK //客户端请求成功 400 Bad Request //客户端请求有语法错误，不能被服务器所理解 401 Unauthorized //请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用 403 Forbidden //服务器收到请求，但是拒绝提供服务 404 Not Found //请求资源不存在，eg：输入了错误的 URL 500 Internal Server Error //服务器发生不可预期的错误 503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

##### http和https的区别

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。

##### 简述websocket协议及实现原理

[参考链接](https://www.cnblogs.com/nnngu/p/9347635.html)

* WebSocket用于在Web浏览器和服务器之间进行任意的双向数据传输的一种技术。WebSocket协议基于TCP协议实现，包含初始的握手过程，以及后续的多次数据帧双向传输过程。其目的是在WebSocket应用和WebSocket服务器进行频繁双向通信时，可以使服务器避免打开多个HTTP连接进行工作来节约资源，提高了工作效率和资源利用率。

##### django中如何实现websocket

* 通过使用channels模块来实现

##### python web开发中跨域问题的解决思路

[参考链接](https://blog.csdn.net/a961634066/article/details/82765554)

1. 使用django-cors-headers模块，给跨域增加忽略
2. 使用jsonp
3. 修改对应的api实现函数views.py，允许其他域通过ajax请求数据。

##### 简述http缓存机制

[参考链接](http://www.cnblogs.com/chenqf/p/6386163.html)

* 对于强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。对于比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。

##### 什么是wsgi

* WSGI是Python在处理HTTP请求时，规定的一种处理方式。如一个HTTP Request过来了，那么就有一个相应的处理函数来进行处理和返回结果。WSGI就是规定这个处理函数的参数长啥样的，它的返回结果是长啥样的？至于该处理函数的名子和处理逻辑是啥样的，那无所谓。简单而言，WSGI就是规定了处理函数的输入和输出格式。

##### 列举django的内置组件

* .Admin是对model中对应的数据表进行增删改查提供的组件
* .model组件：负责操作数据库
* .form组件：1.生成HTML代码2.数据有效性校验3校验信息返回并展示
* .ModelForm组件即用于数据库操作,也可用于用户请求的验证

##### django请求的生命周期

* 当用户在浏览器中输入url时,浏览器会生成请求头和请求体发给服务端 请求头和请求体中会包含浏览器的动作(action),这个动作通常为get或者post,体现在url之中.
* url经过Django中的wsgi,再经过Django的中间件,最后url到过路由映射表,在路由中一条一条进行匹配, 一旦其中一条匹配成功就执行对应的视图函数,后面的路由就不再继续匹配了.
* 视图函数根据客户端的请求查询相应的数据.返回给Django,然后Django把客户端想要的数据做为一个字符串返回给客户端.
* 客户端浏览器接收到返回的数据,经过渲染后显示给用户.

##### 列举django中间件的5个方法

* process\_request : 请求进来时,权限认证
* process\_view : 路由匹配之后,能够得到视图函数
* process\_exception : 异常时执行
* process\_template\_responseprocess : 模板渲染时执行
* process\_response : 请求有响应时执行

##### 简述什么是FBV和CBV

FBV和CBV本质是一样的，基于函数的视图叫做FBV，基于类的视图叫做CBV 在python中使用CBV的优点： - .提高了代码的复用性，可以使用面向对象的技术，比如Mixin（多继承） - .可以用不同的函数针对不同的HTTP方法处理，而不是通过很多if判断，提高代码可读性

##### 如何给CBV的程序添加装饰器

from django.utils.decorators import method\_decorator  
# 1、给方法加：  
@method\_decorator(check\_login)  
def post(self, request):  
 ...  
# 2、给dispatch加：  
@method\_decorator(check\_login)  
def dispatch(self, request, \*args, \*\*kwargs):  
 ...  
# 3、给类加：  
@method\_decorator(check\_login, name="get")  
@method\_decorator(check\_login, name="post")  
class HomeView(View):  
 ...

##### 列举django orm中的所有方法

<1> all(): 查询所有结果   
 <2> filter(\*\*kwargs): 它包含了与所给筛选条件相匹配的对象。获取不到返回None  
 <3> get(\*\*kwargs): 返回与所给筛选条件相匹配的对象，返回结果有且只有一个。  
 如果符合筛选条件的对象超过一个或者没有都会抛出错误。  
 <4> exclude(\*\*kwargs): 它包含了与所给筛选条件不匹配的对象  
 <5> order\_by(\*field): 对查询结果排序  
 <6> reverse(): 对查询结果反向排序   
 <8> count(): 返回数据库中匹配查询(QuerySet)的对象数量。   
 <9> first(): 返回第一条记录   
 <10> last(): 返回最后一条记录   
 <11> exists(): 如果QuerySet包含数据，就返回True，否则返回False  
 <12> values(\*field): 返回一个ValueQuerySet——一个特殊的QuerySet，运行后得到的  
 并不是一系 model的实例化对象，而是一个可迭代的字典序列  
 <13> values\_list(\*field): 它与values()非常相似，它返回的是一个元组序列，values返回的是一个字典序列  
 <14> distinct(): 从返回结果中剔除重复纪录

##### select\_related和prefetch\_related的区别

前提：有外键存在时，可以很好的减少数据库请求的次数,提高性能 select\_related通过多表join关联查询,一次性获得所有数据,只执行一次SQL查询 prefetch\_related分别查询每个表,然后根据它们之间的关系进行处理,执行两次查询

##### filter和exclude的区别？

两者取到的值都是QuerySet对象,filter选择满足条件的,exclude:排除满足条件的.

##### 列举django orm中三种能写sql语句的方法

1.使用execute执行自定义的SQL  
 直接执行SQL语句（类似于pymysql的用法）  
 # 更高灵活度的方式执行原生SQL语句  
 from django.db import connection  
 cursor = connection.cursor()  
 cursor.execute("SELECT DATE\_FORMAT(create\_time, '%Y-%m') FROM blog\_article;")  
 ret = cursor.fetchall()  
 print(ret)  
2.使用extra方法 ：queryset.extra(select={"key": "原生的SQL语句"})  
3.使用raw方法  
 1.执行原始sql并返回模型  
 2.依赖model多用于查询

##### values和values\_list的区别

* values : queryset类型的列表中是字典
* values\_list : queryset类型的列表中是元组

##### cookie和session的区别

* cookie: cookie是保存在浏览器端的键值对,可以用来做用户认证
* session： 将用户的会话信息保存在服务端,key值是随机产生的字符串,value值是session的内容 依赖于cookie将每个用户的随机字符串保存到用户浏览器上
* Django中session默认保存在数据库中：django\_session表
* flask,session默认将加密的数据写在用户的cookie中

##### 如何使用django orm批量创建数据

objs=[models.Book(title="图书{}".format(i+15)) for i in range(100)]  
models.Book.objects.bulk\_create(objs)

##### django的Form组件中，如果字段中包含choices参数，使用两种方式实现数据源实时更新

# 1. 重写构造函数  
def\_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.fields["city"].widget.choices = models.City.objects.all().values\_list("id", "name")  
  
# 2. 利用ModelChoiceField字段，参数为queryset对象  
authors = form\_model.ModelMultipleChoiceField(queryset=models.NNewType.objects.all())//多选

##### django的Model中的ForeignKey字段中的on\_delete参数有什么作用？

* 删除关联表中的数据时,当前表与其关联的field的操作
* django2.0之后，表与表之间关联的时候,必须要写on\_delete参数,否则会报异常

##### django模板中自定义filter和simple\_tag的区别

* 自定义filter：{{ 参数1|filter函数名:参数2 }} 1.可以与if标签来连用 2.自定义时需要写两个形参

例子：自定义filter  
 1. 在app01下创建一个叫templatetags的Python包  
 2. 在templatetags的文件夹下创建py文件 myfilters  
 3. 在py文件中写代码  
 from django import template  
 register = template.Library()  
   
 @register.filter  
 def add\_sb(value,arg='aaa'):  
 return "{}\_sb\_{}".formart(value,arg)  
   
 @register.filter(name='sb')  
 def add\_sb(value,arg='aaa'):  
 return "{}\_sb\_{}".formart(value,arg)   
 4. 使用自定义filter  
 {% load myfilters %}  
 {{ name|add\_sb:'xxx'}}  
 {{ name|sb:'xxx'}}

* simple\_tag:{% simple\_tag函数名 参数1 参数2 %} 1.可以传多个参数,没有限制 2.不能与if标签来连用

例子：自定义simpletag  
 创建  
 1 、在app01中创建一个名字是templatetags的包，  
 2、在包中创建一个py文件  
 3、在py文件中导入  
 from django import template  
 register = template.Library()  
 4、写函数  
 @register.simple\_tag(name="plus")  
 def plus(a,b,c):  
 return '{}+{}+{}'.format(a,b,c)  
 5、加装饰器@register.simple\_tag(name="plus")  
 使用  
 1、导入  
 {% load mytag %}  
 2、使用  
 {% plus 1 2 3 %}

##### django中csrf的实现机制

1. django第一次响应来自某个客户端的请求时,后端随机产生一个token值，把这个token保存在SESSION状态中;同时,后端把这个token放到cookie中交给前端页面；
2. 下次前端需要发起请求（比如发帖）的时候把这个token值加入到请求数据或者头信息中,一起传给后端；Cookies:{csrftoken:xxxxx}
3. 后端校验前端请求带过来的token和SESSION里的token是否一致。

##### 基于django使用ajax发送post请求时，都可以使用哪种方法携带csrf token？

1.后端将csrftoken传到前端，发送post请求时携带这个值发送  
data: {  
 csrfmiddlewaretoken: '{{ csrf\_token }}'  
 },  
  
2.获取form中隐藏标签的csrftoken值，加入到请求数据中传给后端  
data: {  
 csrfmiddlewaretoken:$('[name="csrfmiddlewaretoken"]').val()  
 },  
  
3.cookie中存在csrftoken,将csrftoken值放到请求头中  
headers:{ "X-CSRFtoken":$.cookie("csrftoken")}

##### django本身提供了runserver，为什么不能用来部署

1. 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 。
2. 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。

##### django配置实现数据库读写分离

[参考连接](https://blog.csdn.net/linzi1994/article/details/82934612)

1. 在配置文件中添加slave数据库配置
2. 创建数据库操作的路由分发类
3. 配置读写分离路由

##### django中F和Q的作用

* F查询：对数据本身的不同字段进行操作 如:比较和更新
* Q查询：对对象进行复杂查询，并支持and，or，not等操作符

##### django中only和defer的区别

* defer--> 除了指定字段之外
* only--> 只查询几个字段
* 比如：ret=Book.object.all().only('name')
* id始终会查，结果是queryset对象,套book对象(里面只有id与name字段)
* 如果取price，会发生什么？他会再次查询数据库，对数据库造成压力

##### django中的Form组件和ModelForm组件的作用

* Form作用： 1.在前端生成HTML代码 2.对数据作有效性校验 3.返回校验信息并展示
* ModeForm：根据模型类生成From组件,并且可以操作数据库

##### django路由系统中的name的作用

* 用于反向解析路由，相当于给url取个别名，只要这个名字不变，即使对应的url改变
* 通过改名字也能找到该条url

##### django如何实现单元测试

[参考链接](https://www.cnblogs.com/1204guo/p/8058449.html)

* django的单元测试使用python的unittest模块，这个模块使用基于类的方法来定义测试。类名为django.test

##### 解释orm中db first和code first的含义

* 数据持久化的方式：
* db first基于已存在的数据库，生成模型
* code first基于已存在的模型，生成数据库

##### django-debug-toolbar的作用

* 是django的第三方工具包，给django扩展了调试功能，包括查看sql语句，db查询次数，request，headers等

##### django中如何根据数据库表生成model中的类？

1. 在settings中设置要连接的数据库
2. 生成model模型文件

python python manage.py inspectdb

1. 将模型文件导入到models中

python python manage.py inspectdb > app/models.py

##### 使用orm和原生sql的优缺点

1. orm的开发速度快,操作简单。使开发更加对象化
2. 执行速度慢、处理多表联查等复杂操作时，orm的语法会变得复杂
3. sql开发速度慢，执行速度快，性能强

##### django的contenttype组件的作用

* 这个组件保存了项目中所有app和model的对应关系,每当我们创建了新的model并执行数据库迁移后，ContentType表中就会自动新增一条记录
* 当一张表和多个表FK关联,并且多个FK中只能选择其中一个或其中n个时,可以利用contenttypes

##### 谈谈你对restful规范的认识

# 首先restful是一种软件架构风格或者说是一种设计风格，并不是标准，它只是提供了一组设计#原则和约束条件，主要用于客户端和服务器交互类的软件。

# 就像设计模式一样，并不是一定要遵循这些原则，而是基于这个风格设计的软件可以更简洁，更#有层次，我们可以根据开发的实际情况，做相应的改变。

# 它里面提到了一些规范，例如：

# 1.restful 提倡面向资源编程,在url接口中尽量要使用名词，不要使用动词

# 2、在url接口中推荐使用Https协议，让网络接口更加安全

# https://www.bootcss.com/v1/mycss？page=3

# （Https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（安全套接层协议））

# 3、在url中可以体现版本号

# https://v1.bootcss.com/mycss

# 不同的版本可以有不同的接口，使其更加简洁，清晰

# 4、url中可以体现是否是API接口

# https://www.bootcss.com/api/mycss

# 5、url中可以添加条件去筛选匹配

# https://www.bootcss.com/v1/mycss？page=3

# 6、可以根据Http不同的method，进行不同的资源操作

# （5种方法：GET / POST / PUT / DELETE / PATCH）

# 7、响应式应该设置状态码

# 8、有返回值，而且格式为统一的json格式

# 9、返回错误信息

# 返回值携带错误信息

# 10、返回结果中要提供帮助链接，即API最好做到Hypermedia

# 如果遇到需要跳转的情况 携带调转接口的URL

　　ret = {  
 code: 1000,  
 data:{  
 id:1,  
 name:'小强',  
 depart\_id:http://www.luffycity.com/api/v1/depart/8/  
 }  
}

##### 接口的幂等性是什么意思？

1. 是系统的接口对外一种承诺(而不是实现)
2. 承诺只要调用接口成功，外部多次调用对系统的影响都是一致的，不会对资源重复操作

##### 为什么要使用API

* 系统间为了调用数据
* 数据传输格式：json和xml

##### 为什么要使用django rest framework框架

能自动生成符合 RESTful 规范的 API 1.在开发REST API的视图中，虽然每个视图具体操作的数据不同， 但增、删、改、查的实现流程基本一样,这部分的代码可以简写 2.在序列化与反序列化时，虽然操作的数据不同，但是执行的过程却相似,这部分的代码也可以简写 REST framework可以帮助简化上述两部分的代码编写，大大提高REST API的开发速度

##### django rest framework框架中都有哪些组件

1.序列化组件:serializers 对queryset序列化以及对请求数据格式校验 2.路由组件routers 进行路由分发 3.视图组件ModelViewSet 帮助开发者提供了一些类，并在类中提供了多个方法 4.认证组件 写一个类并注册到认证类(authentication\_classes)，在类的的authticate方法中编写认证逻 5.权限组件 写一个类并注册到权限类(permission\_classes)，在类的的has\_permission方法中编写认证逻辑。 6.频率限制 写一个类并注册到频率类(throttle\_classes)，在类的的allow\_request/wait 方法中编写认证逻辑 7.解析器 选择对数据解析的类，在解析器类中注册(parser\_classes) 8.渲染器 定义数据如何渲染到到页面上,在渲染器类中注册(renderer\_classes) 9.分页 对获取到的数据进行分页处理, pagination\_class 10.版本 版本控制用来在不同的客户端使用不同的行为 在url中设置version参数，用户请求时候传入参数。在request.version中获取版本，根据版本不同 做不同处理

##### 简述django rest framework框架的认证流程

1. 用户请求走进来后,走APIView,初始化了默认的认证方法
2. 走到APIView的dispatch方法,initial方法调用了request.user
3. 如果我们配置了认证类,走我们自己认证类中的authentication方法

##### django rest framework如何实现用户的访问频率控制

#使用IP/用户账号作为键，每次的访问时间戳作为值，构造一个字典形式的数据，存起来，每次访问时对时间戳列表的元素进行判断，  
#把超时的删掉，再计算列表剩余的元素数就能做到频率限制了   
#匿名用户：使用IP控制，但是无法完全控制，因为用户可以换代理IP登录用户：使用账号控制，但是如果有很多账号，也无法限制

##### 给用户提供一个接口之前需要提前做什么

1. 跟前端进行交互，确定前端要什么
2. 把需求写成文档保存

##### PV和UV

1. PV：页面访问量，每打开一次页面PV计算+1，页面刷新也是
2. UV：独立访问数，一台电脑终端为一个访客

##### 如何实现用户的登录认证

1. 使用cookie session
2. token 登录成功后生成加密字符串
3. JWT：json web token 缩写 它将用户信息加密到token中，服务器不保存任何用户信息服务器通过使用保存的秘钥来验证token的正确性

##### 简述MVC和MTV

1. MVC软件系统分为三个基本部分：模型(Model)、视图(View)和控制器(Controller)
   * Model：负责业务对象与数据库的映射(ORM)
   * View：负责与用户的交互
   * Control：接受用户的输入调用模型和视图完成用户的请求
2. Django框架的MTV设计模式借鉴了MVC框架的思想,三部分为：Model、Template和View
   * Model(模型)：负责业务对象与数据库的对象(ORM)
   * Template(模版)：负责如何把页面展示给用户
   * View(视图)：负责业务逻辑，并在适当的时候调用Model和Template

* 此外,Django还有一个urls分发器,
* 它将一个个URL的页面请求分发给不同的view处理,view再调用相应的Model和Template

##### git的常见命令

[参考链接](https://www.cnblogs.com/my--sunshine/p/7093412.html)

* git init：在本地新建一个repo,进入一个项目目录,执行git init,会初始化一个repo,并在当前文件夹下创建一个.git文件夹.
* git clone：获取一个url对应的远程Git repo, 创建一个local copy.
* git status：查询repo的状态。
* git log：查看一个分支的提交历史。
* git diff：查看当前文件和暂存区域之间的差异
* git commit：提交已经被add进来的改动
* git reset：还原到某个提交状态
* git checkout：切换分支
* git merge：把一个分支merge进当前的分支
* git tag：在一个提交上建立一个书签
* git pull：更新本地
* git push：提交分支到远程服务器
* git stash：吧当前改动压入一个栈