传智播客™
www.itcast.cn

万人万薪 就业行动

揭秘知名IT企业招聘面试的“核心机密”
助力程序员斩获高薪OFFER的“神兵利器”

程序员面试宝典

Python篇

宝典在手，OFFER无忧

- ★ **8大热门**学科稀缺面试资源
- ★ 20W+学员真实面经**真题汇总**
- ★ **1000+家**名企面试现场“还原”
- ★ **价值5999元**课程资料免费放送

Python 学科独家面试宝典

Python 学科面试宝典

第一部分：Python 基础

列举 Python 中可变类型和不可变类型数据各 3 种

题目标签：其他

试题编号：MS001149

可变：list,dict

不可变：str,int,float,complex,tuple

字典如何按照指定键排序

题目标签：其他

试题编号：MS001156

d = {c:1, b:2, a:3}

for k in sorted(d.keys()):

当 B 大于 C 时，X 小于 C 但 C 绝不会大于 B，所以：X 绝不会大于 B（），X 绝不会小于 B（），X 绝不会小于 C（）

题目标签：其他

试题编号：MS001164

（1）问题分析

（2）核心问题讲解

A 肯定是对的，B 和 C 在题中的假设已经否定了，而且题中已经说了是三个不等数，所以 B=C 是不成立的，所以只有 B 大于 C 这一种情况，而题中说了当 B 大于 C 时，X 小于 C

（3）问题扩展

（4）结合项目中使用

get 与 post 的区别是什么?

题目标签：其他

试题编号：MS001165

（1）问题分析

（2）核心问题讲解

1、get 是从服务器上获取数据。

2、post 是向服务器传送数据。

（3）问题扩展

（4）结合项目中使用

请用代码实现一个简单的工厂设计模式

题目标签：其他

试题编号：MS001167

（1）问题分析

（2）核心问题讲解

```
class Person:
  def __init__(self):
    self.name = None
    self.gender = None
  def getName(self):
    return self.name
  def getGender(self):
    return self.gender
class Male(Person):
  def __init__(self, name):
    print "Hello Mr." + name
class Female(Person):
  def __init__(self, name):
    print "Hello Miss." + name
class Factory:
  def getPerson(self, name, gender):
    if gender == 'M':
      return Male(name)
```

```
    if gender == 'F':
        return Female(name)
if __name__ == '__main__':
    factory = Factory()
    person = factory.getPerson("Chetan", "M")
```

（3）问题扩展

（4）结合项目中使用

用 Python 或者 java 实现一个单利模式类 Singleton

题目标签：其他

试题编号：MS001216

（1）问题分析

（2）核心问题讲解

```
def singleton(cls):
    _instance = {}
    def inner():
        if cls not in _instance:
            _instance[cls] = cls()
        return _instance[cls]
    return inner
```

```
@singleton
class Cls(object):
    def __init__(self):
        pass
cls1 = Cls()
cls2 = Cls()
print(id(cls1) == id(cls2))
```

（3）问题扩展

（4）结合项目中使用

用 Python 或者 java 实现一个生成斐波那契数列的方法

题目标签：斐波那契数列

试题编号：MS001217 删除 MS001159

（1）问题分析

（2）核心问题讲解

斐波那契数列即著名的兔子数列：1、1、2、3、4、5、13、21、34

数列特点：该数列从第三项开始，每个数的值为前两个数之和

```
def   create_fbnq(n):
        a,b=0,1
        i=0
        while I yield b
```

```
        a,b=b,a+b
        i+=1
for x in   create_fbnq(20):
        print(x)
```

（3）问题扩展

（4）结合项目中使用

选择你熟悉的语言用*打印出任意的三角形

题目标签：其他

试题编号：MS001231

（1）问题分析

（2）核心问题讲解

```
for i in range(5):
    print(" "*(4-i),end="")
    print(" * "*(i+1))
```

（3）问题扩展

（4）结合项目中使用

查询字符串，字符串的替换?

题目标签：字符串

试题编号：MS001483

（1）问题分析

（2）核心问题讲解

Python 查找字符串使用 变量.find("要查找的内容"[，开始位置，结束位置])，开始位置和结束位置，表示要查找的范围，为空则表示查找所有。查找到后会返回位置，位置从 0 开始算，如果每找到则返回-1。

Python 替换字符串使用 变量.replace("被替换的内容"，"替换后的内容"[，次数])，替换次数可以为空，即表示替换所有。要注意的是使用 replace 替换字符串后仅为临时变量，需重新赋值才能保存。

（3）问题扩展

（4）结合项目中使用

正则表达式的 match、search 的区别?

题目标签：正则表达式

试题编号：MS001492

（1）问题分析

（2）核心问题讲解

1）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；

2）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。

3）例如：

print(re.match('super', 'superstition').span())　会返回(0, 5)

而 print(re.match('super', 'insuperable'))　则返回 None

4）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)

（3）问题扩展

（4）结合项目中使用

列表去重?

题目标签：列表

试题编号：MS001503

（1）问题分析

（2）核心问题讲解

1）使用集合(结果为升序)

lt2 = list(set(lt1))

2）使用字典

lt2 = list({}.fromkeys(lt1).keys())

3）使用排序

```
lt2 = sorted(set(lt1),key=lt1.index)
```

4）使用列表生成式

```
lt2 = []
[lt2.append(i) for i in lt1 if not i in lt2]
```

即:

```
lt2 = []
for i in lt1:
    if i not in lt2:
        lt2.append(i)
```

5）lambda + reduce（大才小用）

```
func = lambda x,y:x if y in x else x + [y]
lt2 = reduce(func, [[], ] + lt1)
```

（3）问题扩展

（4）结合项目中使用

Python 可变与不可变类型?

题目标签：可变与不可变类型

试题编号：MS001791

（1）问题分析

（2）核心问题讲解

1）什么是不可变类型

变量对应的值中的数据是不能被修改，如果修改就会生成一个新的值从而分配新的内存空间。

不可变类型：

数字（int,long,float）

布尔（bool）

字符串（string）

元组（tuple）

2）什么是不可变类型

变量对应的值中的数据可以被修改，但内存地址保持不变。

不可变类型：

列表（list）

字典（dict）

（3）问题扩展

（4）结合项目中使用

单例的实现?

题目标签：单例模式

试题编号：MS001800

（1）问题分析

（2）核心问题讲解

class Earth(object):

```
    __instance=None #定义一个类属性做判断
    def __new__(cls):
        if cls.__instance==None:
            #如果__instance 为空证明是第一次创建实例
            #通过父类的__new__(cls)创建实例
            cls.__instance=object.__new__(cls)
            return  cls.__instance
        else:
            #返回上一个对象的引用
            return cls.__instance
a = Earth()
print(id(a))
b = Earth()
print(id(b))
```

（3）问题扩展

（4）结合项目中使用

请简单介绍一下 Python2 和 Python3 的区别，你为什么选择 Python2 或者 Python3?

题目标签：Python2 和 Python3 的区别

试题编号：MS001805

（1）问题分析

（2）核心问题讲解

1）Python2 的代码混乱，重复较多 冗余，因为当时来编写的人有 C 语言的大牛 和 java 的大牛等各种大神 所以里面都含有各种语言的影子；Python3 统一了代码 源码规范 清晰 简单优美。

2）Python3：print（"内容"）Python2：ptint()或者 print '内容'；

3）Python3 编码：utf-8

Python2 编码： 默认编码：ascii 解决办法：在首行 # -*- encoding：utf-8-*-

4）用户交互 input

python2: raw-input（）

python3:input ()

5）python2x :unicode 默认 2 个字节表示一个字符 可以在 LINUX 编译安装时做调整

python3x：unicode 默认是 4 个字节表示一个字符

6）python2x 没有 nonlocal

python3x 加入的

7）python3x 新建的包里面的 init 文件如果你删除该文件 包照样可以被调用

python2x 新建的包如果没有 init 文件 则包不能够被调用 直接报错

8）python2 中的经典类 遍历方法是以深度优先 新式类是以广度优先

python3 中不存在经典类 所有的类都是新式类 所以都是广度优先

（3）问题扩展

（4）结合项目中使用

用 Python 实现功能：有两个列表[‘a’,’b’,’c’]与[1,2,3]写一行代码，将后面一个列表每个值加 1，再将两个列表合并成一个字典{“a”：2，“b”:3，“c”:4}?

题目标签：列表

试题编号：MS001829

（1）问题分析

（2）核心问题讲解

```
list1 = [1, 2, 3]
list3 = []
for i in list1:
    list3.append(i+1)
list2 = ['a', 'b', 'c']
dict_out_put = dict(zip(list2, list3))
print(dict_out_put)
```

（3）问题扩展

（4）结合项目中使用

请写出实现单例的方法?

题目标签：面向对象

试题编号：MS001843

（1）问题分析

（2）核心问题讲解

```
class Tools(object):
    instance = None
    init_flag = False
    def __new__(cls, *args, **kwargs):
        if cls.instance is None:
            cls.instance = super().__new__(cls)
        return cls.instance
    def __init__(self):
        if not Tools.init_flag:
            Tools.init_flag = True
```

（3）问题扩展

（4）结合项目中使用

Python 中的可变、不可变类型?

题目标签：数据类型

试题编号：MS001852

（1）问题分析

（2）核心问题讲解

可变类型：列表 集合 字典 对象

不可变：数值 字符串 布尔 元组

（3）问题扩展

（4）结合项目中使用

Python 里面如何生成随机数?

题目标签：random

试题编号：MS002010

使用 random 模块，randint 函数

如何在一个 function 里面设置一个全局变量?

题目标签：global

试题编号：MS002012

使用 global 关键字

编写一个测试回文（从左往右读和从右往左读一样）的函数?

题目标签：字符串

试题编号：MS002019

（1）问题分析

（2）核心问题讲解

```
def test(num):
    a = num // 100
    b = num %10
    if a==b:
        print(num)
    else:
        print("不是回文数")
```

（3）问题扩展

（4）结合项目中使用

字典 m = m = {'a':0, 'b':1}， 请用多种方式完成 key 和 value 的转换?

题目标签：数据类型

试题编号：MS002036

（1）问题分析

（2）核心问题讲解

```
# 第一种:

d = {'a': 1, 'b': 2}

d = {value: key for key, value in d.items()}

print(d)

# 第二种

d = {'a': 1, 'b': 2}

dict(zip(d.values(), d.keys()))
```

（3）问题扩展

（4）结合项目中使用

a = ['aa', 'cd', 'ff', 'aaa', 'aac','ff','gg'],请对 a 进行去重并保持原来顺序不变?

题目标签：列表

试题编号：MS002038

（1）问题分析

（2）核心问题讲解

```
a = ['aa', 'cd', 'ff', 'aaa', 'aac','ff','gg']

b = []

for i in a:

    if i not in b:
```

```
    b.append(i)
print(b)
```

（3）问题扩展

（4）结合项目中使用

有一个文件 file.h,从而得知 model.so 里面包含一个名叫 funA 的方法，请问 Python 怎么调用这个方法?

题目标签：调用和引入

试题编号：MS002039

import 引入这个模块即可使用

a = [2, 5, 7, 8, 3, 9]，一行代码实现对列表 a 基数位置的元素进行乘以 5 后求和?

题目标签：列表

试题编号：MS002040

（1）问题分析

（2）核心问题讲解

"""

1）取出偶数下标：filter

filter(lambda x:x%2==0, range(len(list)))　　==>[0,2,4]

2）取出对应下标值：map

map(lambda x:list[x],filter(lambda x:x%2==0, range(len(list))))　　==>[1,3,5]

3）对应值加 3：lambda

map(lambda x:x+3,map(lambda x:list[x],filter(lambda x:x%2==0, range(len(list)))))

==>[4,6,8]

4）将数组求和 reduce 也可以用 sum()函数

reduce(lambda x,y: x+y,map(lambda x:x+3,map(lambda x:list[x],filter(lambda

x:x%2==0, range(len(list))))))

"""

from functools import reduce

l = [2, 5, 7, 8, 3, 9]

l2 = reduce(lambda x, y : x + y, map(lambda i: i*5, list(filter(lambda y:y%2 == 1,

l))))

l3 = sum(list(map(lambda x: x*5, list(filter(lambda y:y%2 == 1, l)))))

print(l2)

print(l3)

（3）问题扩展

（4）结合项目中使用

Python 中对 numbers = [1,2,3,4,5,6,7,8,9,10]，number[3:6]得到的输出是什么?

题目标签：列表

试题编号：MS002105

[4, 5, 6]

Python 中的负索引是什么?

题目标签：列表

试题编号：MS002138

负索引就是按照列表的逆序取元素，-1 代表最后一个

Python 中的模块和包是什么?

题目标签：模块

试题编号：MS002148

（1）问题分析：

无

（2）核心答案讲解：

Python 的流行主要依赖于其有众多功能强大的库(Library)，Python 自带的标准库(Standard Library)可以满足大多数的基本需求，除了函数库以外，模块(Module)和包

(Package)也常会被提及。其中库、模块和包常常会分不清谁是谁今天就一起来学习下。

模块：

模块是一种以.py 为后缀的文件，在.py 文件中定义了一些常量和函数。模块的名称是该.py 文件的名称。模块的名称作为一个全局变量_ name__ 的取值 可以被其他模块获取或导入。模块的导入通过 ipmort 来实现，导入模块的方式如下：

import 特定模块名称包：

包体现了模块的结构化管理思想，包由模块文件构成，将众多具有相关功能的模块文件结构化组合形成包。从编程开发的角度看，两个开发者 A 和 B 由可能把各自开发且功能不同的模块文件取了相同的名字。如果第三个开发者通过名称导入模块，则无法确认是哪个模块被导入了。为此，开发这 A 和 B 可以构建一个包，将模块放到包文件夹下，通过“包.模块名”来指定模块。示例:

import 包名称.模块名称

一个包文件一半由_ init_ .py 和其他诸 多.py 文件构成。该_ init_ _.py 内容可以为空，有额可以写入一些包执行时的初始化代码。_ init__ .py 是 包的标志性文件，Python 通过一 个文件夹下是否有_ init_ .py 文件， 来识别出文件夹是否为包文件。

（3）问题扩展：

无

（4）结合项目中使用：

无

为什么 lambda 没有语句?

题目标签：lambda

试题编号：MS002158

匿名函数 lambda 没有语句的原因是它被用于在代码执行的时候构建新的函数对象并且返回。

什么是 PEP8?

题目标签：pep8

试题编号：MS002236

（1）问题分析：

无

（2）核心答案讲解：

Python 的代码风格由 PEP 8 描述。这个文档描述了 Python 编程风格的方方面面。在遵守这个文档的条件下，不同程序员编写的 Python 代码可以保持最大程度的相似风格。这样就易于阅读，易于在程序员之间交流。

1）命名风格

总体原则，新编代码必须按下面命名风格进行，现有库的编码尽量保持风格。

尽量以免单独使用小写字母'l'，大写字母'O'，以及大写字母'I'等容易混淆的字母。

模块命名尽量短小，使用全部小写的方式，可以使用下划线。

包命名尽量短小，使用全部小写的方式，不可以使用下划线。

类的命名使用 CapWords 的方式，模块内部使用的类采用_CapWords 的方式。

异常命名使用 CapWords+Error 后缀的方式。

全局变量尽量只在模块内有效，类似 C 语言中的 static。实现方法有两种，一是__all__机制;

二是前缀一个下划线。对于不会发生改变的全局变量，使用大写加下划线。

函数命名使用全部小写的方式，可以使用下划线。

常量命名使用全部大写的方式，可以使用下划线。

使用 has 或 is 前缀命名布尔元素，如: is_connect = True; has_member = False

用复数形式命名序列。如： members = ['user_1', 'user_2']

用显式名称命名字典，如:

person_address = {'user_1':'10 road WD', 'user_2' : '20 street huafu'}

避免通用名称。诸如 list, dict, sequence 或者 element 这样的名称应该避免。又如 os, sys 这种系统已经存在的名称应该避免。

类的属性（方法和变量）命名使用全部小写的方式，可以使用下划线。

对于基类而言，可以使用一个 Base 或者 Abstract 前缀。如 BaseCookie、AbstractGroup

内部使用的类、方法或变量前，需加前缀'_'表明此为内部使用的。虽然如此，但这只是程序员之间的约定而非语法规定，用于警告说明这是一个私有变量，外部类不要去访问它。但实际上，外部类还是可以访问到这个变量。import 不会导入以下划线开头的对象。

类的属性若与关键字名字冲突，后缀一下划线，尽量不要使用缩略等其他方式。

双前导下划线用于命名 class 属性时，会触发名字重整；双前导和后置下划线存在于用户控制的名字空间的"magic"对象或属性。

为避免与子类属性命名冲突，在类的一些属性前，前缀两条下划线。比如：类 Foo 中声明 __a,访问时，只能通过 Foo._Foo__a，避免歧义。如果子类也叫 Foo，那就无能为力了。

类的方法第一个参数必须是 self，而静态方法第一个参数必须是 cls。

一般的方法、函数、变量需注意，如非必要，不要连用两个前导和后置的下线线。两个前导下划线会导致变量在解释期间被更名。两个前导下划线会导致函数被理解为特殊函数，比如操作符重载等。

2）关于参数

要用断言来实现静态类型检测。断言可以用于检查参数，但不应仅仅是进行静态类型检测。Python 是动态类型语言，静态类型检测违背了其设计思想。断言应该用于避免函数不被毫无意义的调用。

不要滥用 *args 和 **kwargs。*args 和 **kwargs 参数可能会破坏函数的健壮性。它们使签名变得模糊，而且代码常常开始在不应该的地方构建小的参数解析器

3）代码编排

缩进。优先使用 4 个空格的缩进（编辑器都可以完成此功能），其次可使用 Tap，但坚决不能混合使用 Tap 和空格。

每行最大长度 79，换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车。

类和 top-level 函数定义之间空两行；类中的方法定义之间空一行；函数内逻辑无关段落之间空一行；其他地方尽量不要再空行。

一个函数 ：不要超过 30 行代码，即可显示在一个屏幕类，可以不使用垂直游标即可看到整个函数；一个类 ：不要超过 200 行代码，不要有超过 10 个方法；一个模块 不要超过 500

行。

4）文档编排

模块内容的顺序：模块说明和 docstring—import—globals&constants—其他定义。其中 import 部分，又按标准、三方和自己编写顺序依次排放，之间空一行。

不要在一句 import 中多个库，比如 import os, sys 不推荐。

如果采用 from XX import XX 引用库，可以省略‘module.’。若是可能出现命名冲突，这时就要采用 import XX。

5）空格的使用

总体原则，避免不必要的空格。

各种右括号前不要加空格。

函数的左括号前不要加空格。如 Func(1)。

序列的左括号前不要加空格。如 list[2]。

逗号、冒号、分号前不要加空格。

操作符（=/+=/-+/==//!=/<>/<=/>=/in/not in/is/is not/and/or/not)左右各加一个空格，不要为了对齐增加空格。如果操作符有优先级的区别，可考虑在低优先级的操作符两边添加空格。如：hypot2 = x*x + y*y; c = (a+b) * (a-b)

函数默认参数使用的赋值符左右省略空格。

不要将多句语句写在同一行，尽管使用‘;’允许。

if/for/while 语句中，即使执行语句只有一句，也必须另起一行。

6）注释

总体原则，错误的注释不如没有注释。所以当一段代码发生变化时，第一件事就是要修改注

释！避免无谓的注释

注释必须使用英文，最好是完整的句子，首字母大写，句后要有结束符，结束符后跟两个空格，开始下一句。如果是短语，可以省略结束符。

行注释：在一句代码后加注释，但是这种方式尽量少使用。。比如：x = x + 1 # Increment

块注释：在一段代码前增加的注释。在‘#’后加一空格。段落之间以只有‘#’的行间隔。

7）文档描述

为所有的共有模块、函数、类、方法写 docstrings；非共有的没有必要，但是可以写注释（在 def 的下一行）。

如果 docstring 要换行，参考如下例子,详见 PEP 257

8）编码建议

编码中考虑到其他 Python 实现的效率等问题，比如运算符‘+’在 CPython（Python）中效率很高，都是 Jython 中却非常低，所以应该采用.join()的方式。

与 None 之类的单件比较，尽可能使用‘is’ ‘is not’，绝对不要使用‘==’，比如 if x is not None 要优于 if x。

使用 startswith() and endswith()代替切片进行序列前缀或后缀的检查。比如：建议使用 if foo.startswith('bar'): 而非 if foo[:3] == 'bar':

使用 isinstance()比较对象的类型。比如：建议使用 if isinstance(obj, int): 而非 if type(obj) is type(1):

判断序列空或不空，有如下规则：建议使用 if [not] seq: 而非 if [not] len(seq)

字符串不要以空格收尾。

二进制数据判断使用 if boolvalue 的方式。

使用基于类的异常，每个模块或包都有自己的异常类，此异常类继承自 Exception。错误型的异常类应添加"Error"后缀，非错误型的异常类无需添加。

异常中不要使用裸露的 except，except 后跟具体的 exceptions。

异常中 try 的代码尽可能少

（3）问题扩展：

无

（4）结合项目中使用：

无

什么是 pickling 和 unpickling?

题目标签：pickle

试题编号：MS002245

（1）问题分析：

无

（2）核心答案讲解：

在文件中，字符串可以很方便的读取写入，数字可能稍微麻烦一些，因为 read()方法只返回字符串，我们还需要将其传给 int()这样的函数，使其将如"1994"的字符串转为数字 1945.但是，如果要半寸更复杂的数据类型，如列表，字典，或者类的实例，那么就会更复杂了。

为了让用户在平常的编程和测试时保存复杂的数据类型，Python 提供了标准模块，称

为 pickle.这个模块可以将几乎任何的 Python 对象(甚至是 Python 的代码)，转换为字符串表示，这个过程称为 pickling。而要从里面重新构造回原来的对象，则称为 unpickling。在 pickling 和 unpicking 之间，表示这些对象的字符串表示，可以存于一个文件，也可以通过网络远程机器间传输。

pickling: Pickle 模块读入任何 Python 对象,将它们转换成字符串,然后使用 dump 函数将其转储到一个文件中

unpickling: 从存储的字符串文件中提取原始 Python 对象的过程,叫做 unpickling 。

（3）问题扩展：

无

（4）结合项目中使用：

无

Python 是如何进行内存管理的?

题目标签：内存管理

试题编号：MS002256

（1）问题分析：

无

（2）核心答案讲解：

Python 是如何进行内存管理的

Python 引入了一个机制：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，即引用计数，当对象被创建时就创建了一个引用计数，当对象不再需要时，这个对象的引用计数为 0 时，它被垃圾回收。

总结一下对象会在一下情况下引用计数加 1：

1）对象被创建：x=4

2）另外的别人被创建：y=x

3）被作为参数传递给函数：foo(x)

4）作为容器对象的一个元素：a=[1,x,'33']

引用计数减少情况

1）一个本地引用离开了它的作用域。比如上面的 foo(x)函数结束时，x 指向的对象引用减 1。

2）对象的别名被显式的销毁：del x ；或者 del y

3）对象的一个别名被赋值给其他对象：x=789

4）对象从一个窗口对象中移除：myList.remove(x)

5）窗口对象本身被销毁：del myList，或者窗口对象本身离开了作用域。

垃圾回收

1）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。它会去检查那些引用计数为 0 的对象，然后清除其在内存的空间。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，他们本身其他的引用已经为 0

了。

2）垃圾回收机制还有一个循环垃圾回收器，确保释放循环引用对象(a 引用 b, b 引用 a, 导致其引用计数永远不为 0)。

在 Python 中，许多时候申请的内存都是小块的内存，这些小块内存在申请后，很快又会被释放，由于这些内存的申请并不是为了创建对象，所以并没有对象一级的内存池机制。这就意味着 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。

内存池机制

Python 提供了对内存的垃圾收集机制，但是它将不用的内存放到内存池而不是返回给操作系统。

Python 中所有小于 256 个字节的对象都使用 pymalloc 实现的分配器，而大的对象则使用系统的 malloc。另外 Python 对象，如整数，浮点数和 List，都有其独立的私有内存池，对象间不共享他们的内存池。也就是说如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。

（3）问题扩展：

无

（4）结合项目中使用：

无

什么是 lambda 函数？它有什么好处？

题目标签：lambda

试题编号：MS002263

（1）问题分析：

无

（2）核心答案讲解：

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。

（注意：lambda 函数不能包含命令，它们所包含的表达式不能超过一个）

lamda 函数有什么好处？

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）问题扩展：

无

（4）结合项目中使用：

Python 语言，下列代码输出结果是什么？

```
>>>li = [[]]*5
>>>li
```

```
>>>li[0].append(10)

>>>li

>>>li[1].append(20)

>>>li

>>>li[2].append(30)

>>>li
```

题目标签：列表

试题编号：MS002363

```
>>>li = [[]]*5

>>>li

[[], [], [], [], []]

>>>li[0].append(10)

>>>li

[[10], [10], [10], [10], [10]]

>>>li[1].append(20)

>>>li

[[10, 20], [10, 20], [10, 20], [10, 20], [10, 20]]

>>>li[2].append(30)

>>>li

[[10, 20, 30], [10, 20, 30], [10, 20, 30], [10, 20, 30], [10, 20, 30]]
```

尽可能多的写出列表去重的方案，另写出去重的同时保持顺序不变的方案

试题编号：MS002364

题目标签：列表

有道面试题：将列表 L = [3, 1, 2, 1, 3, 4]去掉重复的元素，但保留原先顺序。 最后结果应该是：[3, 1, 2, 4]

如果不保留顺序的去重，很好处理，比如用 set

```
>>> L = [3, 1, 4, 2, 3]
>>> list(set(L))
[1, 2, 3, 4]
```

如果要保留原先顺序不变，同样也有很多种方法:

方法一：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> T = []
>>> for i in L:
...     if not i in T:
...         T.append(i)
>>> T
[3, 1, 2, 4]
```

方法二：

```
>>> L = [3, 1, 2, 1, 3, 4]
```

```
>>> T = []
>>> [T.append(i) for i in L if not i in T ]
[None, None, None, None]

>>> T
[3, 1, 2, 4]
```

方法三：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> T = list(set(L))
>>> T
[1, 2, 3, 4]
>>> T.sort(key=L.index)
>>> T
[3, 1, 2, 4]
```

方法四：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> T = []
>>> for i,v in enumerate(L):
...     if L.index(v) == i:
...         T.append(v)
```

```
>>> T
[3, 1, 2, 4]
```

方法五：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> T = {}.fromkeys(L).keys()
>>> T
[1, 2, 3, 4]

>>> T.sort(key=L.index)
>>> T
[3, 1, 2, 4]
```

方法六：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> T = L[:]
>>> for i in L:
...     while T.count(i) > 1:
...         del T[T.index(i)]
>>> T
[2, 1, 3, 4]

>>> T.sort(key=L.index)
```

```
>>> T
[3, 1, 2, 4]
```

方法七：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> T = sorted(set(L), key=L.index)
>>> T
[3, 1, 2, 4]
```

方法八：

```
>>> L = [3, 1, 2, 1, 3, 4]
>>> func = lambda L,i: L if i in L else L + [i]
>>> T = reduce(func, [[], ] + L)
>>> T
[3, 1, 2, 4]
```

将字符串“abc”转换成“cba”,尝试尽可能多方式实现

试题编号：MS002365

题目标签：字符串

```
>>> a='abc'
>>> a[::-1]
'cba'
```

“google”.count("o",0,2)的值是多少?

题目标签：字符串

试题编号：MS002366

（1）问题分析：

无

（2）核心答案讲解：1

（3）问题扩展：

无

（4）结合项目中使用：

无

列举 Djangoorm 中的方法（QuerySet 对象的方法），列举 8 个

题目标签：Django

试题编号：MS002376

（1）问题分析：

无

（2）核心答案讲解：

all()查询所有结果

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

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。获取不到会抱胸 #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

exclude(**kwargs)它包含了与所给筛选条件不匹配的对象

order_by(*field)对查询结果排序

reverse()对查询结果反向排序

count()返回数据库中匹配查询(QuerySet)的对象数量

first()返回第一条记录

last()返回最后一条记录

exists()如果 QuerySet 包含数据，就返回 True，否则返回 False

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

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

distinct()从返回结果中剔除重复纪录

（3）问题扩展：

无

（4）结合项目中使用：

无

第二部分：Python 高级

bash shell 脚本第一行是什么

题目标签：Linux

试题编号：MS001119

#!/bin/bash

系统管理常用的二进制文件放在哪个目录

题目标签：系统

试题编号：MS001125

二进制命令文件通常都在 bin、/sbin 以及/usr/bin、/usr/sbin 目录

强制杀死进程的命令

题目标签：Linux

试题编号：MS001127

kill - 9 进程号

查找当前用户运行的所有进程的信息

题目标签：Linux

试题编号：MS001131

ps aux | less

你平时喜欢用哪种方式定位元素？最喜欢哪种，为什么？

题目标签：Linux

试题编号：MS001137

喜欢用 xpath 定位元素，因为 xpath 的很多属性用起来很方便，而且直接右键 copy xpath 就可以

webdriver 的工作原理是什么

题目标签：爬虫

试题编号：MS001147

（1）问题分析

（2）核心问题讲解

WebDriver 是一个用来进行复杂重复的 web 自动化测试的工具。意在提供一种比 Selenium1.0 更简单易学，有利于维护的 API。它没有和任何测试框架进行绑定，所以他可以很好的在单元测试和 main 方法中调用。一旦创建好一个 Selenium 工程，你马上会发现 WebDriver 和其他类库一样：它是完全独立的，你可以直接使用而不需要考虑其他配置，这个 Selenium RC 是截然相反的。

（3）问题扩展

（4）结合项目中使用

Python 如何捕获异常?

题目标签：异常

试题编号：MS001158

（1）问题分析

（2）核心问题讲解

try:

语句　　#运行别的代码

Except:

语句　　#如果在 try 部份引发了'name'异常</pre>

（3）问题扩展

（4）结合项目中使用

epoll 和 select 的区别

题目标签：其他

试题编号：MS001160

（1）问题分析

（2）核心问题讲解

epoll 跟 select 都能提供多路 I/O 复用的解决方案。在现在的 Linux 内核里有都能够支持，其中 epoll 是 Linux 所特有，而 select 则应该是 POSIX 所规定，一般操作系统均有实现。

（3）问题扩展

（4）结合项目中使用

Python 内存管理机制有哪些

题目标签：Python 内存

试题编号：MS001161

（1）垃圾回收

（2）引用计数

（3）内存池机制

编写 SQL 语句统计员工人数超够 20 人的部门中工资大于 3000 的高级程序员最大工资和最小工资是多少？标为：create table emp(empid(工号)，name(姓名)，deptno(部门代码),duty(职务)，salary(工资))

题目标签：MySQL

试题编号：MS001163

（1）问题分析

（2）核心问题讲解

select * from 部门 where not exists (select * from 员工 where 员工.部门号=部门.部

门号 and 工资 <= 5000)

（3）问题扩展

（4）结合项目中使用

编写一个方法，用于实现 Redis 连接池

题目标签：其他

试题编号：MS001173

（1）问题分析

（2）核心问题讲解

import Redis

拿到一个 Redis 的连接池

Pool = Redis.ConnectionPool(host='127.0.0.1', port=6379,

max_connections=10)

从池子中拿一个链接

conn = Redis.Redis(connection_pool=pool，decode_responses=True)

print(conn.get(name).decode(utf-8))

（3）问题扩展

（4）结合项目中使用

请简述 HTTP 和 TCP 的区别和联系

题目标签：其他

试题编号：MS001227

（1）问题分析

（2）核心问题讲解

Http 协议是建立在 TCP 协议基础之上的，当浏览器需要从服务器获取网页数据的时候，会发出一次 Http 请求。Http 会通过 TCP 建立起一个到服务器的连接通道，当本次请求需要的数据完毕后，Http 会立即将 TCP 连接断开，这个过程是很短的。所以 Http 连接是一种短连接，是一种无状态的连接。

TCP 是底层协议，定义的是数据传输和连接方式的规范。

HTTP 是应用层协议，定义的是传输数据的内容的规范。

HTTP 协议中的数据是利用 TCP 协议传输的，所以支持 HTTP 就一定支持 TCP

（3）问题扩展

（4）结合项目中使用

写出你知道的 http 协议的请求方式

题目标签：其他

试题编号：MS001229

（1）问题分析

（2）核心问题讲解

HEAD：向服务器索要与 GET 请求相一致的响应，只不过响应体将不会被返回。这一方法可以在不必传输整个响应内容的情况下，就可以获取包含在响应消息头中的元信息。

GET：向特定的资源发出请求。

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创建和/或已有资源的修改。

PUT：向指定资源位置上传其最新内容。

DELETE：请求服务器删除 Request-URI 所标识的资源。

TRACE：回显服务器收到的请求，主要用于测试或诊断。

（3）问题扩展

（4）结合项目中使用

数据库事务的基本性质有几个

题目标签：其他

试题编号：MS001230

（1）问题分析

（2）核心问题讲解

1）原子性（Atomicity）

原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2）一致性（Consistency）

一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

拿转账来说，假设用户 A 和用户 B 两者的钱加起来一共是 5000，那么不管 A 和 B 之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是 5000，这就是事务的一致性。

3）隔离性（Isolation）

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，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。

即要达到这么一种效果：对于任意两个并发的事务 T1 和 T2，在事务 T1 看来，T2 要么在 T1 开始之前就已经结束，要么在 T1 结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

4）持久性（Durability）

持久性是指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，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

（3）问题扩展

（4）结合项目中使用

请给 Js 的 String 原生对象添加一个名为 trim 的原型方法，用于截取空白字符。

题目标签：前端

试题编号：MS001260

```
alert(" taobao".trim());    // 输出 "taobao"

alert(" taobao ".trim());    // 输出 "taobao"
```

说明 REST 与传统 WebServiced 区别，并说明再项目开发中如何选择

题目标签：web 前端

试题编号：MS001261

（1）问题分析

（2）核心问题讲解

表征状态转移（英文：Representational State Transfer，简称 REST）是 Roy Fielding 博士在 2000 年他的博士论文中提出来的一种软件架构风格。之所以 REST 服务会牵涉到 WebService 的话题，那是因为 REST 本来就是 WebService 的一种方式

（3）问题扩展

（4）结合项目中使用

简述 cookie 和 session 的区别

题目标签：webservice

试题编号：MS001262

（1）问题分析

（2）核心问题讲解

1）数据存放位置不同：

cookie 数据存放在客户的浏览器上，session 数据放在服务器上。

2）安全程度不同：

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）性能使用程度不同：

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）数据存储大小不同：

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，而 session 则存储与服务端，浏览器对其没有限制。

（3）问题扩展

（4）结合项目中使用

什么是装饰器，编写代码说明其应用。

题目标签：Python 基础

试题编号：MS001263

（1）问题分析

（2）核心问题讲解

Python 的装饰器和 Java 的注解（Annotation）并不是同一回事，和 C#中的特性

（Attribute）也不一样，完全是两个概念。

装饰器的理念是对原函数、对象的加强，相当于重新封装，所以一般装饰器函数都被命名为 wrapper()，意义在于包装。函数只有在被调用时才会发挥其作用。比如@logging 装饰器可以在函数执行时额外输出日志，@cache 装饰过的函数可以缓存计算结果等等。

而注解和特性则是对目标函数或对象添加一些属性，相当于将其分类。这些属性可以通过反射拿到，在程序运行时对不同的特性函数或对象加以干预。比如带有 Setup 的函数就当成准备步骤执行，或者找到所有带有 TestMethod 的函数依次执行等等。

（3）问题扩展

（4）结合项目中使用

js 中，如何阻止事件冒泡和默认事件?

题目标签：web 前端

试题编号：MS001264

（1）问题分析

（2）核心问题讲解

要停止冒泡行为时，可以使用

```
function stopBubble(e) {
//如果提供了事件对象，则这是一个非 IE 浏览器
if ( e && e.stopPropagation )
    //因此它支持 W3C 的 stopPropagation()方法
```

```
    e.stopPropagation();
else
    //否则，我们需要使用 IE 的方式来取消事件冒泡
    window.event.cancelBubble = true;
}
```

当需要阻止默认行为时，可以使用·

```
//code from
http://caibaojian.com/javascript-stoppropagation-preventdefault.html//阻止浏览器
的默认行为
function stopDefault( e ) {
    //阻止默认浏览器动作(W3C)
    if ( e && e.preventDefault )
        e.preventDefault();
    //IE 中阻止函数器默认动作的方式
    else
        window.event.returnValue = false;
    return false;
}
```

（3）问题扩展

（4）结合项目中使用

在 js 中 null 和 undefined 有什么区别?

题目标签：web 前端

试题编号：MS001266

（1）问题分析

（2）核心问题讲解

undefined 表示变量声明但未初始化时的值，null 表示准备用来保存对象，还没有真正保存对象的值。从逻辑角度看，null 值表示一个空对象指针。

JavaScript（ECMAScript 标准）里共有 5 种基本类型：Undefined, Null, Boolean, Number, String，和一种复杂类型 Object。可以看到 null 和 undefined 分属不同的类型，未初始化定义的值用 typeof 检测出来是"undefined"(字符串)，而 null 值用 typeof 检测出来是"object"（字符串）。

任何时候都不建议显式的设置一个变量为 undefined，但是如果保存对象的变量还没有真正保存对象，应该设置成 null。

实际上，undefined 值是派生自 null 值的，ECMAScript 标准规定对二者进行相等性测试要返回 true

（3）问题扩展

（4）结合项目中使用

在 DOM 中，什么是事件委托?

题目标签：web 前端

试题编号：MS001267

（1）问题分析

（2）核心问题讲解

事件：我们的点击（onclick），鼠标经过/离开（onmouseover/onmouseout），键盘摁下/松开（onkeypress/onkeyup）等行为就是一个个的事件。

委托：虽然是周末，我依然在上班，但我有个快递今天会到，我就让在家休假的女朋友帮我取了这个快递。这个过程称之为“委托”（本来该由我去做的事情我加到了别人-女朋友身上）。

事件委托： 由于事件的冒泡，我们点击子元素的时候，会把事件一层层的传递给父级元素。相反的，我们操作元素的时候，直接把事件绑定在父级元素上，而不是分别给子元素绑定事件。通过判断子元素，从而达到同样的效果，这就是所谓的事件委托

（3）问题扩展

（4）结合项目中使用

简述 yield 的作用

题目标签：Python 基础

试题编号：MS001269

（1）问题分析

（2）核心问题讲解

yield 是 Python 中定义为生成器函数，其本质是封装了 __iter__和__next__方法 的迭代器；

与 return 返回的区别：return 只能返回一次值，函数就终止了，而 yield 能多次返回值，每次返回都会将函数暂停，下一次 next 会从上一次暂停的位置继续执行。

（3）问题扩展

（4）结合项目中使用

请实现一个素数生成器，要求该生成器再迭代时顺序输出自然界中的素数

题目标签：Python 基础

试题编号：MS001270

（1）问题分析

（2）核心问题讲解

先说说思路：

素数是一个无限循环，这一点跟生成器的特性很像，所以我们用生成器所谓这个素数序列的主要部分。

第二步就是要筛选了，首先从自然数序列中，从 2 开始的自然数序列，用一个生成器表示比较合适。

从以第个元素开始 2，肯定是素数，收入到素数集合里，取下一个元素，由于前一个元素的所有倍数都被筛选掉了，所以新序列的第一个元素一定是素数“除了自己之外，没有约数”循环不止，筛选不止。得到是这个无限循环列表就是素数序列

筛选的过程我们想起来内建函数 filter()，特性很像 mapreduce，可以用来所列表筛选。

于是第一步，我们先用生成器组建从 2 开始的自然数序列，代码如下：

```
1 def OddList():
2       n = 1
3       while True:
4           n += 1
5           yield
```

第二步：我们要设置筛选函数，对于一个新的序列，我们要拿到第一个元素，然后对整个序列做一次迭代，删掉第一个元素的倍数，该特性与 filter()一样，

```
1 def MultiFilter(multipleNum):
2       return lambda element: element % multipleNum > 0
```

说明：由于 multifilter 是被 filter()包含的，所以 multerfilter 中有一个参数是序列的元素，一定会传进来的，但是由于无法表示 2 个参数，所以考虑用匿名

第三步：限迭代奇数生成器，在每一个迭代中，生成一个新的序列（生成器），生成新序列的过程又是一个迭代过程，在这个过程中把元素 n 的倍数过滤掉

```
1 def PrimeList(maxNum):
2       mainList = OddList()                    #主要为了生成 2
```

```
3       firstElement = next(OddList())      #curentElement 标签始终指向当前元素，作为倍数参与筛选
4       yield firstElement
5       while firstElement < maxNum:
6           mainList = filter(MultiFilter(firstElement), mainList)      ##mainList 标签始终指向主生成器，用生成器 OddList()初始化
7           firstElement = next(mainList)                   #每次只取新序列的一个元素
8           yield firstElement
```

其实我们发现，是不断筛选 mainList 这个生成器，新序列的第一个元素一定是素数，我们做二次生成，再以生成器的形式存储。剩下的就是迭代新的素数生成器了

```
1 if __name__ == '__main__':
2       for x in PrimeList(1000):
3           print(x)
```

（3）问题扩展

（4）结合项目中使用

Redis 的基本类型有哪些

题目标签：数据库

试题编号：MS001402

（1）问题分析

（2）核心问题讲解

Redis 是键值对的数据库，有 5 中主要数据类型

字符串类型（string），散列类型（hash），列表类型（list），集合类型（set），有序集合类型（zset）

（3）问题扩展

（4）结合项目中使用

tornado 的常用模块

题目标签：移动 web

试题编号：MS001423

（1）问题分析

（2）核心问题讲解

1）Core web framework

tornado.web — 包含 web 框架的大部分主要功能，包含 RequestHandler 和 Application 两个重要的类

tornado.httpserver — 一个无阻塞 HTTP 服务器的实现

tornado.template — 模版系统

tornado.escape — HTML,JSON,URLs 等的编码解码和一些字符串操作

tornado.locale — 国际化支持

2）Asynchronous networking 底层模块

tornado.ioloop — 核心的 I/O 循环

tornado.iostream — 对非阻塞式的 socket 的简单封装，以方便常用读写操作

tornado.httpclient — 一个无阻塞的 HTTP 服务器实现

tornado.netutil — 一些网络应用的实现，主要实现 TCPServer 类

3）Integration with other services

tornado.auth — 使用 OpenId 和 OAuth 进行第三方登录

tornado.database — 简单的 MySQL 服务端封装

tornado.platform.twisted — 在 Tornado 上运行为 Twisted 实现的代码

tornado.websocket — 实现和浏览器的双向通信

tornado.wsgi — 与其他 Python 网络框架/服务器的相互操作

4）Utilities

tornado.autoreload — 生产环境中自动检查代码更新

tornado.gen — 一个基于生成器的接口，使用该模块保证代码异步运行

tornado.httputil — 分析 HTTP 请求内容

tornado.options — 解析终端参数

tornado.process — 多进程实现的封装

tornado.stack_context — 用于异步环境中对回调函数的上下文保存、异常处理

tornado.testing — 单元测试

（3）问题扩展

（4）结合项目中使用

git 的常用命令

题目标签：git&svn

试题编号：MS001430

（1）问题分析

（2）核心问题讲解

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.

一般的格式是 git clone [url].

clone 下来的 repo 会以 url 最后一个斜线后面的名称命名,创建一个文件夹,如果想要指定特定的名称,可以 git clone [url] newname 指定.

git status

查询 repo 的状态.

git status -s: -s 表示 short，-s 的输出标记会有两列,第一列是对 staging 区域而言,第二列是对 working 目录而言.

git log

show commit history of a branch.

git log --oneline --number: 每条 log 只显示一行,显示 number 条.

git log --oneline --graph:可以图形化地表示出分支合并历史.

git log branchname 可以显示特定分支的 log.

git log --oneline branch1 ^branch2,可以查看在分支 1,却不在分支 2 中的提交.^表示排除这个分支(Window 下可能要给^branch2 加上引号).

git log --decorate 会显示出 tag 信息.

git log --author=[author name] 可以指定作者的提交历史.

git log --since --before --until --after 根据提交时间筛选 log.

--no-merges 可以将 merge 的 commits 排除在外.

git log --grep 根据 commit 信息过滤 log: git log --grep=keywords

默认情况下, git log --grep --author 是 OR 的关系,即满足一条即被返回,如果你想让它们是 AND 的关系,可以加上--all-match 的 option.

git log -S: filter by introduced diff.

比如: git log -SmethodName (注意 S 和后面的词之间没有等号分隔).

git log -p: show patch introduced at each commit.

每一个提交都是一个快照(snapshot),Git 会把每次提交的 diff 计算出来,作为一个 patch 显示给你看.

另一种方法是 git show [SHA].

git log --stat: show diffstat of changes introduced at each commit.

同样是用来看改动的相对信息的,--stat 比-p 的输出更简单一些.

git add

在提交之前,Git 有一个暂存区(staging area),可以放入新添加的文件或者加入新的改动. commit 时提交的改动是上一次加入到 staging area 中的改动,而不是我们 disk 上的改动.

git add .

会递归地添加当前工作目录中的所有文件.

git diff

不加参数的 git diff:

show diff of unstaged changes.

此命令比较的是工作目录中当前文件和暂存区域快照之间的差异,也就是修改之后还没有暂存起来的变化内容.

若要看已经暂存起来的文件和上次提交时的快照之间的差异,可以用:

git diff --cached 命令.

show diff of staged changes.

(Git 1.6.1 及更高版本还允许使用 git diff --staged，效果是相同的).

git diff HEAD

show diff of all staged or unstated changes.

也即比较 woking directory 和上次提交之间所有的改动.

如果想看自从某个版本之后都改动了什么,可以用:

git diff [version tag]

跟 log 命令一样,diff 也可以加上--stat 参数来简化输出.

git diff [branchA] [branchB]可以用来比较两个分支.

它实际上会返回一个由 A 到 B 的 patch,不是我们想要的结果.

一般我们想要的结果是两个分支分开以后各自的改动都是什么,是由命令:

git diff [branchA]...[branchB]给出的.

实际上它是:git diff $(git merge-base [branchA] [branchB]) [branchB]的结果.

git commit

提交已经被 add 进来的改动.

git commit -m “the commit message"

git commit -a 会先把所有已经 track 的文件的改动 add 进来,然后提交(有点像 svn 的一次提交,不用先暂存). 对于没有 track 的文件,还是需要 git add 一下.

git commit --amend 增补提交. 会使用与当前提交节点相同的父节点进行一次新的提交,旧的提交将会被取消.

git reset

undo changes and commits.

这里的 HEAD 关键字指的是当前分支最末梢最新的一个提交.也就是版本库中该分支上的最新版本.

git reset HEAD: unstage files from index and reset pointer to HEAD

这个命令用来把不小心 add 进去的文件从 staged 状态取出来,可以单独针对某一个文件操作: git reset HEAD - - filename, 这个- - 也可以不加.

git reset --soft

move HEAD to specific commit reference, index and staging are untouched.

git reset --hard

unstage files AND undo any changes in the working directory since last commit.

使用 git reset —hard HEAD 进行 reset,即上次提交之后,所有 staged 的改动和工作目录的改动都会消失,还原到上次提交的状态.

这里的 HEAD 可以被写成任何一次提交的 SHA-1.

不带 soft 和 hard 参数的 git reset,实际上带的是默认参数 mixed.

总结:

git reset --mixed id,是将 git 的 HEAD 变了(也就是提交记录变了),但文件并没有改变,(也就是 working tree 并没有改变). 取消了 commit 和 add 的内容.

git reset --soft id. 实际上，是 git reset –mixed id 后,又做了一次 git add.即取消了

commit 的内容.

git reset --hard id.是将 git 的 HEAD 变了,文件也变了.

按改动范围排序如下:

soft (commit) < mixed (commit + add) < hard (commit + add + local working)

git revert

反转撤销提交.只要把出错的提交(commit)的名字(reference)作为参数传给命令就可以了.

git revert HEAD: 撤销最近的一个提交.

git revert 会创建一个反向的新提交,可以通过参数-n 来告诉 Git 先不要提交.

git rm

git rm file: 从 staging 区移除文件,同时也移除出工作目录.

git rm --cached: 从 staging 区移除文件,但留在工作目录中.

git rm --cached 从功能上等同于 git reset HEAD,清除了缓存区,但不动工作目录树.

git clean

git clean 是从工作目录中移除没有 track 的文件.

通常的参数是 git clean -df:

-d 表示同时移除目录,-f 表示 force,因为在 git 的配置文件中, clean.requireForce=true, 如果不加-f,clean 将会拒绝执行.

git mv

git rm - - cached orig; mv orig new; git add new

git stash

把当前的改动压入一个栈.

git stash 将会把当前目录和 index 中的所有改动(但不包括未 track 的文件)压入一个栈,然后留给你一个 clean 的工作状态,即处于上一次最新提交处.

git stash list 会显示这个栈的 list.

git stash apply:取出 stash 中的上一个项目(stash@{0}),并且应用于当前的工作目录.

也可以指定别的项目,比如 git stash apply stash@{1}.

如果你在应用 stash 中项目的同时想要删除它,可以用 git stash pop

删除 stash 中的项目:

git stash drop: 删除上一个,也可指定参数删除指定的一个项目.

git stash clear: 删除所有项目.

git branch

git branch 可以用来列出分支,创建分支和删除分支.

git branch -v 可以看见每一个分支的最后一次提交.

git branch: 列出本地所有分支,当前分支会被星号标示出.

git branch (branchname): 创建一个新的分支(当你用这种方式创建分支的时候,分支是基于你的上一次提交建立的).

git branch -d (branchname): 删除一个分支.

删除 remote 的分支:

git push (remote-name) :(branch-name): delete a remote branch.

这个是因为完整的命令形式是:

git push remote-name local-branch:remote-branch

而这里 local-branch 的部分为空,就意味着删除了 remote-branch

git checkout

git checkout (branchname)

切换到一个分支.

git checkout -b (branchname): 创建并切换到新的分支.

这个命令是将 git branch newbranch 和 git checkout newbranch 合在一起的结果.

checkout 还有另一个作用:替换本地改动:

git checkout --

此命令会使用 HEAD 中的最新内容替换掉你的工作目录中的文件.已添加到暂存区的改动以及新文件都不会受到影响.

注意:git checkout filename 会删除该文件中所有没有暂存和提交的改动,这个操作是

不可逆的.

git merge

把一个分支 merge 进当前的分支.

git merge [alias]/[branch]

把远程分支 merge 到当前分支.

如果出现冲突,需要手动修改,可以用 git mergetool.

解决冲突的时候可以用到 git diff,解决完之后用 git add 添加,即表示冲突已经被 resolved.

git tag

tag a point in history as import.

会在一个提交上建立永久性的书签,通常是发布一个 release 版本或者 ship 了什么东西之后加 tag.

比如: git tag v1.0

git tag -a v1.0, -a 参数会允许你添加一些信息,即 make an annotated tag.

当你运行 git tag -a 命令的时候,Git 会打开一个编辑器让你输入 tag 信息.

我们可以利用 commit SHA 来给一个过去的提交打 tag:

git tag -a v0.9 XXXX

push 的时候是不包含 tag 的,如果想包含,可以在 push 时加上--tags 参数.

fetch 的时候,branch HEAD 可以 reach 的 tags 是自动被 fetch 下来的, tags that aren’ t reachable from branch heads will be skipped.如果想确保所有的 tags 都被包含进来,需要加上--tags 选项.

git remote

list, add and delete remote repository aliases.

因为不需要每次都用完整的 url,所以 Git 为每一个 remote repo 的 url 都建立一个别名,然后用 git remote 来管理这个 list.

git remote: 列出 remote aliases.

如果你 clone 一个 project,Git 会自动将原来的 url 添加进来,别名就叫做:origin.

git remote -v:可以看见每一个别名对应的实际 url.

git remote add [alias] [url]: 添加一个新的 remote repo.

git remote rm [alias]: 删除一个存在的 remote alias.

git remote rename [old-alias] [new-alias]: 重命名.

git remote set-url [alias] [url]:更新 url. 可以加上—push 和 fetch 参数,为同一个别名 set 不同的存取地址.

git fetch

download new branches and data from a remote repository.

可以 git fetch [alias]取某一个远程 repo,也可以 git fetch --all 取到全部 repo

fetch 将会取到所有你本地没有的数据,所有取下来的分支可以被叫做 remote branches,它们和本地分支一样(可以看 diff,log 等,也可以 merge 到其他分支),但是 Git 不允许你 checkout 到它们.

git pull

fetch from a remote repo and try to merge into the current branch.

pull == fetch + merge FETCH_HEAD

git pull 会首先执行 git fetch,然后执行 git merge,把取来的分支的 head merge 到当前分支.这个 merge 操作会产生一个新的 commit.

如果使用--rebase 参数,它会执行 git rebase 来取代原来的 git merge.

git rebase

--rebase 不会产生合并的提交,它会将本地的所有提交临时保存为补丁(patch),放在".git/rebase" 目录中,然后将当前分支更新到最新的分支尖端,最后把保存的补丁应用到分支上.

rebase 的过程中,也许会出现冲突,Git 会停止 rebase 并让你解决冲突,在解决完冲突之后,用 git add 去更新这些内容,然后无需执行 commit,只需要:

git rebase --continue 就会继续打余下的补丁.

git rebase --abort 将会终止 rebase,当前分支将会回到 rebase 之前的状态.

git push

push your new branches and data to a remote repository.

git push [alias] [branch]

将会把当前分支 merge 到 alias 上的[branch]分支.如果分支已经存在,将会更新,如果不存在,将会添加这个分支.

如果有多个人向同一个 remote repo push 代码, Git 会首先在你试图 push 的分支上运行 git log,检查它的历史中是否能看到 server 上的 branch 现在的 tip,如果本地历史中不能看到 server 的 tip,说明本地的代码不是最新的,Git 会拒绝你的 push,让你先 fetch,merge,之后再 push,这样就保证了所有人的改动都会被考虑进来.

git reflog

git reflog 是对 reflog 进行管理的命令,reflog 是 git 用来记录引用变化的一种机制,比如记录分支的变化或者是 HEAD 引用的变化.

当 git reflog 不指定引用的时候,默认列出 HEAD 的 reflog.

HEAD@{0}代表 HEAD 当前的值,HEAD@{3}代表 HEAD 在 3 次变化之前的值.

git 会将变化记录到 HEAD 对应的 reflog 文件中,其路径为.git/logs/HEAD, 分支的 reflog 文件都放在.git/logs/refs 目录下的子目录中.

(3) 问题扩展

(4) 结合项目中使用

数据库连接查询

题目标签：数据库

试题编号：MS001454

（1）问题分析

（2）核心问题讲解

1）内连接（inner join）使用比较运算符进行表间列数据的比较操作，并列出这些表中与连接条件相匹配的数据行，并组合成新的记录

2）左连接的结果包括 LEFT OUTER 子句中指定的左表的所有行，而不仅仅是连接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果行中右表的所有选择列表列均为空值

3）右连接是左连接的反向连接，将返回右表的所有行，如果右表的某行在左表中没有匹配项，左表将返回空值

（3）问题扩展

（4）结合项目中使用

爬虫实现除了使用 http 协议还有没有使用过其他协议爬取数据?

题目标签：分布式爬虫

试题编号：MS001560

robots 协议

正则表达式一个小括号什么意思?

题目标签：正则表达式

试题编号：MS001566

（1）问题分析

（2）核心问题讲解

小括号()：匹配小括号内的字符串，可以是一个，也可以是多个，常跟“|”（或）符号搭配使用，是多选结构的

示例 1：string name = "way2014"； regex：(way|zgw) result：结果是可以匹配出 way 的，因为是多选结构，小括号是匹配字符串的

示例 2：string text = "123456789"； regex：(0-9) result：结果是什么都匹配不到的，它只匹配字符串"0-9"而不是匹配数字, [0-9]这个字符组才是匹配 0-9 的数字

（3）问题扩展

（4）结合项目中使用

http 三次握手四次挥手详细说明?

题目标签：http

试题编号：MS001569

（1）问题分析

（2）核心问题讲解

三次握手

1）三次握手过程

TCP 是面向连接的，无论哪一方向另一方发送数据之前，都必须先在双方之间建立一条连接。在 TCP/IP 协议中，TCP 协议提供可靠的连接服务，连接是通过三次握手进行初始化的。三次握手的目的是同步连接双方的序列号和确认号 并交换 TCP 窗口大小信息。这就是面试中经常会被问到的 TCP 三次握手。只是了解 TCP 三次握手的 概念，对你获得一份工作是没有任何帮助的，你需要去了解 TCP 三次握手中的一些细节。

A) 第一次握手：建立连接。客户端发送连接请求报文段，将 SYN 位置为 1，Sequence Number 为 x;然后，客户端进入 SYN_SEND 状态，等待服务器的确认；

B）第二次握手：服务器收到 SYN 报文段。服务器收到客户端的 SYN 报文段，需要对这个 SYN 报文段进行确认，设置 Acknowledgment Number 为 x+1(Sequence Number+1);同时，自己还要发送 SYN 请求信息，将 SYN 位置为 1，Sequence Number 为 y;服务器端将上述所有信息放到一个报文段(即 SYN+ACK 报文段)中，一并发送给客户端，此时服务器进入 SYN_RECV 状态;

C）第三次握手：客户端收到服务器的 SYN+ACK 报文段。然后将 Acknowledgment Number 设置为 y+1，向服务器发送 ACK 报文段，这个报文段发送完毕以后，客户端和服务器端都进入 ESTABLISHED 状态，完成 TCP 三次握手。

完成了三次握手，客户端和服务器端就可以开始传送数据。

2）为什么要握手 3 次

在谢希仁的《计算机网络》中是这样说的：

为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

在书中同时举了一个例子，如下：

"已失效的连接请求报文段”的产生在这样一种情况下：client 发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达 server。本来这是一个早已失效的报文段。但 server 收到此失效的连接请求报文段后，就误认为是 client 再次发出的一个新的连接请求。于是就向 client 发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要 server 发出确认，新的连接就建立了。由于现在 client 并没有发出建立连接的请求，因此不会理睬 server 的确认，也不会向 server 发送数据。但 server 却以为新的运输连接已经建立，并一直等待 client 发来数据。这样，server 的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client 不会向 server 的确认发出确认。server 由于收不到确认，就知道 client 并没有要求建立连接。"这就很明白了，防止了服务器端的一直等待而浪费资源。

四次挥手

1）四次挥手过程

当客户端和服务器通过三次握手建立了 TCP 连接以后，当数据传送完毕，肯定是要断开 TCP 连接的啊。那对于 TCP 的断开连接，这里就有了神秘的“四次分手”。

A）第一次分手：主机 1(可以使客户端，也可以是服务器端)，设置 Sequence Number 和 Acknowledgment Number，向主机 2 发送一个 FIN 报文段;此时，主机 1 进入 FIN_WAIT_1 状态;这表示主机 1 没有数据要发送给主机 2 了;

B）第二次分手：主机 2 收到了主机 1 发送的 FIN 报文段，向主机 1 回一个 ACK 报文

段，Acknowledgment Number 为 Sequence Number 加 1;主机 1 进入 FIN_WAIT_2 状态;主机 2 告诉主机 1，我也没有数据要发送了，可以进行关闭连接了；

C）第三次分手：主机 2 向主机 1 发送 FIN 报文段，请求关闭连接，同时主机 2 进入 CLOSE_WAIT 状态；

D）第四次分手：主机 1 收到主机 2 发送的 FIN 报文段，向主机 2 发送 ACK 报文段，然后主机 1 进入 TIME_WAIT 状态;主机 2 收到主机 1 的 ACK 报文段以后，就关闭连接;此时，主机 1 等待 2MSL 后依然没有收到回复，则证明 Server 端已正常关闭，那好，主机 1 也可以关闭连接了。

2）为什么要四次挥手

那四次分手又是为何呢?TCP 协议是一种面向连接的、可靠的、基于字节流的运输层通信协议。TCP 是全双工 模式，这就意味着，当主机 1 发出 FIN 报文段时，只是表示主机 1 已经没有数据要发送了，主机 1 告诉主机 2， 它的数据已经全部发送完毕了;但是，这个时候主机 1 还是可以接受来自主机 2 的数据;当主机 2 返回 ACK 报文 段时，表示它已经知道主机 1 没有数据发送了，但是主机 2 还是可以发送数据到主机 1 的;当主机 2 也发送了 FIN 报文段时，这个时候就表示主机 2 也没有数据要发送了，就会告诉主机 1，我也没有数据要发送了，之后彼此 就会愉快的中断这次 TCP 连接。如果要正确的理解四次分手的原理，就需要了解四次分手过程中的状态变化。

FIN_WAIT_1: 这个状态要好好解释一下，其实 FIN_WAIT_1 和 FIN_WAIT_2 状态的真正含义都是表示等 待对方的 FIN 报文。而这两种状态的区别是：FIN_WAIT_1 状态实际上是当 SOCKET 在 ESTABLISHED 状态时， 它想主动关闭连接，向对方发送了 FIN 报文，此时该 SOCKET 即进入到 FIN_WAIT_1 状态。而当对方回应 ACK 报 文后，则进入到

FIN_WAIT_2 状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应 ACK 报文，所以 FIN_WAIT_1 状态一般是比较难见到的，而 FIN_WAIT_2 状态还有时常常可以用 netstat 看到。（主动方）

FIN_WAIT_2：上面已经详细解释了这种状态，实际上 FIN_WAIT_2 状态下的 SOCKET，表示半连接，也即 有一方要求 close 连接，但另外还告诉对方，我暂时还有点数据需要传送给你(ACK 信息)，稍后再关闭连接。（主动方）

CLOSE_WAIT：这种状态的含义其实是表示在等待关闭。怎么理解呢?当对方 close 一个 SOCKET 后发送 FIN 报文给自己，你系统毫无疑问地会回应一个 ACK 报文给对方，此时则进入到 CLOSE_WAIT 状态。接下来呢，实 际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以 close 这个 SOCKET，发送 FIN 报文给对方，也即关闭连接。所以你在 CLOSE_WAIT 状态下，需要完成的事情是等待你去关闭连接。(被动方)

LAST_ACK：这个状态还是比较容易好理解的，它是被动关闭一方在发送 FIN 报文后，最后等待对方的 ACK 报 文。当收到 ACK 报文后，也即可以进入到 CLOSED 可用状态了。(被动方)

TIME_WAIT：表示收到了对方的 FIN 报文，并发送出了 ACK 报文，就等 2MSL 后即可回到 CLOSED 可用状态了。 如果 FINWAIT1 状态下，收到了对方同时带 FIN 标志和 ACK 标志的报文时，可以直接进入到 TIME_WAIT 状态，而无 须经过 FIN_WAIT_2 状态。(主动方)

CLOSED：表示连接中断。

（3）问题扩展

（4）结合项目中使用

ajax 动态页面不用 selenium 怎么进行抓取?

题目标签：js 加密

试题编号：MS001574

（1）问题分析

（2）核心问题讲解

寻找 ajax 数据接口，使用 requests 伪装浏览器进行请求获取数据，如果如果有 js 加密还需要对 js 解密

（3）问题扩展

（4）结合项目中使用

用过抓包工具吗，都有哪些??

题目标签：抓包

试题编号：MS001776

（1）问题分析

（2）核心问题讲解

1）Wireshark

2）Flidder

3）Charles

（3）问题扩展

（4）结合项目中使用

爬虫经常会去查看数据库吗?

题目标签：爬虫

试题编号：MS001783

（1）问题分析

（2）核心问题讲解

目前都是基于 scrapy 框架的增量式爬虫，不需要经常查看数据库，去查看数据库的目的就是为了检测爬取的数据是否重复，scrapy_Redis 框架自带去重功能，经常查看数据库会降低爬虫效率，不利于快速抓取数据。

（3）问题扩展

（4）结合项目中使用

Python 的垃圾回收机制?

题目标签：垃圾回收

试题编号：MS001795

（1）问题分析

（2）核心问题讲解

import sys sys.getrefcount()查看引用计数

字符串中间有空格！？等会重新创建新的字符串

总结

1）小整数[-5,257)共用对象，常驻内存，不会被释放。

2）单个字符共用对象，常驻内存。

3）单个单词，不可修改，默认开启 intern 机制，共用对象，引用计数为 0，则销毁。

4）大整数不共用内存，引用计数为 0，销毁。

5）数值类型和字符串类型在 Python 中都是不可变的，这意味着你无法修改这个对象的值，每次对变量的修改，实际上是创建一个新的对象。

Garbage collection(GC 垃圾回收)

Python 采用的是引用计数机制为主，标记-清除和分代收集(隔代回收、分代回收)两种机制为辅的策略

引用计数机制的优点：

1）简单

2）实时性：一旦没有引用，内存就直接释放了。不用像其他机制等到特定时机。实时性还带来一个好处：处理回收内存的时间分摊到了平时。

引用计数机制的缺点：

维护引用计数消耗资源

循环引用，解决不了

gc 模块，垃圾回收机制

导致引用计数+1 的情况

1）对象被创建，例如 a = "hello"

2）对象被引用，例如 b=a

3）对象被作为参数，传入到一个函数中，例如 func(a)

4）对象作为一个元素，存储在容器中，例如 list1=[a,a]

常用函数

1）gc.set_debug(flags) 设置 gc 的 debug 日志，一般设置为 gc.DEBUG_LEAK

2）gc.collect([generation]) 显式进行垃圾回收，可以输入参数，0 代表只检查零代的对象，1 代表检查零，一代的对象，2 代表检查零，一，二代的对象，如果不传参数，执行一个 full collection，也就是等于传 2。 在 Python2 中返回不可达（unreachable objects）对象的数目

3）gc.get_threshold() 获取的 gc 模块中自动执行垃圾回收的频率。

4) gc.set_threshold(threshold0[, threshold1[, threshold2]) 设置自动执行垃圾回收的频率。

5）gc.get_count() 获取当前自动执行垃圾回收的计数器，返回一个长度为 3 的列表

Python 的 GC 模块主要运用了引用计数来跟踪和回收垃圾。在引用计数的基础上，还可以通过“标记-清除”解决容器对象可能产生的循环引用的问题。通过分代回收以空间换取时间进一步提高垃圾回收的效率。

标记-清除

标记-清除的出现打破了循环引用，也就是它只关注那些可能会产生循环引用的对象

缺点：该机制所带来的额外操作和需要回收的内存块成正比。

一旦这个差异累计超过某个阈值(700,10,10)，则 Python 的收集机制就启动了，并且触发上边所说到的零代算法释放“浮动的垃圾”，并且将剩下的对象移动到一代列表。随着时间的推移，程序所使用的对象逐渐从零代列表移动到一代列表。通过这种方法，你的代码所长期使用的对象，那些你的代码持续访问的活跃对象，会从零代链表转移到一代再转移到二代。通过不同的阈值设置，Python 可以在不同的时间间隔处理这些对象。Python 处理零代最为频繁，其次是一代然后才是二代。

隔代回收

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。

dir(__builtins__)查看内建属性

__getattribute__内建属性。属性访问拦截器(方法和属性都可以被拦截)，可以返回一个值：以后不要在__getattribute__方法中调用 self.xxxx 会引起递归时程序死掉

map 函数会根据提供的函数对指定序列做映射返回值是列表

map(function, sequence[, sequence, ...]) -> list

1）function:是一个函数

2）sequence:是一个或多个序列,取决于 function 需要几个参数

3）返回值是一个 list

filter 函数 Python3 返回的是生产器 filter 函数会对指定序列执行过滤操作

filter(function or None, sequence) -> list, tuple, or string

1）function:接受一个参数，返回布尔值 True 或 False

2）sequence:序列可以是 str，tuple，list

list(filter(lambda x x%2==0,[1,2,3,4,5,6])---->[2,4,6]

sorted 函数-排序

sorted(iterable, reverse=False) --> new sorted list

functools 模块 import functools

partial 函数（偏函数）把一个函数的某些参数设置默认值，返回一个新的函数，调用这个新函数会更简单。

wraps 函数使用装饰器时，让外界看被装饰的函数时内容一致。

例如，被装饰后的函数其实已经是另外一个函数了（函数名等函数属性会发生改变）。

functools.wraps(func)

（3）问题扩展

（4）结合项目中使用

请解释线程和协程的区别，你曾经在项目中是如何使用它们的，以及效果？

题目标签：多线程

试题编号：MS001804

（1）问题分析

（2）核心问题讲解

1）线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。线程间通信主要通过共享内存，上下文切换很快，资源开销较少，但相比进程不够稳定容易丢失数据。

2）协程

协程是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

（3）问题扩展

（4）结合项目中使用

请解释一下 Python 的线程锁 Lock 和 Rlock 的区别，以及你曾经在项目中是如何使用的?

题目标签：多线程

试题编号：MS001809

（1）问题分析

（2）核心问题讲解

从原理上来说：在同一线程内，对 RLock 进行多次 acquire()操作，程序不会阻塞。

资源总是有限的，程序运行如果对同一个对象进行操作，则有可能造成资源的争用，甚至导致死锁 也可能导致读写混乱

（3）问题扩展

（4）结合项目中使用

如何用 Python 生成唯一的 ID（不同的程序之间也要唯一，长度不得超过 32 个字节），这些 ID 是如何排序的?

题目标签：架构设计

试题编号：MS001814

（1）问题分析

（2）核心问题讲解

1）uuid1()——基于时间戳

由 MAC 地址、当前时间戳、随机数生成。可以保证全球范围内的唯一性，

但 MAC 的使用同时带来安全性问题，局域网中可以使用 IP 来代替 MAC。

2）uuid2()——基于分布式计算环境 DCE（Python 中没有这个函数）

算法与 uuid1 相同，不同的是把时间戳的前 4 位置换为 POSIX 的 UID。

实际中很少用到该方法。

3）uuid3()——基于名字的 MD5 散列值

通过计算名字和命名空间的 MD5 散列值得到，保证了同一命名空间中不同名字的唯一性，

和不同命名空间的唯一性，但同一命名空间的同一名字生成相同的 uuid。

4）uuid4()——基于随机数

由伪随机数得到，有一定的重复概率，该概率可以计算出来。

5) uuid5()——基于名字的 SHA-1 散列值算法与 uuid3 相同，不同的是使用 Secure Hash Algorithm 1 算法

（3）问题扩展

（4）结合项目中使用

windows 下连接 Linux 的工具有哪些?

题目标签：系统运维

试题编号：MS001827

putty、SecureCRT、xshell、winscp

get 和 post 的区别?

题目标签：http

试题编号：MS001828

（1）问题分析

（2）核心问题讲解

1）大小：get 为 2K

post 默认 8M

2）方式：get 通过 http

Post 往往通过表单

3）安全：get 明文传输

Post 较安全

4）地址：get 通过 url

Post 通过请求头

（3）问题扩展

（4）结合项目中使用

Python 垃圾回收机制是什么?

题目标签：垃圾回收

试题编号：MS002005

（1）问题分析

（2）核心问题讲解

Python 里也同 java 一样采用了垃圾收集机制，不过不一样的是：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。引用计数机制的优点：

1）简单

2）实时性：一旦没有引用，内存就直接释放了。不用像其他机制等到特定时机。实时性还带来一个好处：处理回收内存的时间分摊到了平时。

引用计数机制的缺点：

1）维护引用计数消耗资源

2）循环引用

gc 模块的自动垃圾回收机制，这个机制的主要作用就是发现并处理不可达的垃圾对象。

垃圾回收=垃圾检查+垃圾回收

（3）问题扩展

（4）结合项目中使用

介绍下 except 的作用和用法?

题目标签：异常处理

试题编号：MS002008

（1）问题分析

（2）核心问题讲解

try:

    do something

except:

    handle except

执行 try 下的语句，如果引发异常，则执行过程会跳到第一个 except 语句。如果与 except 中定义的异常与引发的异常匹配，则执行该 except 中的语句。

（3）问题扩展

（4）结合项目中使用

Python 中如何实现多线程?

题目标签：多线程

试题编号：MS002014

（1）问题分析

（2）核心问题讲解

Python 中使用线程有两种方式：函数或者用类来包装线程对象。函数式：调用 thread 模块中的 start_new_thread()函数来产生新线程

（3）问题扩展

（4）结合项目中使用

正在运行的 Python 程序，ctrl+c 中断，会产生什么类的异常?

题目标签：异常处理

试题编号：MS002015

KeyboardInterrupt

编写一个 Python 脚本，删除一个目录及其下面的文件个文件夹?

题目标签：文件操作

试题编号：MS002017

（1）问题分析

（2）核心问题讲解

```
import os
CUR_PATH =
r'C:\Users\shenping\PycharmProjects\Shenping_TEST\day_5\Testfolder'
def del_file(path):
    ls = os.listdir(path)
    for i in ls:
        c_path = os.path.join(path, i)
        if os.path.isdir(c_path):
            del_file(c_path)
        else:
            os.remove(c_path)
del_file(CUR_PATH)
```

（3）问题扩展

（4）结合项目中使用

请列举出 OSI 模型中网络层，传输层，应用层的常见协议？

题目标签：网络模型

试题编号：MS002021

应用层：TFTP（文件传输）、HTTP(超文本传输协议)、DNS（域名解析）、SMTP（邮件传输）

传输层：TCP、UDP

网络层：IP

在 RHEL 系统中，要将 ifocnfig 命令重定向到 out.txt 文件中，该如何?

题目标签：重定向

试题编号：MS002025

ifocnfig >out.txt

对名为 h3c 的文件用 chmod 551 h3c 进行修改，则该文件的许可权为?

题目标签：权限

试题编号：MS002027

-r-xr-x—x

在/etc 目录下，设置 linux 环境中特性的重要文件为?

题目标签：文件操作

试题编号：MS002029

profile

kill 命令想指定进程发出待定的信号，什么信号会强制杀死进程?

题目标签：Linux

试题编号：MS002031

信号 0

某个字段希望存放电话号码，改字段应该是什么类型<?

题目标签：数据库设计

试题编号：MS002033

varchar

数据库优化有那些思路？请列出你常用的数据库?

题目标签：数据库设计

试题编号：MS002042

索引、分库分表分区、数据库引擎、预处理、MySQL like 查询、读写分离。

MySQL 数据库

一张数据库，每天有百万级数据增长量，怎么保证 1 年后表查询的稳定?

题目标签：数据库设计

试题编号：MS002045

数据库设计方面、SQL 语句方面、硬件调整性能、使用存储过程、应用程序结构和算法

下面这段代码的输出结果将是什么？请解释？

```
class Parent(object):
  x = 1
class Child1(Parent):
    pass
class Child2(Parent):
  pass
print Parent.x, Child1.x, Child2.x
 Child1.x = 2
print parent.x, Child1.x, Child2.x
parent.x = 3
print Parent.x, Child1.x, Child2.x
```

题目标签：面向对象

试题编号：MS001517

（1）问题分析

（2）核心问题讲解

111

121

323

答案的关键是，在 Python 中，类变量在内部是以字典的形式进行传递。

如果一个变量名没有在当前类下的字典中发现。则在更高级的类（如它的父类）中尽心搜索直到引用的变量名被找到。（如果引用变量名在自身类和更高级类中没有找到，将会引发一个属性错误。）

因此,在父类中设定 x = 1,让变量 x 类（带有值 1）能够在其类和其子类中被引用到。这就是为什么第一个打印语句输出结果是 1 1 1

因此，如果它的任何一个子类被覆写了值（例如说，当我们执行语句 Child.x = 2），这个值只在子类中进行了修改。这就是为什么第二个打印语句输出结果是 1 2 1

最终，如果这个值在父类中进行了修改，（例如说，当我们执行语句 Parent.x = 3），这个改变将会影响那些还没有覆写子类的值（在这个例子中就是 Child2）这就是为什么第三打印语句输出结果是 3 2 3

（3）问题扩展

（4）结合项目中使用

在系统中需要控制商品对公司是否可见，请问公司和商品的数据关系怎么设计？

题目标签：数据库设计

试题编号：MS002048

（1）问题分析：

考官主要想考察学员对于数据库的设计和操作，有没有数据关系对设计经验

（2）核心答案讲解：

商品表中需有公司ID与公司表对应，并使用一个字段标记是否对公司可见，值为tinyint即可，使用0，1表示

（3）问题扩展：

数据库中的表设计？

（4）结合项目中使用：

无

何为范式，有几种，试列举并简要描述。

题目标签：数据库设计

试题编号：MS002399

（1）问题分析：

无

（2）核心答案讲解：

范式（数据库设计范式，数据库的设计范式）是符合某一种级别的关系模式的集合。构造数据库必须遵循一定的规则。在关系数据库中，这种规则就是范式。关系数据库中的关系必须满足一定的要求，即满足不同的范式。

目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、Boyce-Codd 范式（BCNF）、第四范式（4NF）和第五范式（5NF）。

满足最低要求的范式是第一范式（1NF）。在第一范式的基础上进一步满足更多要求的

称为第二范式（2NF），其余范式以次类推。一般说来，数据库只需满足第三范式（3NF）就行了。

（3）问题扩展：

无

（4）结合项目中使用：

无

E-R 模型的组成包括哪些元素（） （） （）

题目标签：数据库设计

试题编号：MS002402

实体、属性、联系

创建一个 Person 表，包含信息有名称、身份证号码、性别、出生日期、家庭住址（家庭住址包含年省、市、区（县）、详细地址），支持姓名和身份证号码的快速查找，请写出合理 create sql 语句或 djnago orm 模型（请在此处写出答案，sql，orm 二选一）

题目标签：数据库设计

试题编号：MS002060

CREATE TABLE person (

```
ID int NOT NULL AUTO_INCREMENT,

personID char(18) NOT NULL,

personName varchar(20),

province int(2),

city int(2),

area int(2),

address varchar(100)，PRIMARY KEY(ID),UNIQUE(personID))ENGINE=InnoDB

DEFAULT CHARSET=utf8";
```

为什么爬虫要用 mongodb 存储不直接存储到 MySQL?

题目标签：数据库选择

试题编号：MS002108

（1）问题分析：

无

（2）核心答案讲解：

MySQL 关系型数据库

在不同的引擎上有不同的存储方式。

查询语句是使用传统的 sql 语句，拥有较为成熟的体系，成熟度很高。

开源数据库的份额在不断增加，MySQL 的份额页在持续增长。

缺点就是在海量数据处理的时候效率会显著变慢。

Mongodb

非关系型数据库(nosql),属于文档型数据库。先解释一下文档的数据库，即可以存放xml、json、bson 类型系那个的数据。这些数据具备自述性（self-describing），呈现分层的树状数据结构。数据结构由键值(key=>value)对组成。

存储方式：虚拟内存+持久化。

查询语句：是独特的 Mongodb 的查询方式。

适合场景：事件的记录，内容管理或者博客平台等等。

架构特点：可以通过副本集，以及分片来实现高可用。

数据处理：数据是存储在硬盘上的，只不过需要经常读取的数据会被加载到内存中，将数据存储在物理内存中，从而达到高速读写。

成熟度与广泛度：新兴数据库，成熟度较低，Nosql 数据库中最为接近关系型数据库，比较完善的 DB 之一，适用人群不断在增长。

优势：

快速！在适量级的内存的 Mongodb 的性能是非常迅速的，它将热数据存储在物理内存中，使得热数据的读写变得十分快，

高扩展！

自身的 Failover 机制！

json 的存储格式！

（3）问题扩展：

无

（4）结合项目中使用：

无

请简述前端技术中的 id 和 class 的区别?

试题编号：MS002054

题目标签：Html

（1）问题分析：

考官主要想考察学员对于前端技术的掌握

（2）核心答案讲解：

商品表中需有公司 ID 与公司表对应，并使用一个字段标记是否对公司可见，值为 tinyint 即可，使用 0，1 表示

（3）问题扩展：

数据库中的表设计?

（4）结合项目中使用：

无

http 三次握手四次回首详细说明。

题目标签：http 协议

试题编号：MS002083

（1）问题分析：

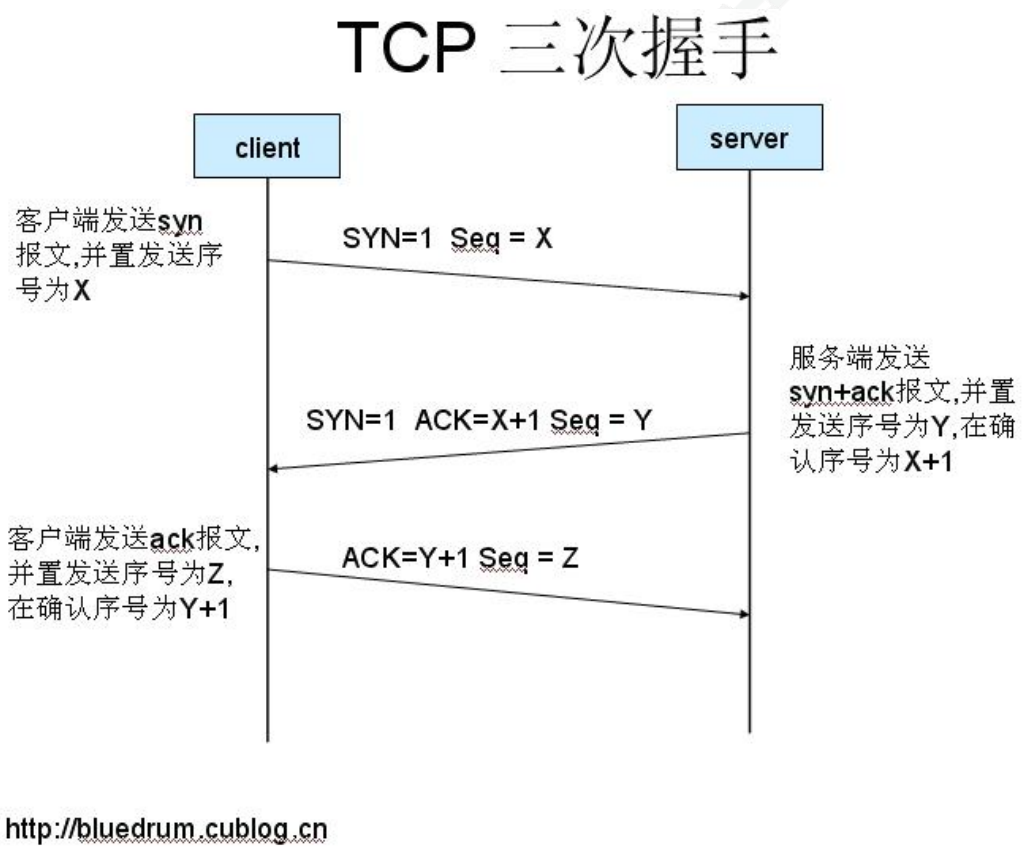
无

（1）核心答案讲解：

TCP 三次握手

所谓三次握手(Three-way Handshake)，是指建立一个 TCP 连接时，需要客户端和服务器总共发送 3 个包。

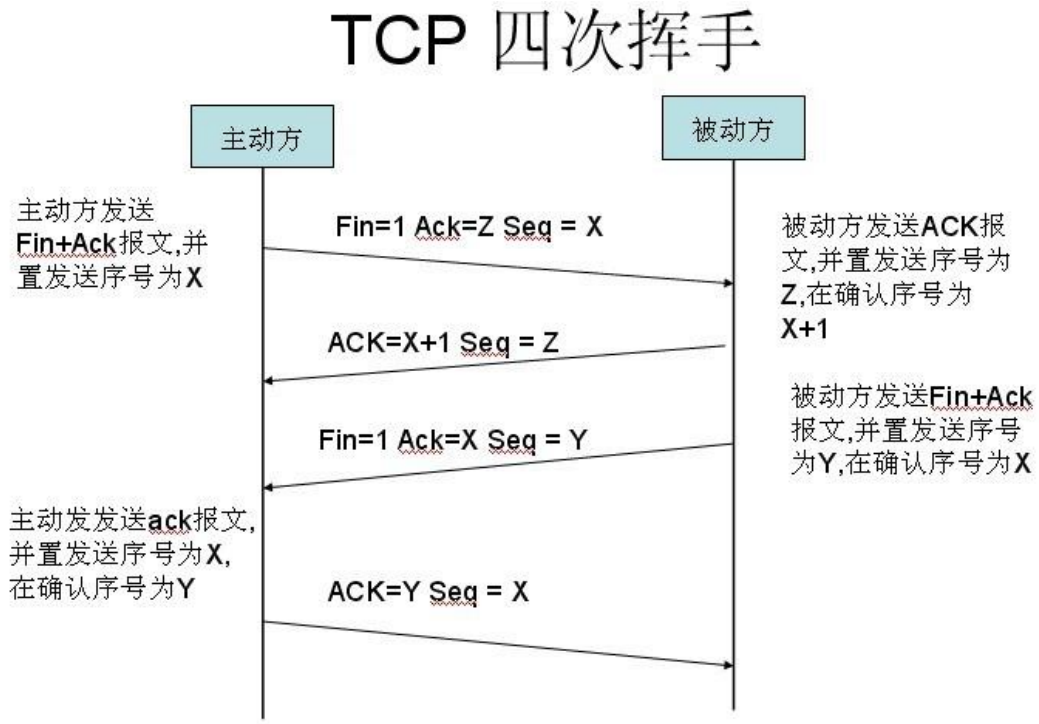
三次握手的目的是连接服务器指定端口，建立 TCP 连接,并同步连接双方的序列号和确认号并交换 TCP 窗口大小信息.在 socket 编程中，客户端执行 connect()时。将触发三次握手。



TCP 四次挥手

TCP 的连接的拆除需要发送四个包，因此称为四次挥手(four-way handshake)。客户端或服务器均可主动发起挥手动作，在 socket 编程中，任何一方执行 close()操作即可产生

挥手操作。



http://bluedrum.cublog.cn

（3）问题扩展：

无

（4）结合项目中使用：

无

如何实现多线程爬虫？如何实现多线程爬虫？

题目标签：爬虫,多线

试题编号：MS002086

（1）问题分析：

无

（2）核心答案讲解：

爬虫的基本步骤分为：获取，解析，存储。假设这里获取和存储为 io 密集型（访问网络和数据存储），解析为 cpu 密集型。那么在设计多线程爬虫时主要有两种方案：第一种方案是一个线程完成三个步骤，然后运行多个线程；第二种方案是每个步骤运行一个多线程，比如 N 个线程进行获取，1 个线程进行解析（多个线程之间切换会降低效率），N 个线程进行存储。

（3）问题扩展：

无

（4）结合项目中使用：

无

爬虫项目在哪里运行?

题目标签：爬虫运行环境

试题编号：MS002109

爬虫一般运行在公司的服务器上

Linux，下递归查找当前目录下名称包含 honglian 的文件，命令为：

题目标签：系统运维

试题编号：MS002102

find . -name "honglian"

将/etc/passwd 文件复制到/home 目录下，命令为：

题目标签：系统运维

试题编号：MS002103

cp /etc/passwd /home

linux 一个 test 文件，对其执行 chmod 750 test 达到什么效果

题目标签：系统运维

试题编号：MS002104

（1）问题分析：

无

（2）核心答案讲解：

750 中 7 是表示的属主(文件所有者)具有读、写和执行的权限；5 是文件所有者所在的组的权限,5 表示有读和执行的权限,表示文件属主所在组的同组人有读和执行的权限,没有对文件写入的权限.0 其他人没有权限

（3）问题扩展：

无

（4）结合项目中使用：

无

操作系统 windows 下查看 80 端口是否被占用?

题目标签：系统运维

试题编号：MS002106

netstat -aon|findstr "80"

在 Python 中如何拷贝一个对象?

题目标签：深浅拷贝

试题编号：MS002129

（1）问题分析：

无

核心答案讲解：

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。

In [168]: a

Out[168]: [1, 2, 3]

In [169]: b=a

In [170]: a.append(4)

In [171]: a

Out[171]: [1, 2, 3, 4]

In [172]: b

Out[172]: [1, 2, 3, 4]

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）

In [187]: import copy#copy 浅拷贝，没有拷贝子对象，所以原始数据改变，子对象会改变

In [188]: a=[1,2,3,4,['a','b']]

In [189]: c=copy.copy(a)

In [190]: a.append(5)

In [191]: a

Out[191]: [1, 2, 3, 4, ['a', 'b'], 5]

```
In [192]: c

Out[192]: [1, 2, 3, 4, ['a', 'b']]


In [193]: a[4].append('c')


In [194]: a

Out[194]: [1, 2, 3, 4, ['a', 'b', 'c'], 5]


In [195]: c

Out[195]: [1, 2, 3, 4, ['a', 'b', 'c']]
```

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）

```
In [196]: import copy


In [197]: a=[1,2,3,4,['a','b']]


In [198]: c=copy.deepcopy(a)


In [199]: a.append(5)
```

```
In [200]: a

Out[200]: [1, 2, 3, 4, ['a', 'b'], 5]


In [201]: c

Out[201]: [1, 2, 3, 4, ['a', 'b']]


In [202]: a[4].append('c')


In [203]: a

Out[203]: [1, 2, 3, 4, ['a', 'b', 'c'], 5]


In [204]: c

Out[204]: [1, 2, 3, 4, ['a', 'b']]
```

（3）问题扩展：

无

（4）结合项目中使用：

无

什么是协程?

题目标签：协程

试题编号：MS002276

（1）问题分析：

无

（2）核心答案讲解：

什么是协程

你可能已经听过『进程』和『线程』这两个概念。

进程就是二进制可执行文件在计算机内存里的一个运行实例，就好比你的.exe 文件是个类，进程就是 new 出来的那个实例。

进程是计算机系统进行资源分配和调度的基本单位（调度单位这里别纠结线程进程的），每个 CPU 下同一时刻只能处理一个进程。

所谓的并行，只不过是看起来并行，CPU 事实上在用很快的速度切换不同的进程。

进程的切换需要进行系统调用，CPU 要保存当前进程的各个信息，同时还会使 CPUCache 被废掉。

所以进程切换不到非不得已就不做。

那么怎么实现『进程切换不到非不得已就不做』呢?

首先进程被切换的条件是：进程执行完毕、分配给进程的 CPU 时间片结束，系统发生中断需要处理，或者进程等待必要的资源（进程阻塞）等。你想下，前面几种情况自然没有什么话可说，但是如果是在阻塞等待，是不是就浪费了。

其实阻塞的话我们的程序还有其他可执行的地方可以执行，不一定要傻傻的等！所以就有了线程。

线程简单理解就是一个『微进程』，专门跑一个函数（逻辑流）。所以我们就可以在编

写程序的过程中将可以同时运行的函数用线程来体现了。

线程有两种类型，一种是由内核来管理和调度。

我们说，只要涉及需要内核参与管理调度的，代价都是很大的。这种线程其实也就解决了当一个进程中，某个正在执行的线程遇到阻塞，我们可以调度另外一个可运行的线程来跑，但是还是在同一个进程里，所以没有了进程切换。

还有另外一种线程，他的调度是由程序员自己写程序来管理的，对内核来说不可见。这种线程叫做『用户空间线程』。

协程可以理解就是一种用户空间线程。

协程，有几个特点：

协同，因为是由程序员自己写的调度策略，其通过协作而不是抢占来进行切换

在用户态完成创建，切换和销毁

从编程角度上看，协程的思想本质上就是控制流的主动让出（yield）和恢复（resume）机制 generator 经常用来实现协程。

（3）问题扩展：

无

（4）结合项目中使用：

无

闭包的作用?

题目标签：闭包

试题编号：MS002285

（1）问题分析：

无

（2）核心答案讲解：

闭包：装饰器的本质也是闭包

“闭包”的本质就是函数的嵌套定义，即在函数内部再定义函数，如下所示。

“闭包”有两种不同的方式，第一种是在函数内部就“直接调用了”；第二种是“返回一个函数名称”。

1）第一种形式——直接调用

```
def Maker(name):
num=100
def func1(weight,height,age):
weight+=1
height+=1
age+=1
print(name,weight,height,age)
func1(100,200,300) #在内部就直接调用“内部函数”
Maker('feifei') #调用外部函数，输出 feifei 101 201 301
```

2）第二种形式——返回函数名称

```
def Maker(name):
num=100
```

```
def func1(weight,height,age):
weight+=1
height+=1
age+=1
print(name,weight,height,age)
return func1 #此处不直接调用，而是返回函数名称（Python 中一切皆对象）
maker=Maker('feifei') #调用包装器
maker(100,200,300) #调用内部函数
```

3）“闭包”的作用——保存函数的状态信息，使函数的局部变量信息依然可以保存下来，如下。

```
ef Maker(step): #包装器
num=1
def fun1(): #内部函数
nonlocal num #nonlocal 关键字的作用和前面的 local 是一样的，如果不使用该关键字，则不能再内部函数改变“外部变量”的值
num=num+step #改变外部变量的值（如果只是访问外部变量，则不需要适用 nonlocal）
print(num)
return fun1
#=====================================
j=1
```

```
func2=Maker(3) #调用外部包装器
while(j<5):
func2() #调用内部函数 4 次 输出的结果是 4、7、10、13
j+=1
```

从上面的例子可以看出，外部装饰器函数的局部变量 num=1、以及调用装饰器 Maker(3)时候传入的参数 step=3 都被记忆了下来，所以才有 1+3=4、4+3=7、7+3=10、10+3=13。

从这里可以看出，Maker 函数虽然调用了，但是它的局部变量信息却被保存了下来，这就是“闭包”的最大的作用——保存局部信息不被销毁。

（3）问题扩展：

无

（4）结合项目中使用：

无

多任务线程，进程，协程在实际项目中哪里可以用到

题目标签：Python 精选面试题-答案

试题编号：MS006482，删除：MS006876、MS006572

（1）问题分析：

首先我们需要弄清楚进程线程写成的区别，举个例子：

有一个老板想要开个工厂进行生产某件商品（例如剪子）

他需要花一些财力物力制作一条生产线，这个生产线上有很多的器件以及材料这些所有

的为了能够生产剪子而准备的资源称之为：进程

只有生产线是不能够进行生产的，所以老板的找个工人来进行生产，这个工人能够利用这些材料最终一步步的将剪子做出来，这个来做事情的工人称之为：线程

这个老板为了提高生产率，想到 3 种办法：

1）在这条生产线上多招些工人，一起来做剪子，这样效率是成倍增长，即单进程 多线程方式

2）老板发现这条生产线上的工人不是越多越好，因为一条生产线的资源以及材料毕竟有限，所以老板又花了些财力物力购置了另外一条生产线，然后再招些工人这样效率又再一步提高了，即多进程 多线程方式

3）老板发现，现在已经有了很多条生产线，并且每条生产线上已经有很多工人了（即程序是多进程的，每个进程中又有多个线程），为了再次提高效率，老板想了个损招，规定：如果某个员工在上班时临时没事或者再 等待某些条件(比如等待另一个工人生产完谋道工序 之后他才能再次工作），那么这个员工就利用这个时间去做其它的事情，那么也就是说：如果一个线程等待某些条件，可以充分利用这个时间去做其它事情，其实这就是：协程方式。

（2）核心答案讲解：

需要频繁创建销毁的优先使用线程，线程的切换速度快，需要大量计算，频繁切换的时候使用线程，还有耗时的操作使用线程能提高程序的响应；在 CPU 系统的效率使用上线程更高，多核分布用线程,多机分布用进程；并行操作时使用线程，进程开发稳定性强多线程用于无序的，资源开销小的情况；切换频繁的时候用协程，资源分配少的情况下用协程，需要开发稳定使用线程。

（3）问题扩展：

无

（4）结合项目中使用：

无

Navicat 这款操作软件怎么用，什么是 msql 的触发器

题目标签：Navicat,MySQL 触发器

试题编号：MS006575

（1）问题分析：

考官主要想考察学员对于数据库的设计和操作，对于数据库操作有没有经验

（2）核心答案讲解：

Navicat 是一套快速、可靠并价格相当便宜的数据库管理工具，让用户用直观化的图形界面来操作数据库。新版本的 Navicat Premium 支持 MySQL、oracle、postgresql、sqllite、以及 SQL server 等多种数据库。

常用的功能：

数据库设计，表关系设计

ER 图表查看

数据库查询展示

SQL 编辑器、sql 简化和美化

历史日志产看

数据库备份，导入导出（支持多种格式）

触发器（trigger）：监视某种情况，并触发某种操作，它是提供给程序员和数据分析员来保证数据完整性的一种方法，它是与表事件相关的特殊的存储过程，它的执行不是由程序调用，也不是手工启动，而是由事件来触发，例如当对一个表进行操作（ insert，delete，update）时就会激活它执行。

触发器经常用于加强数据的完整性约束和业务规则等。 触发器创建语法四要素：

1）监视地点(table)

2）监视事件(insert/update/delete)

3）触发时间(after/before)

4）触发事件(insert/update/delete)

首先在 Navicat for MySQL 找到需要建立触发器对应的表，右键“设计表”，然后创建触发器。

（3）问题扩展：

触发器应该尽量避免使用

1）存储过程和触发器二者是有很大的联系的，我的一般理解就是触发器是一个隐藏的存储过程，因为它不需要参数，不需要显示调用，往往在你不知情的情况下已经做了很多操作。从这个角度来说，由于是隐藏的，无形中增加了系统的复杂性，非 DBA 人员理解起来数据库就会有困难，因为它不执行根本感觉不到它的存在。

2）再有，涉及到复杂的逻辑的时候，触发器的嵌套是避免不了的，如果再涉及几个存储过程，再加上事务等等，很容易出现死锁现象，再调试的时候也会经常性的从一个触发器转到另外一个，级联关系的不断追溯，很容易使人头大。其实，从性能上，触发器并没有提升多少性能，只是从代码上来说，可能在 coding 的时候很容易实现业务。

3）在编码中存储过程显示调用很容易阅读代码，触发器隐式调用容易被忽略。

4）存储过程的致命伤在于移植性，存储过程不能跨库移植，比如事先是在 MySQL 数据库的存储过程，考虑性能要移植到 oracle 上面那么所有的存储过程都需要被重写一遍。

（4）结合项目中使用：

我们在做项目中一般情况下避免使用触发器

触发器和存储过程本身难以开发和维护，不能高效移植。触发器完全可以用事务替代。存储过程可以用后端脚本替代。

解释下 http 协议的含义

题目标签：http 协议

试题编号：MS002289

（1）问题分析：

无

（2）核心答案讲解：

HTTP 是一个属于应用层的面向对象的协议，HTTP 协议一共有五大特点：

1）支持客户/服务器模式；

2）简单快速；

3）灵活；

4）无连接；

5）无状态。

无连接

无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

早期这么做的原因是 HTTP 协议产生于互联网，因此服务器需要处理同时面向全世界数十万、上百万客户端的网页访问，但每个客户端（即浏览器）与服务器之间交换数据的间歇性较大（即传输具有突发性、瞬时性），并且网页浏览的联想性、发散性导致两次传送的数据关联性很低，大部分通道实际上会很空闲、无端占用资源。因此 HTTP 的设计者有意利用这种特点将协议设计为请求时建连接、请求完释放连接，以尽快将资源释放出来服务其他客户端。

随着时间的推移，网页变得越来越复杂，里面可能嵌入了很多图片，这时候每次访问图片都需要建立一次 TCP 连接就显得很低效。后来，Keep-Alive 被提出用来解决这效率低的问题。

Keep-Alive 功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接。市场上的大部分 Web 服务器，包括 iPlanet、IIS 和 Apache，都支持 HTTP Keep-Alive。对于提供静态内容的网站来说，这个功能通常很有用。但是，对于负担较重的网站来说，这里存在另外一个问题：虽然为客户保留打开的连接有一定的好处，但它同样影响了性能，因为在处理暂停期间，本来可以释放的资源仍旧被占用。当 Web 服务器和应用服务器在同一台机器上运行时，Keep-Alive 功能对资源利用的影响尤其突出。

这样一来，客户端和服务器之间的 HTTP 连接就会被保持，不会断开（超过 Keep-Alive 规定的时间，意外断电等情况除外），当客户端发送另外一个请求时，就使用这条已经建立的连接。

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。

HTTP 是一个无状态协议，这意味着每个请求都是独立的，Keep-Alive 没能改变这个结果。

缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

HTTP 协议这种特性有优点也有缺点，优点在于解放了服务器，每一次请求“点到为止”不会造成不必要连接占用，缺点在于每次请求会传输大量重复的内容信息。

客户端与服务器进行动态交互的 Web 应用程序出现之后，HTTP 无状态的特性严重阻碍了这些应用程序的实现，毕竟交互是需要承前启后的，简单的购物车程序也要知道用户到底在之前选择了什么商品。于是，两种用于保持 HTTP 连接状态的技术就应运而生了，一个是 Cookie，而另一个则是 Session。

Cookie 可以保持登录信息到用户下次与服务器的会话，换句话说，下次访问同一网站时，用户会发现不必输入用户名和密码就已经登录了（当然，不排除用户手工删除 Cookie）。而还有一些 Cookie 在用户退出会话的时候就被删除了，这样可以有效保护个人隐私。

Cookies 最典型的应用是判定注册用户是否已经登录网站，用户可能会得到提示，是否在下一次进入此网站时保留用户信息以便简化登录手续，这些都是 Cookies 的功用。另一个重要应用场合是“购物车”之类处理。用户可能会在一段时间内在同一家网站的不同页面中选择不同的商品，这些信息都会写入 Cookies，以便在最后付款时提取信息。

与 Cookie 相对的一个解决方案是 Session，它是通过服务器来保持状态的。

当客户端访问服务器时，服务器根据需求设置 Session，将会话信息保存在服务器上，同时将标示 Session 的 SessionId 传递给客户端浏览器，浏览器将这个 SessionId 保存在内存中，我们称之为无过期时间的 Cookie。浏览器关闭后，这个 Cookie 就会被清掉，它不会存在于用户的 Cookie 临时文件。

以后浏览器每次请求都会额外加上这个参数值，服务器会根据这个 SessionId，就能取得客户端的数据信息。

如果客户端浏览器意外关闭，服务器保存的 Session 数据不是立即释放，此时数据还会存在，只要我们知道那个 SessionId，就可以继续通过请求获得此 Session 的信息，因为此时后台的 Session 还存在，当然我们可以设置一个 Session 超时时间，一旦超过规定时间没有客户端请求时，服务器就会清除对应 SessionId 的 Session 信息。

HTTP 协议的主要特点可概括如下：

1）支持客户/服务器模式。

2）简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

3）灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type（Content-Type 是 HTTP 包中用来表示内容类型的标识）加以标记。

4）无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5）无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

（3）问题扩展：

无

（4）结合项目中使用：

无

列举出 3 个你知道的设计模式，并简述你的理解

题目标签：Python 版本

试题编号：MS002280

（1）问题分析：

考官主要想考察学员对于前端技术的掌握

（2）核心答案讲解：

1）Singleton，单例模式：保证一个类只有一个实例，并提供一个访问它的全局访问点

2）Abstract Factory，抽象工厂：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们的具体类。

3）Factory Method，工厂方法：定义一个用于创建对象的接口，让子类决定实例化哪一个类，Factory Method 使一个类的实例化延迟到了子类。

（3）问题扩展：

无

（4）结合项目中使用：

无

第三部分：实际业务问题

介绍 django web 框架，如何创建项目、app 和同步数据库（1.8 版本以上）

题目标签：其他

试题编号：MS001180

（1）问题分析

（2）核心问题讲解

问题描述：

在 django 中创建了一个 app，而且在 app 中自定义创建了几个数据表，在同步的时候系统自带的表可以成功，但是 models 中的没有生效，而且进入对应 app 下的 migrations 目录，发现为空，应该如何解决呢！

解决方式：

python3 manage.py makemigrations--empty managerbook # managerbook 就是你的 app 名字，此处要写成自己的 app 名字

python3 manage.py makemigrations # 再次正常运行生成迁移文件的命令

python3 manage.py migrate # 同步数据库

（3）问题扩展

（4）结合项目中使用

如果让你开发一个电商系统的秒杀活动，后台架构你会怎么设计，以确保系统能支撑活动开始后可能导致的交易高峰？

题目标签：架构设计

试题编号：MS001820

（1）问题分析

（2）核心问题讲解

（3）问题扩展

（4）结合项目中使用

请你介绍一下你所参与的你认为最成功的一个项目，包括该项目采用的技术?

题目标签：项目描述

试题编号：MS001826

（1）问题分析

（2）核心问题讲解

在此项目中我主要负责后台管理模块，主要实现商品管理和商品规格参数管理，对商品和商品规格进行 CRUD 操作。；在实现前台调用后台数据时，为了实现系统间的调用，便使用了 httpclient 技术来实现此功能，在后台提供了需要调用的接口。（httpclient 介绍，工作原理，优缺点）。如果在后台对商品进行操作，为了使前台数据与后台数据实现同步，我们使用了 RabbitMQ 消息队列机制实现商品同步功能（RabbitMQ 介绍，工作原理，优缺点）；

在此项目中，我还参与了购物车模块的开发。在开发这个模块时候，我们考虑了会员在未登录和登录两种情况下把商品加入购物车，后台如何该保存商品信息。

（3）问题扩展

（4）结合项目中使用

请说明 Django 中的 web 认证登陆机制?

题目标签：Django

试题编号：MS001832

（1）问题分析

（2）核心问题讲解

首先，前端通过 Web 表单将自己的用户名和密码发送到后端的接口。这一过程一般是一个 HTTP POST 请求。建议的方式是通过 SSL 加密的传输（https 协议），从而避免敏感信息被嗅探。

后端核对用户名和密码成功后，将用户的 id 等其他信息作为 JWT Payload（负载），将其与头部分别进行 Base64 编码拼接后签名，形成一个 JWT。形成的 JWT 就是一个形同 lll.zzz.xxx 的字符串。

后端将 JWT 字符串作为登录成功的返回结果返回给前端。前端可以将返回的结果保存在 localStorage 或 sessionStorage 上，退出登录时前端删除保存的 JWT 即可。

前端在每次请求时将 JWT 放入 HTTP Header 中的 Authorization 位。(解决 XSS 和 XSRF 问题)

后端检查是否存在，如存在验证 JWT 的有效性。例如，检查签名是否正确；检查 Token 是否过期；检查 Token 的接收方是否是自己（可选）。

验证通过后后端使用 JWT 中包含的用户信息进行其他逻辑操作，返回相应结果。

（3）问题扩展

（4）结合项目中使用

请说明 Django 中的 web 认证登陆机制?

题目标签：Django

试题编号：MS002007

（1）问题分析

（2）核心问题讲解

首先，前端通过 Web 表单将自己的用户名和密码发送到后端的接口。这一过程一般是一个 HTTP POST 请求。建议的方式是通过 SSL 加密的传输（https 协议），从而避免敏感信息被嗅探。

后端核对用户名和密码成功后，将用户的 id 等其他信息作为 JWT Payload（负载），将其与头部分别进行 Base64 编码拼接后签名，形成一个 JWT。形成的 JWT 就是一个形同 lll.zzz.xxx 的字符串。

后端将 JWT 字符串作为登录成功的返回结果返回给前端。前端可以将返回的结果保存在 localStorage 或 sessionStorage 上，退出登录时前端删除保存的 JWT 即可。

前端在每次请求时将 JWT 放入 HTTP Header 中的 Authorization 位。(解决 XSS 和 XSRF 问题)

后端检查是否存在，如存在验证 JWT 的有效性。例如，检查签名是否正确；检查 Token 是否过期；检查 Token 的接收方是否是自己（可选）。

验证通过后后端使用 JWT 中包含的用户信息进行其他逻辑操作，返回相应结果。

（3）问题扩展

（4）结合项目中使用

搜索）、数据显示（BootstrapTable 翻页）、权限（控制表）等问题

产品迭代是怎么做的，可以讲一下详细的流程吗?

题目标签：产品迭代

试题编号：MS006608，删除：MS006883

（1）问题分析

考官主要想考察的是有没有产品迭代的经验

（2）核心问题讲解

1）需求初定

先由产品经理从需求池当中取出部分需求，作为本周期内需要开发的内容，并进行优先级排序，一般 P0、P1、P2 三级即可。优先级分类太多，很容易导致在不同需求的优先级排序上造成不必要的时间浪费。排序完成后，产品经理还可以先预估一下开发成本，如果感觉开发负担太重，那么就有必要砍掉一些优先级或投入产出比低的需求。

2）需求评估

召集设计同学、技术同学和测试同学，进行本周期的需求评估，以确定最终的开发内容，以及各部门工作的排期。这部分流程最好能通过一次稍微正式些的会议来进行。在会议这种正式场合上，大家表达意见一般都是经过认真思考的，给出排期时也会较为谨慎，而且有利于形成规范。会议结束后，可以发一封邮件给整个项目团队，说明会议内容与排期确定情况，越详细越好。这样将项目流程初步落实到纸面上，一定程度上可以防止迭代规划流于空谈。

3）需求落地（设计与开发）

这是一个至关重要的环节，直接决定着本周期内的需求迭代能否成功。在上一个流程即需求评估阶段，我们已经确定了最终的开发内容，但这并不代表迭代进入这个阶段后我们就没事可做了。作为产品经理，我们在产品生命周期的每一个阶段，都需要保持活跃。而这个阶段我们需要做的，就是跟进产品的设计、开发进度，以保证产品能够在拟定的期限内开发完成，达到可测试水平。

但“跟进”并不等于“监督”，我们是产品经理而不是包工头。不需要整天跟在设计和技术同学后头问“XX 需求做得怎么样了”，一方面无益于项目的实际进度，另一方面也会

让别人觉得你自身能力不够，却只会一味要求别人，从而影响到你们在项目当中的合作。

4）需求测试

在这个环节，我们要将本周期内开发完成的需求全部提交测试。需求测试分为两部分，第一部分是产品经理自测整体逻辑，也就是说不需要关注细节与极限问题，只要逻辑总体上没有问题，此部分测试便可通过。第二部分是提交 QA 测试，简称“提测”。与需求落地环节一样，这个阶段中的产品经理看似无事可做，实则不可或缺。我们需要跟进测试进度，在测试对提测内容和逻辑有疑问时，需要及时解答。

（3）问题扩展

无

（4）结合项目中使用

主要负责的就是产品的迭代开发。

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=3C7B191B7DC691EA9C33DC5901307461

常小型网站的一般并发数是多少？如何解决高并发？

题目标签：网站高并发

试题编号：MS006615，删除：MS006885

（1）问题分析

考官主要想考察的是对网站并发情况的掌握与经验

（2）核心问题讲解

一秒内可以处理的请求数量称之为服务器的 QPS　00～800QPS——带宽极限型

目前服务器大多用了 IDC 提供的“百兆带宽”，这意味着网站出口的实际带宽是 8M Byte 左右。假定每个页面只有 10K Byte，在这个并发条件下，百兆带宽已经吃完。首要考虑是 CDN 加速 / 异地缓存，多机负载等技术。

1）HTML 静态化

其实大家都知道，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。但是对于大量内容并且频繁更新的网站，我们无法全部手动去挨个实现，于是出现了我们常见的信息发布系统 CMS，像我们常访问的各个门户站点的新闻频道，甚至他们的其他频道，都是通过信息发布系统来管理和实现的，信息发布系统可以实现最简单的信息录入自动生成静态页面。

2）图片服务器分离

对于 Web 服务器来说，不管是 Apache、IIS 还是其他容器，图片是最消耗资源的，于是我们有必要将图片与页面进行分离，这是基本上大型网站都会采用的策略，他们都有独立的图片服务器，甚至很多台图片服务器。这样的架构可以降低提供页面访问请求的服务器系统压力，并且可以保证系统不会因为图片问题而崩溃，在应用服务器和图片服务器上，可以进行不同的配置优化，比如 apache 在配置 ContentType 的时候可以尽量少支持，尽可能少的 LoadModule，保证更高的系统消耗和执行效率。

3）数据库集群和库表散列

在数据库集群方面，很多数据库都有自己的解决方案，Oracle、Sybase 等都有很好的方案，常用的 MySQL 提供的 Master/Slave 也是类似的方案，以在 web 开发中使用

4）缓存

架构方面的缓存，对 Apache 比较熟悉的人都能知道 Apache 提供了自己的缓存模块，也可以使用外加的 Squid 模块进行缓存，这两种方式均可以有效的提高 Apache 的访问响应能力。

网站程序开发方面的缓存，Linux 上提供的 Memory Cache 是常用的缓存接口，可以在 web 开发中使用

5）镜像

镜像是大型网站常采用的提高性能和数据安全性的方式，镜像的技术可以解决不同网络接入商和地域带来的用户访问速度差异，比如 ChinaNet 和 EduNet 之间的差异就促使了很多网站在教育网内搭建镜像站点，数据进行定时更新或者实时更新。

6）负载均衡

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）问题扩展

如果某个时间段比如电商网站的双 11，用户访问数过大，就必须要进行限流，限制相同 ip 的重复访问，避免爬虫用户多次访问网站

（4）结合项目中使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=B250630CF8697F489C33DC5901307461

线程和协程的具体应用?

题目标签：进程/线程/协程

试题编号：MS006617

（1）问题分析

考官主要想考察的是对线程和协程的区别已及对多线程和协程的理解

（2）核心问题讲解

线程具体应用：

1）后台任务，例如：定时向大量（100w 以上）的用户发送邮件；

2）异步处理，例如：发微博、记录日志等；

3）分布式计算

协程具体应用：

当程序中存在大量不需要 CPU 的操作时（IO），适用于协程；

（3）问题扩展

线程和协程的区别：线程更多是抢占式线程，协程是协作式线程。我们暂时不考虑是有栈还是无栈的区别，那么协程跟线程的区别更多在于你觉得哪个调度策略更高效。操作系统还是开发者。协程上下文切换现在是在用户空间，因此调度的成本相对较低。不过也不代表这就是高效的。我们可以笼统地认为在面对 IO 密集型任务的时候，协程更高效，因为绝大部分时间都是在等待 IO。而面对 CPU 密集型的任务，线程更高效。

然而 Python 的常规实现因为有 GIL 的存在，你可以认为协程吊打线程。但是仅在 CPython 跟 PyPy 这些实现里面，千万不要认为是定律。

（4）结合项目中使用

tornado 中异步工作原理是什么?

题目标签：tornado 异步

试题编号：MS006619，删除：MS006894

（1）问题分析

考官主要想考察的是 tornado 框架中异步工作的原理。

（2）核心问题讲解

tornado 有四类异步事件：**立即事件，定时器异步事件，io 异步事件，Future 异步事件。**

Tornado 异步的核心是 ioloop.py 和 iostream.py 这两个文件。ioloop.py 实现了一个处理 I/O 事件的循环，iostream 封装了非阻塞的 socket 并把 I/O 事件注册到 ioloop 上。Tornado 的异步在 linux 平台基于 epoll，它是基于事件而非轮询的，这是其高效的原因（windows 平台没有 epoll，tornado 只能使用 select，效率比 epoll 低）。tornado 的 ioloop 管理所有的异步事件，并在适当的时机调用异步事件的回掉函数。

四类异步事件均在 ioloop 的 start 函数中调度。

立即事件：

场景：当前函数执行完后，下次 ioloop 调度时直接调度某函数

用法：ioloop.add_callback(callback, *args, **kwargs)

原理：立即事件全部存放在 ioloop._callbacks 中，IOLoop 每次循环都会调用这些立即事件的回调函数。

定时器异步事件：

场景：用户希望在某一段时间后执行某函数

用法：ioloop.call_at(when, callback, *args, **kwargs), ioloop.call_later(delay, callback, *args, **kwargs)

原理：定时器事件存放在 ioloop._timeouts 中，IOLoop 每次循环开始都会找出所有已经超时的定时器，并调用对应的回调函数。

IO 异步事件：

场景：等待某个文件描述符的某个事件，如 TCPserver 等待 socket 的 READ 事件

用法：ioloop.add_handler(fd, callback, events)

原理：所有的文件描述符全部存放在 ioloop._impl 中，windows 平台下_impl 是 tornado.platform.select.SelectIOLoop 对象

在 linux 平台下_impl 是 tornado.platform.epoll.EPollIOLoop 对象，作用都是同时监听多个文件描述符。

Future 异步事件：

场景：等待某个异步事件结束后执行回掉函数

用法：ioloop.add_future(future, callback)， future.add_done_callback(callback)

原理：异步事件结束后调用 Future.set_result()，当执行 set_result 时将 future 所有的回掉函数添加为 ioloop 的立即事件。

（3）问题扩展

Tornado 中推荐用协程来编写异步代码.协程使用 Python 中的关键字 yield 来替代链式回调来实现挂起和继续程序的执行(像在 gevent 中使用的轻量级线程合作的方法有时也

称作协程,但是在 Tornado 中所有协程使用异步函数来实现的明确的上下文切换).

（4）结合项目中使用

人工智能这一块，除量化交易外，有没有图像识别或者语义识别的项目，想找这方面工作，通过具体的多个项目来练习

题目标签：人工智能

试题编号：MS006626

（1）问题分析

想了解目前市场上有哪些人工智能方面的应用，该学哪些方面的知识

（2）核心问题讲解

目前课程很少涉及图像识别和语义识别，市场上工作不多，且学历一般要求高。项目例如人脸识别、车牌号识别、验证码识别等方面

（3）问题扩展

（4）结合项目中使用

淘宝主页的商品推送是怎么按照每人喜好不同来实现的?

题目标签：具体业务实现

试题编号：MS006628，删除：MS006906

（1）问题分析

考官主要想考察面试者对推荐系统的理解

（2）核心问题讲解

1）根据商品的分类或其他基础属性进行推荐（相似性推荐），对于某一个商品来说，这是一种替代性的推荐方式，也即用户不想买它的时候，还可以有其他的选择。就比如说用户正在浏览一个斯伯丁的篮球，看完描述之后发现不是自己理想的款式，规格材质不太对，但是在这个单品下方，出现了一个同类型的耐磨材质的篮球，OK！那么这个用户可能就会把这个推荐的篮球带回家。这个例子中仅仅是依据商品的分类进行推荐，当然我们还可以根据实际情况加入商品的更多基础属性进行加权取值，算最后得分来进行推荐。

2）根据商品的被动销售级属性进行推荐（相关性推荐），根据商品最终在订单中出现的概率来判断商品间的相关性，目前还可以依赖于其他几个维度来参考一同做判断（被同时浏览的几率，被同时加入购物车的几率，被同时购买的几率），以下采取订单中是否通知购买来讲解算法模型，简单解释一下就是想买 A 的人还可能想买 C。

3）通过记录用户行为数据，如搜索、浏览、咨询、加购、支付、收藏、评价、分享等等进行推荐；

4）基于用户的协同过滤，通过用户对不同类型的商品的喜好度进行评分，然后根据每类商品的喜好度评分构建一个多维向量，使用余弦公式有来评测用户之间喜好度的相似性，基于此将其他相似用户非常喜欢而该用户还没有了解的产品进行推荐。这部分推荐本质上是给用户推荐其他相似用户喜欢的内容，一句话概括：和你类似的人也喜欢这些商品。

（3）问题扩展

说到底，淘宝的推荐系统是找到用户和商品的连接。基于这样一个事实，万事万物有相互连接的大趋势，比如人和人倾向于有更多的社会连接，于是有了各种社交产品；比如人和

商品有越来越多的消费连接，于是有了各种各样的电商平台；人和资讯有越来越多的阅读连接，于是有了信息流产品。总结一下推荐系统就是：用已有的连接去预测未来用户和商品间会出现的连接。

（4）结合项目中使用

推荐系统在项目中主语会遇到冷启动问题，探索和利用问题；安全问题。

百度地图如何调用，如何实现一级一级放大并显示数据库数据?

题目标签：具体业务实现

试题编号：MS006632，删除：MS006887

（1）问题分析

考官想考查有没有百度地图这些第三方平台的调用经验

（2）核心问题讲解

1）明确自己的需求，调用地图实现什么功能，定位、地图、轨迹、导航、路况等等。百度地图的调用，上百度地图开放平台，申请账号和 ak，根据业务需求（iOS、安卓、web 等等），选择开发文档；以 web 为例，选择 JavaScript API，创建 HTML 文档，引用百度地图 API 文件，创建实例，初始化。。。。一步一步根据自己的需求，参考开发文档来实现。

2）百度地图的缩放，创建实例后，启用滚轮放大缩小(默认是关闭的)。就可以通过鼠标滚轮缩放地图。通过 setZoom（）函数控制地图的级别，通过级别不同来调用数据库数据。

（3）问题扩展

如何调起百度地图，跳转到百度地图

如何调用高德地图等等第三方应用

（4）结合项目中使用

hadoop 和 spark 的开发模式

题目标签：hadoop 和 sparl 的开发模式

试题编号：MS006637，删除：MS006907

（1）问题分析

Hadoop：只提供了 Map 和 Reduce 两种操作所有的作业都得转换成 Map 和 Reduce 的操作。

Spark：提供很多种的数据集操作类型比如 Transformations 包括 map, filter, flatMap, sample, groupByKey, reduceByKey, union, join, cogroup, mapValues,sort,partionBy 等多种操作类型，还提供 actions 操作包括 Count,collect, reduce, lookup, save 等多种。这些多种多样的数据集操作类型，给开发上层应用的用户提供了方便。

（2）核心问题讲解

Hadoop:MapRedcue 由 Map 和 Reduce 两个阶段，并通过 shuffle 将两个阶段连接起来的。但是套用 MapReduce 模型解决问题，不得不将问题分解为若干个有依赖关系的子问题，每个子问题对应一个 MapReduce 作业，最终所有这些作业形成一个 DAG。

Spark:是通用的 DAG 框架，可以将多个有依赖关系的作业转换为一个大的 DAG。核心思想是将 Map 和 Reduce 两个操作进一步拆分为多个元操作，这些元操作可以灵活组合，

产生新的操作，并经过一些控制程序组装后形成一个大的 DAG 作业。

（3）问题扩展

1）速度快

Apache Spark 拥有先进的 DAG 调度器、查询优化器以及物理执行引擎从而高性能的实现批处理和流数据处理。

2）易用性（可以使用 Java，Scala，Python，R 以及 SQL 快速的写 Spark 应用）

Spark 提供 80 个以上高级算子便于执行并行应用，并且可以使用 Scala、Python、R 以及 SQL 的 shell 端交互式运行 Spark 应用。

```
#通过 Spark 的 Python 的 DataFrame 的 API 读取 JSON 文件
df = spark.read.json("logs.json")
df.where("age > 21").show()
```

3）通用性(支持 SQL，流数据处理以及复杂分析)

Spark 拥有一系列库，包括 SQL 和 DataFrame，用于机器学习的 MLib,支持图计算 GraphX 以及流计算模块 Streaming。你可以在一个应用中同时组合这些库。

4）支持多种模式运行（平台包括 Hadoop,Apache Mesos,Kubernete,standalone 或者云上，也可以获取各种数据源上的数据）

Spark 可以直接运行以自身的 standalone 集群模式运行，也可以在亚马逊 EC2 上运行，不过企业级用的比较多的是 Hadoop Yarn 模式，当然也有 Mesos 和 Kubernetes 模式。可以获取不限于来自于 HDFS、Apache Cassandra、Apache HBase 和 Apache Hive 等上百种数据源。

（4）结合项目中使用

项目中实现项目一：流分析

很多人会把这个“流”，但流分析是不同的，从设备流。通常，流分析是一个组织在批处理中的实时版本。以反洗钱和欺诈检测：为什么不在交易的基础上，抓住它发生而不是在一个周期结束？同样的库存管理或其他任何。在某些情况下，这是一种新的类型的交易系统，分析数据位的位，因为你将它并联到一个分析系统中。这些系统证明自己如 Spark 或 Storm 与 Hbase 作为常用的数据存储。请注意，流分析并不能取代所有形式的分析，对某些你从未考虑过的事情而言，你仍然希望分析历史趋势或看过去的数据。

项目二：数据整合

称之为“企业级数据中心”或“数据湖”，这个想法是你有不同的数据源，你想对它们进行数据分析。这类项目包括从所有来源获得数据源（实时或批处理）并且把它们存储在 hadoop 中。有时，这是成为一个“数据驱动的公司”的第一步；有时，或许你仅仅需要一份漂亮的报告。“企业级数据中心”通常由 HDFS 文件系统和 HIVE 或 IMPALA 中的表组成。未来，HBase 和 Phoenix 在大数据整合方面将大展拳脚，打开一个新的局面，创建出全新的数据美丽新世界。销售人员喜欢说“读模式”，但事实上，要取得成功，你必须清楚的了解自己的用例将是什么（Hive 模式不会看起来与你在企业数据仓库中所做的不一样）。真实的原因是一个数据湖比 Teradata 和 Netezza 公司有更强的水平扩展性和低得多的成本。许多人在做前端分析时使用 Tabelu 和 Excel。许多复杂的公司以“数据科学家”用 Zeppelin 或 IPython 笔记本作为前端。

zookeeper 和 kafka 分布式框架的使用以及作用

题目标签：zookeeper 和 kafka

试题编号：MS006638，删除：MS006908

（1）问题分析

该题主要想考察面试者对 zookeeper 和 kafka 分布式框架的概念理解与应用方向，那么在回答时讲解清楚概念以及这两个框架的使用和作用即可

（2）核心问题讲解

Zookeeper 框架概念：

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

Zookeeper 应用方向：

1）zookeeper 是一个精简的文件系统。这点它和 hadoop 有点像，但是 zookeeper 这个文件系统是管理小文件的，而 hadoop 是管理超大文件的。

2）zookeeper 提供了丰富的“构件”，这些构件可以实现很多协调数据结构和协议的操作。例如：分布式队列、分布式锁以及一组同级节点的“领导者选举”算法。

3）zookeeper 是高可用的，它本身的稳定性是相当之好，分布式集群完全可以依赖 zookeeper 集群的管理，利用 zookeeper 避免分布式系统的单点故障的问题。

4）zookeeper 采用了松耦合的交互模式。这点在 zookeeper 提供分布式锁上表现最为明显，zookeeper 可以被用作一个约会机制，让参入的进程不在了解其他进程的（或网络）的情况下能够彼此发现并进行交互，参入的各方甚至不必同时存在，只要在 zookeeper 留下一条消息，在该进程结束后，另外一个进程还可以读取这条信息，从而解耦了各个节点之间的关系。

5）zookeeper 为集群提供了一个共享存储库，集群可以从这里集中读写共享的信息，避免了每个节点的共享操作编程，减轻了分布式系统的开发难度。

6）zookeeper 的设计采用的是观察者的设计模式，zookeeper 主要是负责存储和管理大家关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式。

Zookeeper 的作用：

1）Zookeeper 用来注册服务和进行负载均衡，由于哪个服务由哪个机器来提供需要让调用者知道，即 ip 地址和服务名称的对应关系必须一致所以通过硬编码的方式把这种对应关系在调用方业务代码中实现。但是如果提供服务的机器挂掉而调用者无法知晓，若不更改代码则会继续请求挂掉的机器提供服务。

2）Zookeeper 通过心跳机制可以检测挂掉的机器并将挂掉机器的 ip 和服务对应关系从列表中删除。至于支持高并发，简单来说就是横向扩展，在不更改代码的情况通过添加机器来提高运算能力。通过添加新的机器向 Zookeeper 注册服务，服务的提供者多了能服务的客户就多了

Kafka 框架概念：

Kafka 是由 Apache 软件基金会开发的一个开源流处理平台，由 Scala 和 Java 编写。Kafka 是一种高吞吐量的分布式发布订阅消息系统，它可以处理消费者规模的网站中的所有动作流数据。 这种动作（网页浏览，搜索和其他用户的行动）是在现代网络上的许多社会功能的一个关键因素。 这些数据通常是由于吞吐量的要求而通过处理日志和日志聚合来解决。 对于像 Hadoop 一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一

个可行的解决方案。Kafka 的目的是通过 Hadoop 的并行加载机制来统一线上和离线的消息处理，也是为了通过集群来提供实时的消息。

Kafka 的应用方向：

1）Messaging

对于一些常规的消息系统,kafka 是个不错的选择;partitons/replication 和容错,可以使 kafka 具有良好的扩展性和性能优势.不过到目前为止,我们应该很清楚认识到,kafka 并没有提供 JMS 中的"事务性""消息传输担保(消息确认机制)""消息分组"等企业级特性;kafka 只能使用作为"常规"的消息系统,在一定程度上,尚未确保消息的发送与接收绝对可靠(比如,消息重发,消息发送丢失等

2）Websit activity tracking

kafka 可以作为"网站活性跟踪"的最佳工具;可以将网页/用户操作等信息发送到 kafka 中.并实时监控,或者离线统计分析等

3）Log Aggregation

kafka 的特性决定它非常适合作为"日志收集中心";application 可以将操作日志"批量""异步"的发送到 kafka 集群中,而不是保存在本地或者 DB 中;kafka 可以批量提交消息/压缩消息等,这对 producer 端而言,几乎感觉不到性能的开支.此时 consumer 端可以使 hadoop 等其他系统化的存储和分析系统.

Kafka 的作用：

1）生产者的负载与消费者的负载解耦

2）消费者按照自己的能力 fetch 数据

3）消费者可以自定义消费的数量

（3）问题扩展

Kafka 就是一个分布式的消息系统。由 producer、broker、consumer、zookeeper 这几大构件组成。Zookeeper 的引入极大的支持了其拓展。

zookeeper 管理 broker、consumer

创建 Broker 后，向 zookeeper 注册新的 broker 信息，实现在服务器正常运行下的水平拓展。具体的，通过注册 watcher，获取 partition 的信息。

Topic 的注册，zookeeper 会维护 topic 与 broker 的关系，通过 /brokers/topics/topic.name 节点来记录。

Producer 向 zookeeper 中注册 watcher,了解 topic 的 partition 的消息，以动态了解运行情况，实现负载均衡。Zookeepr 是没有管理 producer，只是能够提供当前 broker 的相关信息。

Consumer 可以使用 group 形式消费 kafka 中的数据。所有的 group 将以轮询的方式消费 broker 中的数据，具体的按照启动的顺序。Zookeeper 会给每个 consumer group 一个 ID,即同一份数据可以被不同的用户 ID 多次消费。因此这就是单播与多播的实现。以单个消费者还是以组别的方式去消费数据，由用户自己去定义。Zookeeper 管理 consumer 的 offset 跟踪当前消费的 offset

（4）结合项目中使用

Openstack 和 K8S 技术的使用

题目标签：Openstack 和 K8S

试题编号：MS006639，删除：MS006899

（1）问题分析

考官是想考察对 Openstack 和 K8S 结合使用的概念思路

（2）核心问题讲解

OpenStack 与 K8S 结合主要有两种方案。**一是 K8S 部署在 OpenStack 平台之上，二是 K8S 和 OpenStack 组件集成。**

首先第一种方案目前也是大多数用户选择的方案，这种方式的优点是 K8S 能够快速部署、弹性扩容，并且通过虚拟机的多租户间接实现了容器的多租户，隔离性好。缺点是容器跑在虚拟机上，多多少少计算性能可能会有点损耗，网络的多层 overlay 嵌套也可能导致性能下降。

OpenStack Magnum 项目是该方案实现的代表，该项目为 OpenStack 提供容器编排服务，通过该组件，用户可以快速部署一个 K8S、Mesos 以及 Swarm 集群，原理和 OpenStack 大多数的高级服务实现差不多，先通过 heat 完成资源编排（创建虚拟机、volume、安全组等），然后通过镜像里面的 heat-container-agent 以及一些脚本完成 K8S、Mesos 以及 Swarm 集群的安装配置。当然，通过 Ironic，Magnum 支持将容器编排组件直接部署在物理机（裸机）上。

第二种方案是 K8S 与 OpenStack 的各个组件集成，在 OpenStack 社区以及 K8S 社区的共同努力下，目前可以集成的组件还是挺多的，下面简单介绍下。

1）K8S 与 OpenStack Keystone 集成

K8S 可以和 OpenStack Keystone 集成，即 K8S 可以使用 Keystone 认证，参考 keystone authentication kubernetes-cluster。

2）K8S 与 OpenStack Glance 集成

这个没有必要，因为 Docker 的镜像是分层的，使用 Registry 或者 Harbor 即可。当然如果有必要可以使用 Glance 存储 Docker 镜像作为备份，不过更建议备份到 OpenStack Swift，Registry 以及 Harbor 都原生支持使用 Swift 作为存储后端。

3）K8S 与 OpenStack Neutron 集成

前面提到的通过 Magnum 把容器部署在虚拟机，其实并没有根本改变 K8S 的网络模型，K8S 的底层网络依然还是诸如 Flannel、Contrail 等网络模型，和 Neutron 其实没有多大关系。另外，前面也说了，容器运行在虚拟机中不仅可能会导致计算性能损耗，网络的多层 Overlay 嵌套也可能会大大降低容器的网络性能。

其实社区已经实现 K8S 直接 OpenStack Neutron 网络集成，即 kuryr-kubernetes 项目。K8S 的 pod 与 OpenStack 虚拟机是平等公民，共享 Neutron 网络服务，K8S 网络具备和 OpenStack 虚拟机等同的功能，比如安全组、防火墙、QoS 等。

不过遗憾的是，目前 kuryr 还不支持多租户，Kuryr 使用 Neutron 的 network 以及 subnet 都是配置写死的，而不是创建 port 时指定。

4）K8S 与 Cinder 集成

目前 K8S 已经实现了很多 volume 插件，PV 支持对接各种存储系统，比如 Ceph RBD、GlusterFS、NFS 等等，参考 kubernetes persistent volumes，其中就包含了 Cinder，即 K8S 可以使用 Cinder 提供 volume 服务，这样 K8S 和 Nova 共享一套存储系统，都是 Cinder 的消费者。Cinder 屏蔽了底层存储系统，K8S 直接对接 Cinder，省去了一堆 plugins 的安装配置。

5）K8S 与 Manila 集成

前面提到 K8S 与 Cinder 集成，其实 K8S 还支持与 OpenStack Manila 服务集成，目

前该插件已经包含在 K8S 的 external storage 项目中。

（3）问题扩展

OpenStack：

OpenStack 覆盖了网络、虚拟化、操作系统、服务器等各个方面。它是一个正在开发中的云计算平台项目，根据成熟及重要程度的不同，被分解成核心项目、孵化项目，以及支持项目和相关项目。每个项目都有自己的委员会和项目技术主管，而且每个项目都不是一成不变的，孵化项目可以根据发展的成熟度和重要性，转变为核心项目。截止到 Icehouse 版本，下面列出了 10 个核心项目（即 OpenStack 服务）。

计算（Compute）：Nova。一套控制器，用于为单个用户或使用群组管理虚拟机实例的整个生命周期，根据用户需求来提供虚拟服务。负责虚拟机创建、开机、关机、挂起、暂停、调整、迁移、重启、销毁等操作，配置 CPU、内存等信息规格。自 Austin 版本集成到项目中。

对象存储（Object Storage）：Swift。一套用于在大规模可扩展系统中通过内置冗余及高容错机制实现对象存储的系统，允许进行存储或者检索文件。可为 Glance 提供镜像存储，为 Cinder 提供卷备份服务。自 Austin 版本集成到项目中

镜像服务（Image Service）：Glance。一套虚拟机镜像查找及检索系统，支持多种虚拟机镜像格式（AKI、AMI、ARI、ISO、QCOW2、Raw、VDI、VHD、VMDK），有创建上传镜像、删除镜像、编辑镜像基本信息的功能。自 Bexar 版本集成到项目中。

身份服务（Identity Service）：Keystone。为 OpenStack 其他服务提供身份验证、服务规则和服务令牌的功能，管理 Domains、Projects、Users、Groups、Roles。自 Essex 版本集成到项目中。

网络&地址管理（Network）：Neutron。提供云计算的网络虚拟化技术，为 OpenStack 其他服务提供网络连接服务。为用户提供接口，可以定义 Network、Subnet、Router，配置 DHCP、DNS、负载均衡、L3 服务，网络支持 GRE、VLAN。插件架构支持许多主流的网络厂家和技术，如 OpenvSwitch。自 Folsom 版本集成到项目中。

块存储（Block Storage）：Cinder。为运行实例提供稳定的数据块存储服务，它的插件驱动架构有利于块设备的创建和管理，如创建卷、删除卷，在实例上挂载和卸载卷。自 Folsom 版本集成到项目中。

UI 界面（Dashboard）：Horizon。OpenStack 中各种服务的 Web 管理门户，用于简化用户对服务的操作，例如：启动实例、分配 IP 地址、配置访问控制等。自 Essex 版本集成到项目中。

测量（Metering）：Ceilometer。像一个漏斗一样，能把 OpenStack 内部发生的几乎所有的事件都收集起来，然后为计费和监控以及其它服务提供数据支撑。自 Havana 版本集成到项目中。

部署编排（Orchestration）：Heat [2] 。提供了一种通过模板定义的协同部署方式，实现云基础设施软件运行环境（计算、存储和网络资源）的自动化部署。自 Havana 版本集成到项目中。

数据库服务（Database Service）：Trove。为用户在 OpenStack 的环境提供可扩展和可靠的关系和非关系数据库引擎服务。自 Icehouse 版本集成到项目中。

Kubernetes：

是一个全新的基于容器技术的分布式架构领先方案。Kubernetes(k8s)是 Google 开源的容器集群管理系统（谷歌内部:Borg）。在 Docker 技术的基础上，为容器化的应用提供

部署运行、资源调度、服务发现和动态伸缩等一系列完整功能，提高了大规模容器集群管理的便捷性。

Kubernetes 是一个完备的分布式系统支撑平台，具有完备的集群管理能力，多扩多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和发现机制、内建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制以及多粒度的资源配额管理能力。同时 Kubernetes 提供完善的管理工具，涵盖了包括开发、部署测试、运维监控在内的各个环节。

Kubernetes 中，Service 是分布式集群架构的核心，一个 Service 对象拥有如下关键特征：

拥有一个唯一指定的名字

拥有一个虚拟 IP（Cluster IP、Service IP、或 VIP）和端口号

能够体统某种远程服务能力

被映射到了提供这种服务能力的一组容器应用上

Service 的服务进程目前都是基于 Socket 通信方式对外提供服务，比如 Redis、Memcache、MySQL、Web Server，或者是实现了某个具体业务的一个特定的 TCP Server 进程，虽然一个 Service 通常由多个相关的服务进程来提供服务，每个服务进程都有一个独立的 Endpoint（IP+Port）访问点，但 Kubernetes 能够让我们通过服务连接到指定的 Service 上。有了 Kubernetes 内奸的透明负载均衡和故障恢复机制，不管后端有多少服务进程，也不管某个服务进程是否会由于发生故障而重新部署到其他机器，都不会影响我们队服务的正常调用，更重要的是这个 Service 本身一旦创建就不会发生变化，意味着在 Kubernetes 集群中，我们不用为了服务的 IP 地址的变化问题而头疼了。

容器提供了强大的隔离功能，所有有必要把为 Service 提供服务的这组进程放入容器中

进行隔离。为此，Kubernetes 设计了 Pod 对象，将每个服务进程包装到相对应的 Pod 中，使其成为 Pod 中运行的一个容器。为了建立 Service 与 Pod 间的关联管理，Kubernetes 给每个 Pod 贴上一个标签 Label，比如运行 MySQL 的 Pod 贴上 name=MySQL 标签，给运行 PHP 的 Pod 贴上 name=php 标签，然后给相应的 Service 定义标签选择器 Label Selector，这样就能巧妙的解决了 Service 于 Pod 的关联问题。

在集群管理方面，Kubernetes 将集群中的机器划分为一个 Master 节点和一群工作节点 Node，其中，在 Master 节点运行着集群管理相关的一组进程 kube-apiserver、kube-controller-manager 和 kube-scheduler，这些进程实现了整个集群的资源管理、Pod 调度、弹性伸缩、安全控制、系统监控和纠错等管理能力，并且都是全自动完成的。Node 作为集群中的工作节点，运行真正的应用程序，在 Node 上 Kubernetes 管理的最小运行单元是 Pod。Node 上运行着 Kubernetes 的 kubelet、kube-proxy 服务进程，这些服务进程负责 Pod 的创建、启动、监控、重启、销毁以及实现软件模式的负载均衡器。

（4）结合项目中使用

1）部署、恢复速度快：准备好非 openstack 组件外（MySQL、rabbitmq、haproxy、keepalived 等），使用 k8s 的 pod、deployment、daemonset 实现自动化部署和高速回溯。

2) 服务及主机监控: 主机信息 (cpu、内存、文件系统、网络) 使用 kubelet 的 cAdvisor 监控，通过 web 展示；openstack 服务使用 heapster 监控，通过 k8s dashboard 展示，方便运维。

3）服务高可用、弹性伸缩、快速扩容：openstack 进程服务使用 k8s 的 failover 机制（包括容器异常重启及故障迁移），弹性伸缩改进了例如 nova-scheduler、cinder-volume

这些依赖 rpc 服务的高可用方式，daemonset 负责 API 节点以及 nova-compute、ironic-conductor 节点的快速扩容。

4）可视化容器仓库及滚动升级：使用 harbor 作为容器提供，提供 portal 界面；k8s 的滚动升级能够在尽可能保证服务的情况下，逐步升级版本。

5）Config 配置文件节点：所有配置文件放在同一目录下，使用 git 控制版本，k8s 重建服务即可更新配置。

6）日志分析：所有 openstack 日志都放在控制节点的/var/log/openstack 下，使用 logstash 的 daemonset，配合 es、kibana 分析，快速定位问题。

7）CICD：修改配置文件{通过修改 config 节点配置文件->git commit->k8s 重建服务}；版本升级{新版本 rpm 包更新至源->运维节点生成最新容器镜像、打 tag、上传至 harbor 仓库->k8s 滚动升级}；错误回退{k8s 能够滚动回退到历史版本}。

Django 作为后端怎么实现给客户端推送消息，app 里的推送如何实现?

题目标签：后端消息推送

试题编号：MS006641，删除：MS006881

（1）问题分析

考官主要想考察的是在网页应用中，需要在处理完表单或其它类型的用户输入后，显示一个通知消息（也叫做“flash message”）给用户。

（2）核心问题讲解

Django 后台可以使用 django-push-notifications 推送模块来进行消息推送

也可以在 Django 中使用 Message 框架向模板中推送消息内容

（https://yiyibooks.cn/xx/django_182/ref/contrib/messages.html）

app 中的推送，有两种方式，第一种是自己研发的，但由于研发成本高，所以大多数都采用第二中方式，也就是使用第三方工具进行推送，如：极光推送、个推、百度云推送、华为推送等

（3）问题扩展

App 有本地推送和远程推送：

本地推送通知：本地通知不需要连接网络，一般是开发人员在合适的情况下在 App 内发送通知，应用场景：当能够确定在某个时间时需要提醒用户。

远程通知：远程通知必须需要连接网络，远程推送服务又称为 APNs(Apple Push Notification Services),一般是服务器端发送通知。

对于用户，通知一般是指的推送通知，即本地推送通知和远程推送通知。

（4）结合项目中使用

在视图和模板中使用消息：

add_message(request, level, message, extra_tags='', fail_silently=False)

新增一条消息，调用：

from django.contrib import messages

messages.add_message(request, messages.INFO, 'Hello world.')

有几个快捷方法提供标准的方式来新增消息并带有常见的标签（这些标签通常表示消息的 HTML 类型）：

messages.debug(request, '%s SQL statements were executed.' % count)

messages.info(request, 'Three credits remain in your account.')

messages.success(request, 'Profile details updated.')

messages.warning(request, 'Your account expires in three days.')

messages.error(request, 'Document deleted.')

数据结构与算法：实现一个二分查找，快速排序。

题目标签：数据结构和算法

试题编号：MS006655，删除：MS006868

（1）问题分析

考官考察对数据结构与算法的理解与实战。

（2）核心问题讲解

二分查找又叫折半查找，二分查找属于分治法的应用。所谓分治法，就是将原问题分解成若干个子问题后，利用了规模为 n 的原问题的解与较小规模（通常是 n/2）的子问题的解之间的关系。

使用二分查找的前提是原数列是有序的。其基本思想为：在有序表中，取中间记录作为比较对象，若给定值与中间记录的关键码相等，则查找成功；若给定值小于中间记录的关键码，则在中间记录的左半边继续查找；若给定值大于中间记录的关键码，则在中间记录右半边区继续查找。不断重复上述过程，直到查找成功，或所查找的区域无记录，查找失败。

快速查找算法实现：

快速排序（英语：Quicksort），又称划分交换排序（partition-exchange sort），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。

步骤为：从数列中挑出一个元素，称为"基准"（pivot），重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区结束之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会结束，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去。

（3）问题扩展

二分查找的时间复杂度是 O(log(n))，最坏情况下的时间复杂度是 O(n)。

快速排序复杂度分析：

时间复杂度：

快速排序算法的时间复杂度为 O(nlogn)。

空间复杂度：主要是递归造成的栈空间的使用。

最好情况，递归树的深度为 log2n，其空间复杂度也就为 O(logn)；

最坏情况，需要进行 n-1 递归调用，其空间复杂度为 O(n)；

平均情况，空间复杂度也为 O(logn)。

（4）结合项目中使用

二分查找：

```
def binary_search(data,num,low,high):
    if low==high:
```

```
        if data[low]==num:
            return low
        else:
            return None
    mid = (low+high)//2
    if num ==data[mid]:
        return mid
    elif num < data[mid]:
        return binary_search(data,num,low,mid-1)
    else:
        return binary_search(data,num,mid+1,high)
```

快速排序：

```
def QuickSort(alist, start, end):
    '''快速排序'''
    # 建立递归终止条件
    if start >= end:
        return
    # low 为序列左边要移动的游标
    low = start
    # last 为序列右边要移动的游标
    last = end
```

```
    # 将起始元素设为要寻找位置的基准元素
    mid_num = alist[start]
    while low < last:
        # 当 low 与 last 未重合，并且比基准元素要大，就将游标向左移动
        while low < last and alist[last] >= mid_num:
            last -= 1
        # 如果比基准元素小，就跳出循环，并且把其放在基准元素左边
        alist[low] = alist[last]
        # 当 low 与 last 未重合，并且比基准元素要小，就将游标向右移动
        while low < last and alist[low] < mid_num:
            low += 1
        # 如果比基准元素大，就跳出循环，并且把其放在基准元素右边
        alist[last] = alist[low]
    # 当 low 与 last 相等，就是 mid_num 的排序位置
    alist[low] = mid_num
    # 然后对排序好的元素左右两边的序列进行递归
    QuickSort(alist, start, low-1) # 对左边的序列进行递归
    QuickSort(alist, low+1, end) # 对右边的序列进行递归

if __name__ == '__main__':
    alist = [30, 40, 60, 10, 20, 50]
```

```
#alist = [54,26,93,17,77,31,44,55,20]
print(alist)
QuickSort(alist, 0, len(alist)-1)
print(alist)
```

单例模式在工作中的应用场景。

题目标签：设计模式

试题编号：MS006658

（1）问题分析

单例模式在工作中的实际应用。

（2）核心问题讲解

首先介绍一下单例模式：

单例模式（Singleton），也叫单子模式，是一种常用的软件设计模式。在应用这个模式时，单例对象的类必须保证只有一个实例存在。许多时候整个系统只需要拥有一个的全局对象，这样有利于我们协调系统整体的行为。比如在某个服务器程序中，该服务器的配置信息存放在一个文件中，这些配置数据由一个单例对象统一读取，然后服务进程中的其他对象再通过这个单例对象获取这些配置信息。这种方式简化了在复杂环境下的配置管理。育网内搭建镜像站点，数据进行定时更新或者实时更新。在镜像的细节技术方面，这里不阐述太深，有很多专业的现成的解决架构和产品可选。也有廉价的通过软件实现的思路，比如 Linux 上的 rsync 等工具。

应用场景举例：

1）外部资源：每台计算机有若干个打印机，但只能有一个 PrinterSpooler，以避免两个打印作业同时输出到打印机。内部资源：大多数软件都有一个（或多个）属性文件存放系统配置，这样的系统应该有一个对象管理这些属性文件。

2）Windows 的 Task Manager（任务管理器）就是很典型的单例模式（这个很熟悉吧），想想看，是不是呢，你能打开两个 windows task manager 吗?

3）windows 的 Recycle Bin（回收站）也是典型的单例应用。在整个系统运行过程中，回收站一直维护着仅有的一个实例。

4）网站的计数器，一般也是采用单例模式实现，否则难以同步。

5）应用程序的日志应用，一般都何用单例模式实现，这一般是由于共享的日志文件一直处于打开状态，因为只能有一个实例去操作，否则内容不好追加。

6）Web 应用的配置对象的读取，一般也应用单例模式，这个是由于配置文件是共享的资源。

7）数据库连接池的设计一般也是采用单例模式，因为数据库连接是一种数据库资源。数据库软件系统中使用数据库连接池，主要是节省打开或者关闭数据库连接所引起的效率损耗，这种效率上的损耗还是非常昂贵的，因为何用单例模式来维护，就可以大大降低这种损耗。

8）多线程的线程池的设计一般也是采用单例模式，这是由于线程池要方便对池中的线程进行控制。

9）操作系统的文件系统，也是大的单例模式实现的具体例子，一个操作系统只能有一个文件系统。

10）HttpApplication 也是单位例的典型应用。熟悉 ASP.Net(IIS)的整个请求生命周

期的人应该知道 HttpApplication 也是单例模式，所有的 HttpModule 都共享一个 HttpApplication 实例。

（3）问题扩展

优点：

1）在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就防止其它对象对自己的实例化，确保所有的对象都访问一个实例

2）单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。

3）提供了对唯一实例的受控访问。

4）由于在系统内存中只存在一个对象，因此可以 节约系统资源，当 需要频繁创建和销毁的对象时单例模式无疑可以提高系统的性能。

5）允许可变数目的实例。

6）避免对共享资源的多重占用。

缺点：

1）不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。

2）由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

3）单例类的职责过重，在一定程度上违背了“单一职责原则”。

4）滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

（4）结合项目中使用

```
class ConnectionPool:
    __instance = None
    def __init__(self):
        self.addr = '127.0.0.1'
        self.port = 8008
        self.name = 'sql'
        self.conn_list = [1,2,3,4,5,6,7,8,9,10]
    @staticmethod
    def get_instance():
        if ConnectionPool.__instance:
            return ConnectionPool.__instance
        else:
            ConnectionPool.__instance == ConnectionPool()
            return ConnectionPool.__instance
    def get_connection(self):
        r = random.randrange(1, 11)
        return r
```

以上示例，通过一个私有变量，一个静态方法 get_instance()实现了一次创建对象，多次连接。

在不使用框架的前提下，如何使用 Python 发送一封带图片的邮件?

题目标签：第三方包扩展

试题编号：MS006659

（1）问题分析

考察对 Python 发送邮件的进阶理解。使用 Python 模块的灵活度。

（2）核心问题讲解

Python 发邮件用到两个包：smtplib 和 email

Python 的 email 模块里包含了许多实用的邮件格式设置函数，可以用来创建邮件“包裹”。使用的 MIMEText 对象，为底层的 MIME（Multipurpose Internet MailExtensions，多用途互联网邮件扩展类型）协议传输创建了一封空邮件，最后通过高层的 SMTP 协议发送出去。 MIMEText 对象 msg 包括收发邮箱地址、邮件正文和主题，Python 通过它就可以创建一封格式正确的邮件。smtplib 模块用来设置服务器连接的相关信息。

要想通过 QQ 邮箱来发送邮件，需要开启 QQ 邮箱的设置-账户里 SMTP 服务，接下来会通过发送短信验证来获得授权码，有了授权码后就可以在代码里添加了。

接下来看看 QQ 的邮件服务器配置：

根据此配置来设置 smtplib.SMTP_SSL()函数的参数。

（3）问题扩展

使用 Python 发送邮件带附件，以及发送 Html 形式等多种形式。

(https://blog.csdn.net/xiaosongbk/article/details/60142996)

（4）结合项目中使用

```
# coding=utf-8

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

from email.mime.image import MIMEImage

msg_from = 'xxxxx@qq.com'  # 发送方邮箱

passwd = 'xxxxxxxxxxx'  # 填入发送方邮箱的授权码

msg_to = 'xxxx@qq.com'  # 收件人邮箱

def send():

    subject = "Python 邮件测试"  # 主题

    msg = MIMEMultipart('related')

    content = MIMEText('

', 'html', 'utf-8')  # 正文

    # msg = MIMEText(content)

    msg.attach(content)

    msg['Subject'] = subject

    msg['From'] = msg_from

    msg['To'] = msg_to

    file = open("QR.png", "rb")

    img_data = file.read()
```

```
file.close()
img = MIMEImage(img_data)
img.add_header('Content-ID', 'imageid')
msg.attach(img)
try:
    s = smtplib.SMTP_SSL("smtp.qq.com", 465)  # 邮件服务器及端口号
    s.login(msg_from, passwd)
    s.sendmail(msg_from, msg_to, msg.as_string())
```

在 Python 中什么是互斥锁?

题目标签：互斥锁

试题编号：MS006660

（1）问题分析

考官主要想考察的是资源安全这方面线程之间资源是共享的，在多线程的情况下如何保证同一时刻只能有一个线程在操作资源。

（2）核心问题讲解

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。

（3）问题扩展

互斥锁：保证锁定同一个线程，修改这个完整的数据，是用户程序自己的锁。

使用情况：因为使用时暂用时间，影响其他线程等待，所以尽量修改处理块的数据后立即释放锁。

（4）结合项目中使用

在 Flask 中请求钩子的使用

题目标签：Flask 请求钩子

试题编号：MS006661

（1）问题分析

考官主要想考察的是请求钩子的概念，为什么使用请求钩子以及使用场景

（2）核心问题讲解

为了让每个视图函数避免编写重复功能的代码，Flask 提供了通用设施的功能，即请求钩子，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

（3）问题扩展

请求勾子

在客户端和服务器交互的过程中，有些准备工作或扫尾工作需要处理，比如：在请求开始时，建立数据库连接；

在请求开始时，根据需求进行权限校验；

在请求结束时，指定数据的交互格式。

（4）结合项目中使用

1）before_request

可以对用户的权限作校验

可以对接口请求权限做校验

对非法请求做校验

在新经咨询中,使用 before_request 管理员模块做了权限的校验

防止普通用户访问管理员的页面

2）after_request

可以统一的响应做一些值的修改或者添加，再返回给客户端

在新经咨询中,使用 after_request 做了 cookie 中,csrf_token 的统一设置类似于 Django 的中间件

跨站请求伪造过程以及如何实现跨站保护?

题目标签：跨站请求

试题编号：MS006662

（1）问题分析

考官主要想考察是否清楚跨站请求伪造是因为 cookie 状态保持产生的,以及伪造流程。

（2）核心问题讲解

选择攻击原理：

用户 C 访问正常网站 A 时进行登录，浏览器保存 A 的 cookie

用户 C 再访问攻击网站 B，网站 B 上有某个隐藏的链接或者图片标签会自请求网站 A 的 URL 地址,例如表单提交，传指定的参数

而攻击网站 B 在访问网站 A 的时候，浏览器会自动带上网站 A 的 cookie

所以网站 A 在接收到请求之后可判断当前用户是登录状态，所以根据用户的权限做具体的操作逻辑，造成网站攻击成功

防范措施：

在指定表单或者请求头的里面添加一个随机值做为参数。

在响应的 cookie 里面也设置该随机值。

那么用户 C 在正常提交表单的时候会默认带上表单中的随机值，浏览器会自动带上 cookie 里面的随机值，那么服务器下次接受到请求之后就可以取出两个值进行校验。

而对于网站 B 来说网站 B 在提交表单的时候不知道该随机值是什么，所以就形成不了攻击。

（3）问题扩展

CSRF，全称为 Cross-Site Request Forgery，跨站请求伪造，是一种网络攻击方式，它可以在用户毫不知情的情况下，以用户的名义伪造请求发送给被攻击站点，从而在未授权的情况下进行权限保护内的操作。具体来讲，可以这样理解 CSRF。攻击者借用用户的名义，向某一服务器发送恶意请求，对服务器来讲，这一请求是完全合法的，但攻击者确完成了一个恶意操作，比如以用户的名义发送邮件，盗取账号，购买商品等等。

（4）结合项目中使用

使用 flask_wtf 中 CSRFProtect 类，初始化该类并传入 app

使用 flask_wtf.csrf 模块中的 generate_csrf 方法生成 csrf_token

使用请求勾子 after_request，取到响应，统一设置到 cookie 中

如果前端使用 form 表单提交，需要在表单中添加隐藏的 input，并设置其 value 值为：

{{ csrf_token() }}

如果前端使用 ajax 的方式提交，则在 header 中添加 X-CSRFToken 并设置相关值(值从 cookie 中取)

CSRFProtect 的实现原理是：使用请求勾子，before_request 的时候去取到 cookie 里面的值和表单(或者请求头)里面的值进行对比

Django 中使用中间件的方式实现

你们涉及网站的一般并发数是多少？如何解决高并发？

题目标签：网站高并发

试题编号：MS006663

（1）问题分析

考官想考察有没有处理过高并发

（2）核心问题讲解

1）HTML 页面静态化

2）图片服务器分离（我使用的是 fastdfs 轻量级的分布式文件存储系统）

3)使用缓存(缓存存在于内存中读取快我的项目中使用 Redis 作为缓存的数据库，Redis 是内存型数据作为存储缓存的数据库挺适合)

4）数据库集群、库表散列

5）使用负载均衡的方法（简单的配置可以用 nginx 来配置负载均衡，只需要设置 如下代码，即可实现简单的负载均衡

```
upstream djangoserver {
        server 192.168.72.49:8080;
        server 192.168.72.49:8081;
    }
```

6）镜像

镜像是大型网站常采用的提高性能和数据安全性的方式，镜像的技术可以解决不同网络接入商和地域带来的用户访问速度差异，比如 ChinaNet 和 EduNet 之间的差异就促使了很多网站在教育网内搭建镜像站点，数据进行定时更新或者实时更新。在镜像的细节技术方面，这里不阐述太深，有很多专业的现成的解决架构和产品可选。也有廉价的通过软件实现的思路，比如 Linux 上的 rsync 等工具。

（3）问题扩展

有高并发的模块用 go 写， 外加消息队列 rabbitmq

（4）结合项目中使用

常见的反爬措施，具体是怎么实现的?

题目标签：反爬措施

试题编号：MS006666

（1）问题分析

考察有没有反爬经验

（2）核心问题讲解

1）通过 headers 字段来反爬

A）通过 headers 中的 User-Agent 字段来反爬

通过 User-Agent 字段反爬的话，只需要给他在请求之前添加 User-Agent 即可，更好的方式是使用 User-Agent 池来解决,我们可以考虑收集一堆 User-Agent 的方式，或者是随机生成 User-Agent

B）通过 referer 字段或者是其他字段来反爬

通过 referer 字段来反爬，我们只需要添加上即可

C）通过 cookie 来反爬

如果目标网站不需要登录 每次请求带上前一次返回的 cookie，比如 requests 模块的 session

如果目标网站需要登录 准备多个账号，通过一个程序获取账号对应的 cookie，组成 cookie 池，其他程序使用这些 cookie

2）通过 js 来反爬

A）通过 js 实现跳转来反爬

在请求目标网站的时候，我们看到的似乎就请求了一个网站，然而实际上在成功请求目标网站之前，中间可能有通过 js 实现的跳转，我们肉眼不可见，这个时候可以通过点击 perserve log 按钮实现观察页面跳转情况

在这些请求中，如果请求数量很多，一般来讲，只有那些 response 中带 cookie 字段的请求是有用的，意味着通过这个请求，对方服务器有设置 cookie 到本地

B）通过 js 生成了请求参数

对应的需要分析 js，观察加密的实现过程

可以使用 selenium 模块解决

C）通过 js 实现了数据的加密

对应的需要分析 js，观察加密的实现过程,也是使用 selenium 模块实现

3）通过验证码来反爬

通过打码平台或者是机器学习的方法识别验证码，其中打码平台廉价易用，建议使用

4）通过 ip 地址来反爬

同一个 ip 大量请求了对方服务器，有更大的可能性会被识别为爬虫，对应的通过购买高质量的 ip 的方式能够解决

5）其他的反爬方式

A）通过自定义字体来反爬

解决思路：可以尝试切换到手机版试试

B）通过 css 来反爬

如通过 css 掩盖真实数据

（3）问题扩展

（4）结合项目中使用

你所知道的分布式爬虫方案有哪些？（这道题缺少图，请核对答案的完整性）

题目标签：爬虫分布式

试题编号：MS006670

（1）问题分析

考官主要想考察你对分布式爬虫的理解，而不是单单只知道 scrapy 框架。

（2）核心问题讲解

主从式分布爬虫

所谓主从模式，就是由一台服务器充当 master，若干台服务器充当 slave，master 负责管理所有连接上来的 slave，包括管理 slave 连接、任务调度与分发、结果回收并汇总等；每个 slave 只需要从 master 那里领取任务并独自完成任务最后上传结果即可，期间不需要与其他 slave 进行交流。

对于主从分布式爬虫，不同的服务器承担不同的角色分工，其中有一台专门负责对其他服务器提供 URL 分发服务，其他机器则进行实际的网页下载。URL 服务器维护待抓取 URL 队列，并从中获得待抓取网页的 URL，分配给不同的抓取服务器，另外还要对抓取服务器之间的工作进行负载均衡，使得各服务器承担的工作量大致相等，不至于出现忙闲不均的情况。抓取服务器之间没有通信联系，每个待抓取服务器只和 URL 服务器进行消息传递。

主从式分布爬虫

对等式分布爬虫

在对等式分布爬虫体系中，服务器之间不存在分工差异，每台服务器承担相同的功能，各自负担一部分 URL 的抓取工作，下图是其中一种对等式分布爬虫，Mercator 爬虫采用此种体系结构。

由于没有 URL 服务器存在，每台待抓取服务器的任务分工就成为问题。在下图所示的体系结构下，有服务器自己来判断某个 URL 是否应该由自己来抓取，或者将这个 URL 传递给相应的服务器。至于采取的判断方法，则是对网址的主域名进行哈希计算，之后取模（即

hash【域名】%m，这里的 m 为服务器个数），如果计算所得的值和服务器编号匹配，则自己下载该网页，否则将网址转发给对应编号的抓取服务器

对等分布式爬虫（哈希取模）

以上图的例子来说，因为有 3 台服务器，所以取模的时候 m 设定为 3，图中的 1 号抓取服务器负责抓取哈希取模后值为 1 的网页，当其接受到网址 www.google.com 时，首先利用哈希函数计算这个主域名的哈希值，之后对 3 取模，发现取模后值为 1，属于自己的职责范围，于是就自己下载网页；如果接受到网页 www.baidu.com，哈希后对 3 取模，发现值等于 2，不属于自居的职责范围，则会将这个要下载的 URL 转发给 2 号服务器，由 2 号抓取服务器来进行下载。通过这种方式，每台服务器平均承担大约 1/3 的抓取工作量。

由于没有 URL 分发服务器，所以此种方法不存在系统瓶颈问题，另外其哈希数不是针对整个 URL，而只针对主域名，所以可以保证同一网站的网页都由同一台服务器抓取，这样一方面可以提高下载效率（DNS 域名解析可以缓存），另外一方面也可以主动控制对某个网站的访问速度，避免对某个网站访问压力过大。

上图这种体系结构也存在一些缺点，假设在抓取过程中某个服务器宕机，或者此时新加入一台抓取服务器，因为取模时 m 是以服务器个数确定的，所以此时 m 值发生变化，导致发部分 URL 哈希取模后的值跟着变化，这意味着几乎所有任务都需要重新进行分配，无疑会导致资源的极大浪费。

为了解决哈希取模的对等式分布爬虫存在的问题，UbiCrawler 爬虫提出了改进方案，即放弃哈希取模方式，转而采用一致性哈希方法来确定服务器的任务分工。

一致性哈希将网站的主域名进行哈希，映射为一个范围在 0 到 2 的 32 次方之间的某个数值，大量的网站主域名会被平均的哈希到这个数值区间。可以如下图所示那样，将哈希值范围首尾相接，即认为数值 0 和最大值重合，这样可以将其看做有序的环状序列，从数值 0 开始，沿着环的顺时针方向，哈希值逐渐增大，直到环的结尾。而某个抓取服务器则负责这个环装序列的一个片段，即落在某个哈希取值范围内的 URL 都由该服务器负责下载。这样即可确定每台服务器的职责范围。

我们以下图为例说明其优势，假设 2 号抓取服务器接收到了域名 www.baidu.com，经过哈希值计算后，2 号服务器知道在自己的管辖范围内，于是自己下载这个 URL。在此之后，2 号服务器收到了 www.sina.com.cn 这个域名，经过哈希计算，可知是 3 号服务器负责的范围，于是将这个 URL 转发给 3 号服务器。如果 3 号服务器死机，那么 2 号服务器得不到回应，于是知道 3 号服务器出了状况，此时顺时针按照环的大小顺序查找，将 URL 转发给第一个碰到的服务器，即 1 号服务器，此后 3 号服务器的下载任务都由 1 号服务器接管，直接到 3 号服务器重新启动为止。

对等分布式爬虫（一致性哈希）

从上面的流程可知，即使某台服务器出了问题，那么本来应该由这台服务器负责 URL 则由顺时针的下一个服务器接管，并不会对其他服务器的任务造成影响，这样就解决了哈希取模方式的弊端，将影响范围从全局限制到了局部，如果新加入下一台服务器也是如此。

（3）问题扩展

基于 scrapy 的分布式爬虫是主从分布式的一种实现方式。

（4）结合项目中使用

说说你有没有实现过分布式爬虫，采用的是哪种方式实现的？

你在工作中是如何使用或解决的，比如对数据库进行优化，或如何处理表的死锁、慢查询优化？

题目标签：实际问题

试题编号：MS006673

（1）问题分析

一般上只有大公司才会遇到死锁、慢查询优化

（2）核心问题讲解

查询优化就是建立单独索引啊 联合索引啊 或是分库分表之类的 代码层面的就是查询语句的优化了

（3）问题扩展

关于死锁的问题，先查询哪些表被死锁，查询后会返回一个包含 spid 和 tableName 列的表.其中 spid 是进程名,tableName 是表名.然后查下主机名，通过 spid 列的值进行关闭进程.

（4）结合项目中使用

开发过程中从开发到上线的流程是怎么样的？

题目标签：项目开发流程

试题编号：MS006675

（1）问题分析

主要是考察熟悉项目流程，开发一个需求的过程

（2）核心问题讲解

产品先出需求 然后开需求分析会 ，后台再梳理出接口文档 评估开发排期和联调测试上线完就接下一版

（3）问题扩展

从开发到上线这个过程就是上面产品迭代联调测试通过了，就可以部署上线

（4）结合项目中使用

每个功能都需要经历这个过程。

Redis 中 list 底层实现有哪几种？有什么区别？

题目标签：Redis 底层

试题编号：MS006678，删除：MS006904

（1）问题分析

考官主要想考察学员对 Redis 数据的理解和拓展，有没有深入的去理解过数据库。

（2）核心问题讲解

列表对象的编码可以是 ziplist 或者 linkedlist。

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

的区域。

linkedlist 是一种双向链表。它的结构比较简单，节点中存放 pre 和 next 两个指针，还有节点相关的信息。当每增加一个 node 的时候，就需要重新 malloc 一块内存。

（3）问题扩展

获取当前同时还会问 Redis 数据库有几种数据类型，然后深入去问比如有没有深入了解过 Redis，说说 Redis 里面 list 的底层实现，说说 set 数据类型为何支持去重等等，是一类拓展的问题

（4）结合项目中使用

说说你做过的项目里哪些地方用到了 Redis 数据库

为什么要在项目中使用 Redis 数据库

scrapy 和 scrapy-Redis 有什么区别？为什么选择 Redis 数据库？

题目标签：scrappy 与 scrapy-Redis

试题编号：MS006679

（1）问题分析

考官主要想对 snrapy 框架和 scrapy-Redis 的区别进行考察，考察学员对框架以及组件的理解，同时还考察学员对分布式原理的理解，以及对 Redis 数据库的理解

（2）核心问题讲解

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

指纹集合。

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

（3）问题扩展

Scrapy-Reids 就是将 Scrapy 原本在内存中处理的 调度(就是一个队列 Queue)、去重、这两个操作通过 Redis 来实现。

scrapy-redi 重写了 scrapy 一些比较关键的代码，将 scrapy 变成一个可以在多个主机上同时运行的分布式爬虫。

多个 Scrapy 在采集同一个站点时会使用相同的 Redis key（可以理解为队列）添加 Request 获取 Request 去重 Request，这样所有的 spider 不会进行重复采集。

（4）结合项目中使用

说说你哪个项目是用的 scrapy-Redis，哪个项目用到了分布式，然后说说你们用到了几台服务器（电脑）

进程、线程、协程在项目中的使用场景（请完善答案）

题目标签：线程

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=458ABFAFC5BDB0479C33DC5901307461

让线程按顺序执行有几种方法?

题目标签：线程

试题编号：MS006867

（1）问题分析

（2）核心问题讲解

使用线程的 join 方法

join():是 Theard 的方法，作用是调用线程需等待该 join()线程执行完成后，才能继续用下运行。

应用场景：当一个线程必须等待另一个线程执行完毕才能执行时可以使用 join 方法。

（3）问题扩展

使用互斥锁：

对共享数据进行锁定，保证同一时刻只能有一个线程去操作。

应用场景：保证同一时刻只能有一个线程去操作共享数据，只有对公共数据进行访问或者操作的时候是串行模式。

（4）结合项目中使用

设计模式（单例 工厂 ）在工作中的应用场景（非学校学习项目）。

题目标签：设计模式

试题编号：MS006869

（1）问题分析

单例模式在工作中的实际应用。

（2）核心问题讲解

首先介绍一下单例模式：

单例模式（Singleton），也叫单子模式，是一种常用的软件设计模式。在应用这个模式时，单例对象的类必须保证只有一个实例存在。许多时候整个系统只需要拥有一个的全局对象，这样有利于我们协调系统整体的行为。比如在某个服务器程序中，该服务器的配置信息存放在一个文件中，这些配置数据由一个单例对象统一读取，然后服务进程中的其他对象再通过这个单例对象获取这些配置信息。这种方式简化了在复杂环境下的配置管理。育网内搭建镜像站点，数据进行定时更新或者实时更新。在镜像的细节技术方面，这里不阐述太深，有很多专业的现成的解决架构和产品可选。也有廉价的通过软件实现的思路，比如 Linux 上的 rsync 等工具。

应用场景举例：

1）外部资源：每台计算机有若干个打印机，但只能有一个 PrinterSpooler，以避免两个打印作业同时输出到打印机。内部资源：大多数软件都有一个（或多个）属性文件存放系统配置，这样的系统应该有一个对象管理这些属性文件。

2）Windows 的 Task Manager（任务管理器）就是很典型的单例模式（这个很熟悉吧），想想看，是不是呢，你能打开两个 windows task manager 吗?

3）windows 的 Recycle Bin（回收站）也是典型的单例应用。在整个系统运行过程中，回收站一直维护着仅有的一个实例。

4）网站的计数器，一般也是采用单例模式实现，否则难以同步。

5）应用程序的日志应用，一般都何用单例模式实现，这一般是由于共享的日志文件一直处于打开状态，因为只能有一个实例去操作，否则内容不好追加。

6）Web 应用的配置对象的读取，一般也应用单例模式，这个是由于配置文件是共享的资源。

7）数据库连接池的设计一般也是采用单例模式，因为数据库连接是一种数据库资源。数据库软件系统中使用数据库连接池，主要是节省打开或者关闭数据库连接所引起的效率损耗，这种效率上的损耗还是非常昂贵的，因为何用单例模式来维护，就可以大大降低这种损耗。

8）多线程的线程池的设计一般也是采用单例模式，这是由于线程池要方便对池中的线程进行控制。

9）操作系统的文件系统，也是大的单例模式实现的具体例子，一个操作系统只能有一个文件系统。

10）HttpApplication 也是单位例的典型应用。熟悉 ASP.Net(IIS)的整个请求生命周期的人应该知道 HttpApplication 也是单例模式，所有的 HttpModule 都共享一个 HttpApplication 实例。

（3）问题扩展

优点：

1）在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就防止其它对象对自己的实例化，确保所有的对象都访问一个实例。

2）单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。

3）提供了对唯一实例的受控访问。

4）由于在系统内存中只存在一个对象，因此可以 节约系统资源，当 需要频繁创建和销毁的对象时单例模式无疑可以提高系统的性能。

5）允许可变数目的实例。

6）避免对共享资源的多重占用。

缺点：

1）不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。

2）由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

3）单例类的职责过重，在一定程度上违背了“单一职责原则”。

4）滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

（4）结合项目中使用

```
class ConnectionPool:
    __instance = None
    def __init__(self):
        self.addr = '127.0.0.1'
        self.port = 8008
        self.name = 'sql'
        self.conn_list = [1,2,3,4,5,6,7,8,9,10]
```

```
@staticmethod
def get_instance():
    if ConnectionPool.__instance:
        return ConnectionPool.__instance
    else:
        ConnectionPool.__instance == ConnectionPool()
        return ConnectionPool.__instance
def get_connection(self):
    r = random.randrange(1, 11)
    return r
```

以上示例，通过一个私有变量，一个静态方法 get_instance()实现了一次创建对象，多次连接。

es 在项目中如何使用及过程的搭建

题目标签：elasticsearch

试题编号：MS006870

（1）问题分析

（2）核心问题讲解

安装相关包。

1）在工程文件中 settings 中，注册 haystack 应用并添加配置。

2）在被检索的应用目录下创建 search_indexes.py 文件

3）在 templates 下面新建目录 search/indexes/应用名，指定索引字段，使用命令生成索引文件。

4）项目 urls.py 文件配置 url

（3）问题扩展

（4）结合项目中使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=8800C1BFCEA0A69A9C33DC5901307461

django 项目中如何区分是哪个用户发来的请求

题目标签：django

试题编号：MS006871

（1）问题分析

（2）核心问题讲解

每个 Web 请求中都提供一个 request.user 属性来表示当前用户。如果当前用户未登录，则该属性为 AnonymousUser 的一个实例，反之，则是一个 User 实例核心答案讲解：可以通过 is_authenticated()来区分。

（3）问题扩展

（4）结合项目中使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=8636BB1AC9949F0C9C33DC59013074611

es 倒排索引的原理及常用的 API

题目标签：elasticsearch

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=E56A74BA638524F59C33DC5901307461

es 是怎么和数据库关联起来的

题目标签：elasticsearch

试题编号：MS006872

（1）问题分析

（2）核心问题讲解

1）关系型数据库中的数据库（DataBase），等价于 ES 中的索引（Index）。

2)一个数据库下面有 N 张表(Table),等价于 1 个索引 Index 下面有 N 多类型(Type)。

3）一个数据库表（Table）下的数据由多行（ROW）多列（column，属性）组成，等价于 1 个 Type 由多个文档（Document）和多 Field 组成。

4）在一个关系型数据库里面，schema 定义了表、每个表的字段，还有表和字段之间的关系。与之对应的，在 ES 中：Mapping 定义索引下的 Type 的字段处理规则，即索引如何建立、索引类型、是否保存原始索引 JSON 文档、是否压缩原始 JSON 文档、是否需要分词处理、如何进行分词处理等。

5）在数据库中的增 insert、删 delete、改 update、查 search 操作等价于 ES 中的增 PUT、删 Delete、改 POST、查 GET。

（3）问题扩展

（4）结合项目中使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=E9074FEBED1DB6FB9C33DC5901307461

haystack 在使用过程中遇到的问题有哪些?

题目标签：haystack

试题编号：MS006873

（1）问题分析

考官主要想考察学员对于 haystack 的理解，对于 haystack 有没有实际操作经验

（2）核心问题讲解

1）要想添加、删除和修改数据时，自动生成索引，需要在 settings 文件中指定：

HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'

2）haystack 建立数据索引，要先创建索引类。然后创建 text 字段索引值模板文件。最后手动生成初始索引

（3）问题扩展

（4）结合项目中使用

商品全文搜索功能的实现

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=E41EF9B656A4C5E09C33DC5901307461

对 es（elasticsearch）存储底层的原理你是如何理解的

题目标签：elasticsearch

试题编号：MS006874

（1）问题分析

考官主要想考察的是学员对于 elasticsearch 更深层次的理解

（2）核心问题讲解

1）Elasticsearch 对复杂分布式机制的透明隐藏特性

分片机制

shard 副本

集群发现机制

shard 负载均衡

elasticsearch 是分布式的，意味着索引可以被分成分片（仅保存数据的一部分），每个分片可以有 0 个或多个副本。每个节点（运行中的 es 实例）托管一个或多个分片。用户可以将请求发送到集群中的任何节点，每个节点都知道任意文档所处的位置。无论我们将请求发送到哪个节点，它都能负责从各个包含我们所需文档的节点收集回数据，并将最终结果返回給客户端

2）Elasticsearch 的垂直扩容与水平扩容

垂直扩容：采购更强大的服务器，成本非常高昂，而且会有瓶颈，假设世界上最强大的服务器容量就是 10T，但是当你的总数据量达到 5000T 的时候，你要采购多少台最强大的服务器啊

水平扩容：业界经常采用的方案，采购越来越多的普通服务器，性能比较一般，但是很多普通服务器组织在一起，就能构成强大的计算和存储能力

3）节点平等的分布式架构

节点对等，每个节点都能接收所有的请求

自动请求路由

响应收集

4）master 节点

创建或删除索引

增加或删除节点

5）基于_version 进行乐观锁并发控制

搜索的原理：

倒排索引，用来存储在全文搜索下某个单词在文档下的存储位置的映射。分析（分词，标准化），是由分析器完成的。

相关性排序，最相关的排在前面，主要通过检索词频率，反向文档频率，字段长度准则。

（3）问题扩展

（4）结合项目中使用

商品模块中搜索框的使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=B3A4251BBA75790A9C33DC5901307461

celery 队列出现阻塞怎么办（请完善答案）

题目标签：celery

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=83196365E952A57B9C33DC5901307461

rabbitmq 是什么，在项目中具体的使用场景

题目标签：Rabbitmq

试题编号：MS006875，删除：MS006481、MS006569

（1）问题分析

Celery 异步任务的实现方式有三种，其中就有 celery+rabbitmq 的实现方式，面试中经常会提及。

（2）核心问题讲解

RabbitMQ,用在实时的对可靠性要求比较高的消息传递上。学过 websocket 的来理解 rabbitMQ 应该是非常简单的了，websocket 是基于服务器和页面之间的通信协议，一次握手，多次通信。 而 rabbitMQ 就像是服务器之间的 socket，一个服务器连上 MQ 监听，而另一个服务器只要通过 MQ 发送消息就能被监听服务器所接收。但是 MQ 和 socket 还是有区别的，socket 相当于是页面直接监听服务器。而 MQ 就是服务器之间的中转站，例如邮箱，一个人投递信件给邮箱，另一个人去邮箱取，他们中间没有直接的关系，所以耦合度相比 socket 小了很多。

（3）问题扩展

服务器之间通信的相对于其他通信在中间做了一个中间仓库。好处 1：降低了两台服务器之间的耦合，哪怕是一台服务器挂了，另外一台服务器也不会报错或者休克，反正他监听的是 MQ，只要服务器恢复再重新连上 MQ 发送消息，监听服务器就能再次接收。好处 2：MQ 作为一个仓库，本身就提供了非常强大的功能，例如不再是简单的一对一功能，还能一对多，多对一，自己脑补保险箱场景，只要有特定的密码，谁都能存，谁都能取。也就是说能实现群发消息和以此衍生的功能。

（4）结合项目中使用

Django 项目异步任务发送邮件及短信，可在此处介绍 rabbitmq 的使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=3A6323FF3741D2749C33DC5901307461

xadmin 是什么，后台管理的几种实现方式，分别有什么优缺点

题目标签：Xadmin

试题编号：MS006877，删除：MS006577

（1）问题分析

（2）核心问题讲解

xadmin 是开源的一个类似于 django 自带的后台管理系统 admin 的开源模块，它基于 bootstrap3 框架,内置强大的插件系统,根据项目需求可以自定义扩展，比如具有小组件的开发，报表的展示和导入导出等。它比 admin 功能更加丰富，更加便于我们项目的开发。目前只知道这些也没找到相关文档。

后台管理建议多讲，不仅要讲 xadmin 后台管理，同时建议也讲一些不利用 xadmin 如何制作后台，因为一方面一些公司是不用 xadmin 制作后台管理，另一方面，后台管理也是前端和后端的数据进行交互，可以增加学生的项目经验。

（3）问题扩展

（4）结合项目中使用

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=A7E20137A4CDB1FC9C33DC5901307461

odoo 框架在项目中你是如何使用，以及如何进行优化的

题目标签：odoo

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=582BB6D0DC59BE339C33DC5901307461

Django 项目中的分享

题目标签：odoo

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=7B884E17C787423E9C33DC5901307461

如何对 django 框架进行二次开发，增大并发数

题目标签：django

试题编号：MS006878，删除 MS006580

（1）问题分析

考察学员对于 django 框架的理解和对 web 项目高并发的理解

（2）核心问题讲解

二次开发，简单的说就是在现有的软件上进行定制修改，功能的扩展，然后达到自己想要的功能，一般来说都不会改变原有系统的内核。

开发过程中会选择其他组件或者自己开发一部分组件来替代 django 的组件。

关于提高并发量：

在 django 框架中引入 celery 实现分布式异步非阻塞地执行一些需要耗时的任务。

原生 orm 的效率不如直接写 sql 的效率高，必要的时候我们会选择直接用 sql 来实现后端操作，因此提高程序的效率。

缓存组件的引入，引入 drf 框架前后端分离的设计模式，不使用后端渲染页面，减少服务器负载，从而提高并发量。

Xadmin 的引入是二次开发了 django-admin，但是目的是为了拓展更多的功能。

对 django 的文件管理系统进行二次开发，使用 fastdfs 或者第三方文件存储系统替代 django 的文件存储系统，以此来提高并发量。

（3）问题扩展

提高并发量的方面有很多。

前端方面，部分按钮不能让用户不停点击，设置短期失效。

后端方面，缓存、celery 异步、页面静态化、fastdfs 分布式文件存储、第三方文件存储系统。

部署方面：Redis 集群的使用、MySQL 集群的使用、nginx+uwsig 的部署方式。

数据库方面：数据库设计角度、数据库查询角度、数据库配置方案角度。

（4）结合项目中使用

上述方面在项目中都学过，结合我们在项目中所做的操作来答即可。

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=A7E20137A4CDB1FC9C33DC5901307461

Django 项目大文件下载，大多数浏览器不支持，容易报错，该怎么解决

题目标签：django

试题编号：MS006879，删除：MS006592

（1）问题分析

考官主要想考察的是 Django 项目大文件下载，大多数浏览器不支持，容易报错，该怎么解决。

（2）核心问题讲解

```
def file_download(request):
    # do something...
    with open('file_name.txt')asf:
        c=f.read()
    returnHttpResponse(c)
```

这种方式简单粗暴，适合小文件的下载，但如果这个文件非常大，这种方式会占用大量的内存，甚至导致服务器崩溃，因此需要采用更加合理的方式。

更合理的文件下载功能

Django 的 HttpResponse 对象允许将迭代器作为传入参数，将上面代码中的传入参数 c 换成一个迭代器，便可以将上述下载功能优化为对大小文件均适合；而 Django 更进一步，

推荐使用 StreamingHttpResponse 对象取代 HttpResponse 对象，StreamingHttpResponse 对象用于将文件流发送给浏览器，与 HttpResponse 对象非常相似，对于文件下载功能，使用 StreamingHttpResponse 对象更合理。

因此，更加合理的文件下载功能，应该先写一个迭代器，用于处理文件，然后将这个迭代器作为参数传递给 StreaminghttpResponse 对象，如：

```
from django.http import StreamingHttpResponse

def big_file_download(request):
    # do something...

    def file_iterator(file_name,chunk_size=512):
        with open(file_name)asf:
            whileTrue:
                c=f.read(chunk_size)
                ifc:
                    yieldc
                else:
                    break

    the_file_name="file_name.txt"
    response=StreamingHttpResponse(file_iterator(the_file_name))

    returnresponse
```

文件下载功能再次优化

上述的代码，已经完成了将服务器上的文件，通过文件流传输到浏览器，但文件流通常会以乱码形式显示到浏览器中，而非下载到硬盘上，因此，还要在做点优化，让文件流写入硬盘。优化很简单，给 StreamingHttpResponse 对象的 Content-Type 和 Content-Disposition 字段赋下面的值即可，如：

```
response['Content-Type']='application/octet-stream'
response['Content-Disposition']='attachment;filename="test.pdf"'
```

完整代码如下：

```python
from django.http import StreamingHttpResponse

def big_file_download(request):
    # do something...

    def file_iterator(file_name,chunk_size=512):
        with open(file_name)asf:
            whileTrue:
                c=f.read(chunk_size)
                ifc:
                    yieldc
                else:
                    break

    the_file_name="big_file.pdf"
    response=StreamingHttpResponse(file_iterator(the_file_name))
    response['Content-Type']='application/octet-stream'
    response['Content-Disposition']='attachment;filename="
{0}"'.format(the_file_name)

    returnresponse
```

（3）问题扩展

（4）结合项目中使用

Django 项目大文件下载，大多数浏览器不支持，容易报错。

Django 后台管理权限设置问题，怎么跟菜单或者网页按钮关联?

题目标签：django

试题编号：MS006880，删除：MS006597

（1）问题分析

考官主要想考察的是比如说一个网页有几个按钮（添加，编辑，删除，查看），张三用户可以访问这几个按钮，李四只能查看（没有其它权限），怎么办？考官相信对于初学者来说，完成这样的需求，确实有点难度，所以考官把这个流程跟大家分享一下。

（2）核心问题讲解

首先设计表结构

创建一个用户表，并且做了一对一关联 django user 表

1）自定义用户表

```
from __future__ import unicode_literals

from django.db import models

from django.contrib.auth.models import User

# Create your models here.

#自定义用户表

class Userinfo(models.Model):

    user = models.OneToOneField(User) #关联 django user 表

    username = models.CharField(max_length=100)

    password = models.CharField(max_length=100)

    def __unicode__(self):

        return self.username
```

2）自定义权限表

```
class quanxian(models.Model):

    shuoming=models.CharField(max_length=100)

    def __unicode__(self):

        return self.shuoming

    class Meta:
```

```
        permissions = (
            ('edit', u'编辑权限'),
            ('add', u'添加权限'),
            ('DEL',u'删除权限'),
            ('list',u'查看权限'),
        )
```

3）views.py 定义一个 login 视图方法

```
from test01.models import Userinfo
from django import forms
from django.contrib import auth
from django.contrib.auth.models import User
from django.contrib.auth import authenticate
from django.template import RequestContext
# Create your views here.
class UserForm(forms.Form):
    username = forms.CharField(label="user",max_length=100)
    password = forms.CharField(label="passwd",widget=forms.PasswordInput())
def index(request):
    return render_to_response('index.html')
def login(request):
    if request.method == 'POST':
```

```
        uf = UserForm(request.POST)
        if uf.is_valid():
username = uf.cleaned_data['username']
            password = uf.cleaned_data['password']
            print username,password,"[*******]"
            user1 = authenticate(username=username, password=password)
            is_add = True if user1.has_perm('test01.add') else False
            print 'user1--->',user1,user1.has_perm('test01.add'), is_add
            if user1:
                return render_to_response('index.html',locals(),
context_instance=RequestContext(request))
            else:
                return HttpResponseRedirect('/login/')
    else:
        uf = UserForm()
return render_to_response('login.html',{'uf':uf})
```

4）在 admin.py 注册 models.py 里的表

```
from django.contrib import admin
from test01 import models
admin.site.register(models.quanxian)
admin.site.register(models.Userinfo)
```

5）urls.py

```
from django.conf.urls import url

from django.contrib import admin

from test01 import views

urlpatterns = [

    url(r'^admin/', admin.site.urls),

    url(r'^index/$',views.index),



    url(r'^$', views.login, name='login'),

]
```

6）初始化数据表，创建后台 admin 管理员

```
python manage.py   makemigrations

python manage.py   migrate
```

创建后台 admin 管理员

```
bogon:model_test will.xin$ python manage.py createsuperuser

Username (leave blank to use 'will.xin'): admin

Email address:

Password:

Password (again):

Superuser created successfully.
```

7）登录 admin 后台,创建用户

http://127.0.0.1:8000/admin/

8）前端页面

vim login.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/D
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>登录</title>
</head>
 <!--style type="text/css">
    body{color:#efd;background:#453;padding:0 5em;margin:0}
    h1{padding:2em 1em;background:#675}
    h2{color:#bf8;border-top:1px dotted #fff;margin-top:2em}
    p{margin:1em 0}
  </style-->
<body>
<h1>登录页面: </h1>
<form method = 'post' enctype="multipart/form-data">
    `uf`.`as_p`
    <input type="submit" value = "ok" />
</form>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>index</title>
</head>
<body>

{% if is_add %}
    <button>编辑</button>
    <button>添加</button>
    <button>删除</button>
{% else %}
    <button>查看</button>
{% endif %}


<div>欢迎{{ username }} 登录</div>
</body>
</html>
```

测试成功

（3）问题扩展

利用 Django 搭建公司后台系统，在开发中遇到数据分页（django 原生翻页），后台自定义页面、搜索功能（基于日期单搜索和日期项目名称多选项搜索）、数据显示（BootstrapTable 翻页）、权限（控制表）等问题。

（4）结合项目中使用

django 后台管理权限

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=1B318D8179D8FD5

39C33DC5901307461

Django 中的自定义过滤器，可以传几个参数？为什么？

题目标签：django

试题编号：MS002056

（1）问题分析：

无

（2）核心答案讲解：

自定义过滤器只是一个接受一个或两个参数的 Python 函数：变量的值（输入）并不必要是一个字符串。参数的值，这个可以有一个默认的值或者完全留空。

（3）问题扩展：

无

（4）结合项目中使用：

无

业务实现：Django 中用户上传头像，怎样避免图片名重复？

题目标签：django

试题编号：MS002057

（1）问题分析：

无

（2）核心答案讲解：

MD5 时间戳，图片名称可能会重复，但是上传图片的时间生成的 MD5 字符串是唯一的，可以以此来作为图片保存的方式，就避免了图片重名导致覆盖的惨剧

（3）问题扩展：

无

（4）结合项目中使用：

无

Python 中的 unittest 是什么?

题目标签：单元测试

试题编号：MS002152

（1）问题分析：

无

（2）核心答案讲解：

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

（3）问题扩展：

无

（4）结合项目中使用：

无

具体阐述 MVC 和 MTV 框架，并说明 Django 框架的 MTV 模式的流程思想

题目标签：Django

试题编号：MS002405

（1）问题分析：

无

（2）核心答案讲解：

MVC，全名是 Model View Controller，是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型(Model)、视图(View)和控制器(Controller)，具有耦合性低、重用性高、生命周期成本低等优点。

Django 的 MTV 模式

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，它的作用是将一个个 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。

（3）问题扩展：

无

（4）结合项目中使用：

无

web 基础：什么是 Restful API?

题目标签：restful

试题编号：MS002259

（1）问题分析：

无

（2）核心答案讲解：

Restful API 是目前比较成熟的一套互联网应用程序的 API 设计理念，Rest 是一组架构约束条件和原则，如何 Rest 约束条件和原则的架构，我们就称为 Restful 架构，Restful 架构具有结构清晰、符合标准、易于理解以及扩展方便等特点，受到越来越多网站的采用!

Restful API 接口规范包括以下部分：

1）协议

API 与用户的通信协议，总是使用 HTTPs 协议。

2）域名

应该尽量将 API 部署在专用域名之下，如 https://api.oldboyedu.com;如果确定 API 很简单，不会有进一步扩展，可以考虑放在主域名下，如 https://oldboyedu.com/api/。

3）版本

可以将版本号放在 HTTP 头信息中，也可以放入 URL 中，如 https://api.oldboyedu.com/v1/

4）路径

路径是一种地址，在互联网上表现为网址，在 RESTful 架构中，每个网址代表一种资源(resource)，所以网址中不能有动词，只能有名词，而且所用的名词往往与数据库的表格名对应。一般来说，数据库中的表都是同种记录的"集合"(collection)，所以 API 中的名词也应该使用复数，如 https://api.oldboyedu.com/v1/students。

5）HTTP 动词

对于资源的具体操作类型，由 HTTP 动词表示，HTTP 动词主要有以下几种，括号中对应的是 SQL 命令。

A）GET(SELECT)：从服务器取出资源(一项或多项)；

B）POST(CREATE)：在服务器新建一个资源；

C）PUT(UPDATE)：在服务器更新资源(客户端提供改变后的完整资源)；

D）PATCH(UPDATE)：在服务器更新资源(客户端提供改变的属性)；

E）DELETE(DELETE)：从服务器删除资源；

F）HEAD：获取资源的元数据；

G）OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。

6）过滤信息

如果记录数量很多，服务器不可能都将它们返回给用户，API 会提供参数，过滤返回结果，常见的参数有：

A）?limit=20：指定返回记录的数量为 20；

B）?offset=8：指定返回记录的开始位置为 8；

C）?page=1&per_page=50：指定第 1 页，以及每页的记录数为 50；

D）?sortby=name&order=asc：指定返回结果按照 name 属性进行升序排序；

E）?animal_type_id=2：指定筛选条件。

7）状态码

服务器会向用户返回状态码和提示信息，以下是常用的一些状态码：

A）200 OK - [GET]：服务器成功返回用户请求的数据；

B）201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功；

C）202 Accepted - [*]：表示一个请求已经进入后台排队(异步任务)；

D）204 NO CONTENT - [DELETE]：用户删除数据成功；

E）400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作；

F）401 Unauthorized - [*]：表示用户没有权限(令牌、用户名、密码错误)；

G）403 Forbidden - [*] 表示用户得到授权(与 401 错误相对)，但是访问是被禁止的；

H）404 NOT FOUND - [*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作；

I）406 Not Acceptable - [GET]：用户请求的格式不可得；

J）410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的；

K）422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误；

L）500 INTERNAL SERVER ERROR - [*]：服务器发生错误，用户将无法判断发出的请求是否成功。

8）错误处理

如果状态码是 4xx，就会向用户返回出错信息，一般来说，返回的信息中将 error 作为键名，出错信息作为键值。

9）返回结果

针对不同操作，服务器向用户返回的结果应该符合以下规范：

A）GET /collection：返回资源对象的列表(数组)；

B）GET /collection/resource：返回单个资源对象;

C）POST /collection：返回新生成的资源对象;

D）PUT /collection/resource：返回完整的资源对象;

E）PATCH /collection/resource：返回完整的资源对象;

F）DELETE /collection/resource：返回一个空文档。

10）Hypermedia API

RESTful API 最好做到 Hypermedia，即返回结果中提供链接，连向其他 API 方法，使得用户不查文档，也知道下一步应该做什么。

以上是 Restful API 设计应遵循的十大规范，除此之外，Restful API 还需注意身份认证应该使用 OAuth 2.0 框架，服务器返回的数据格式，应该尽量使用 JSON，避免使用 XML。

（3）问题扩展：

无

（4）结合项目中使用：

无

你公司的服务器用的是什么， fastdfs 分布式存储具体怎么实现存储在多个服务器上，服务器之间怎么交互

题目标签：FastDFS

试题编号：MS006882

（1）核心问题讲解

使用阿里云服务器。linux 环境 Ubuntu16.04

1）在指定目录下创建 fastdfs 安装文件夹

mkdir /wjw #下载 tar.zp 软件包并安装

mkdir /wjw/testfile #上载文件测试文件夹

mkdir /wjw #fastdfs 根文件

mkdir /wjw/fastdfs/track #tracker 文件配置路径

mkdir /wjw/fastdfs/storage #storage 配置路径

mkdir /wjw/fastdfs/clintlog #client 配置路径

2）安装 libfastcommon

A）解压安装 libfastcommon

https://github.com/happyfish100/libfastcommon/archive/V1.0.7.tar.gz

tar -zxvf V1.0.7.tar.gz

cd libfastcommon-1.0.7

./make.sh

./make.sh install

B）安装 FastDFS

https://github.com/happyfish100/fastdfs/archive/V5.05.tar.gz

tar -zxvf V5.05.tar.gz

cd fastdfs-5.05

./make.sh

./make.sh install

3）Tracker、Storage、Client、HTTP 服务

A） 配置 Tracker 服务

cd /etc/fdfs

ls

cp tracker.conf.sample tracker.conf # 配置跟踪文件

vi tracker.conf # 进入 conf 文件

修改配置文件

base_path=/data/fastdfs/track # 修改跟踪路径

http.server_port=80 # 修改端口号

方式启动服务，查看监听：

export LD_LIBRARY_PATH=/usr/lib64/

/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf

netstat -unltp|grep fdfs #查看服务

B）配置 Storage 服务

cd /etc/fdfs

cp storage.conf.sample storage.conf # 修改存储路径

vi storage.conf # 修改存储文件

group_name=group1 # 修改组名

base_path=/data/fastdfs/storage # 修改存储路径

store_path0=/data/fastdfs/storage #这里可以设置多个存储服务器

tracker_server=10.0.2.15:22122 # 改为本地 ip，查看本地 ip：ifconfig

http.server_port=8888 # 设置端口号，如果没有冲突尽量默认

启动查看 storage 服务

/usr/bin/fdfs_storaged /etc/fdfs/storage.conf

netstat -unltp|grep fdfs #查看服务

C）配置 Clint 服务

cd /etc/fdfs

cp clint.conf.sample clint.conf #修改客户端路径文件

vi clint.conf

进入 conf 文件后完成以下参数的修改：

base_path=/wjw/fastdfs/clientlog #设置客户端存储路径

tracker_server=10.0.2.15:22122 #改为本地 ip

http.tracker_server_port=80

#include http.conf 注意，#include http.conf 这句，原配置文件中有 2 个#，删掉一个。

D）置 HTTP 服务

cp /software/fastdfs-5.05/conf/http.conf /etc/fdfs/http.conf # 配置 http 服务文件

cd /etc/fdfs

vi http.conf

进入 conf 文件后完成以下参数的修改：

http.anti_steal.token_check_fail=/data/fastdfs/httppic/anti-steal.jpg

4）设置环境变量和软链接

export LD_LIBRARY_PATH=/usr/lib64/

5）启动 fastdfs

/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf

/usr/bin/fdfs_storaged /etc/fdfs/storage.conf

测试 Tracker 和 Storage 服务通信

/usr/bin/fdfs_monitor /etc/fdfs/storage.conf

6）测试上传文件功能

cd /software/testfile

ls

touch test.txt

vi test.txt

fdfs_test /etc/fdfs/client.conf upload

/wjw/fastdfs/fastdfs-5.05/conf/anti-steal.jpg

此时浏览器浏览不能直接访问因为还需要配置 nginx 进行文件上传下载功能

7）安装 Nginx 和 fastdfs-nginx-module,pcre,zlib

https://nginx.org/download/nginx-1.10.1.tar.gz

在安装 Nginx 之前，需要安装如下（gcc/pcre/zlib/openssl）插件

dpkg -l | grep zlib

插件安装

openssl 安装：

sudo apt-get install openssl libssl-dev

pcre 安装：

sudo apt-get install libpcre3 libpcre3-dev

sudo apt-get install openssl libssl-dev

zlib 安装：

sudo apt-get install zlib1g-dev

gcc 安装：

sudo apt-get install build-essential

https://github.com/happyfish100/fastdfs-nginx-module/archive/master.tar.gz

解压并修改 congfig

编辑配置文件 config

命令： vim /wjw/fast/fastdfs-nginx-module/src/config

修改内容：去掉下图中的 local 文件层次

安装 nginx

tar -zxvf nginx-1.10.1.tar.gz

cd nginx-1.10.1

./configure --add-module=/wjw/fastdfs/ fastdfs-nginx-module/src/

make

make install

启动、停止 nginx

cd /usr/local/nginx/sbin/

./nginx #启动

./nginx -s stop #此方式相当于先查出 nginx 进程 id 再使用 kill 命令强制杀掉进程

./nginx -s quit #:此方式停止步骤是待 nginx 进程处理任务完毕进行停止。

./nginx -s reload

8）配置 fastdfs-nginx-module 和 Nginx

A）配置 mod-fastdfs.conf，并拷贝到/etc/fdfs 文件目录下

cd /wjw/fastdfs/fastdfs-nginx-module/src

vi mod_fastdfs.conf

对 conf 文件作以下修改

valid only when load_fdfs_parameters_from_tracker is true

tracker_server=116.196.117.183:22122

default value is false

url_have_group_name = true

must same as storage.conf

store_path0=/wjw/fastdfs/storage

#store_path1=/home/yuqing/fastdfs1

复制文件到/etc/fdfs 目录

cp mod_fastdfs.conf /etc/fdfs

拷贝以下文件到/etc/fdfs

A）进入 fastdfs conf 目录下

cd /wjw/fastdfs/fastdfs-5.05/conf

B）拷贝 anti-steal.jpg http.conf mime.types 到 /etc/fdfs

cp anti-steal.jpg http.conf mime.types /etc/fdfs/

C）配置 Nginx,编辑 nginx.config

加入

location /group1/M00 {

root /wjw/fastdfs/storage/;

ngx_fastdfs_module;

}

由于我们配置了 group1/M00 的访问，我们需要建立一个 group1 文件夹，并建立 M00 到 data 的软链接。

mkdir /wjw/fastdfs/storage/data/group1

ln -s /wjw/fastdfs/storage/data /wjw/fastdfs/storage/data/group1/M00

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=C6CEB3655761B2619C33DC5901307461

公司项目开发中，一个大项目有 400 多个分支，自己负责开发的时候，是每个人都需要建立一个自己的分支吗（GIT）？

题目标签：git 工作流

试题编号：MS006884，删除：MS006610

（1）问题分析：

考官主要想考察的是学生在之前公司的开发经验问题，以及版本控制器（git）在开发

中的运用。

（2）核心答案讲解：

一个大型项目中可以有多个服务，每个服务都可以有一个小组团队进行开发，一般 Python 的小组开发人员配置举例说明（一个组长，两个后端 Python 工程师，一个前端工程师，一个新人）。

以一个 20 人的团队举例说明，可以分为 4 个项目小组，一个项目经理负责。4 个小组可以并行开发多个服务 (以云计算平台 openstack 开发为例) : 分为 OSS BSS 消息服务 底层服务（计算 Nova、对象存储 Swift、镜像服务 Glance、身份服务 Kestone、网络服务 Network、块存储 Cinder、UI 界面、测量 Ceilometer、部署编排 Heat、数据库服务 Trove 等），相当于多人对多服务进行开发。每个人在对一个服务进行修改的时候，为了不影响该服务的现有功能逻辑，都会开启一个新的分支，进行开发，并打上相应的 tag 标签，一旦这个阶段的功能开发完成后，通过两个以上人的+1 操作，可以合进当前版本的开发分支，并将之前个人的开发分支进行删除。

综上所述：对服务开发新功能的时候需要建立自己的分支，新的功能一旦通过验收后，合并进 devlop 分支时，为了分支管理的清晰需要将自己的分支进行删除，合并分支后，会在 devlop 分支上保存合并记录。

（3）问题扩展：

400 分支的由来，大型项目多数都是版本迭代出来的，一般版本命名方式为：X_Y_Z（如：2_3_1），X：通常为一个大版本,周期为 1 年，Y：通常为一个小版本，周期为 1 个季度，Z 通常问特殊表示。每一个版本必须有一个分支与其对应

工程的项目分支命令：

master 分支为总分支 由项目经理负责，为项目打版使用（打版本）

devlop 分支为开发分支 由项目组长负责，也为现阶段开发主分支

test 分支为测试分支 由测试组长负责，相对稳定的分支，提供给测试进行专门测试使用

每个服务理论上都会有这三个分支

个人分支，以当前的功能名称命名（例如 f/user_update_0921） 指定项目 f 下/用户更新功能_开发时间为 9 月 21 号。开发人员在开发当前功能的时候，会创建该分支，在功能开发完成后，将分支合并如 devlop 分支，并将当前分支删除，如果该功能为不稳定功能，可以保留该分支（加上说明），并在使用的时候 merge devlop 分支。

（4）结合项目中使用：

1、

```
#先创建一个分支
git branch f/2160904_router_update
#切换到指定分支
git checkout
#将本地分支提交映射到远程分支
git push origin f/2160904_router_update:f/2160904_router_update
#关联远程分支和本地指定分支
git branch --set-upstream-to=origin/f/2160904_router_update f/2160904_rou
```

2、

```
#切换到指定分支
```

```
git checkout develop

#拉去远程分支到本地仓储

git pull

#直接创建本地分支，并进行切换

git checkout -b  新分支名称

#将新创建的分支，推送到远程 git 服务上面

git push origin  新分支名称

#关联远程分支和本地分支

git branch --set-upstream-to=origin/远程分支名称  本地分支名称
```

3、

```
#提交缓存

git add .

#提交的本地的仓库

git commit -m 'wanghao First Comment'

#查看分支

git branch

#切回到 dev 的分支中

git checkout dev

#指定分支与当前分支进行 merge 操作

git merge --no-ff  指定分支

#删除无效分支
```

git branch -d 无效分支

#将仓储中的数据进行提交

git push origin dev

#临时挂起 本地操作的代码

git stash

#同步远程分支

git pull

git pull --progress --no-stat -v --progress origin develop

#同步指定远程分支到本地分支

git pull origin [remote] [branch]

#合并对应的代码操作

git stash pop

4、

#查看当前分支的日志

git log

#提取分支的 core 编号，进行对应的 cherry-pick 操作

git check-pick -x -n 提取的提交号

(8f34737bf4bad1746873bd7d4394ae5c4e5ce0bf)

5、

#删除本地分支操作

git branch -d 本地分支名称

```
#查看对应的远程分支

git branch -r

#删除远程分支操作

git push origin --delete 远程分支名称

#本地分支进行重命名

git branch -m 旧名称 新名称

#产看变更状态

git status

#批量删除本地分支

git branch | grep 'bran '| xargs git branch -d
```

怎么监控日志文件，发邮件吗？具体如何实现，使用哪种技术？

题目标签：监控日志文件，发送邮件

试题编号：MS006886，删除：MS006622

（1）问题分析：

需要对应用程序的日志进行实时分析，当符合某个条件时就立刻报警，而不是被动等待出问题后去解决，分析出最耗时的请求，然后去改进代码。

（2）核心答案讲解：

Python 监控有个很好的三方库 sentry 对各种框架都支持集成也很简单 几行代码基本上能够满足。

（3）问题扩展：

可以做下日志分析，最简单的 Linux 下有 tail 命令

1）采用 Python 对文件的操作来实现，用文件对象的 tell(), seek()方法分别得到当前文件位置和要移动到的位置；

2 ）利用 Python 的 yield 来实现一个生成器函数，然后调用这个生成器函数，这样当日志文件有变化时就打印新的行。

（4）结合项目中使用：

无。

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=18F9E078C650177D9C33DC5901307461

业务需求是做音乐和视频的，这样的前后端交互如何开发?

题目标签：业务实现

试题编号：MS006888，删除：MS006634

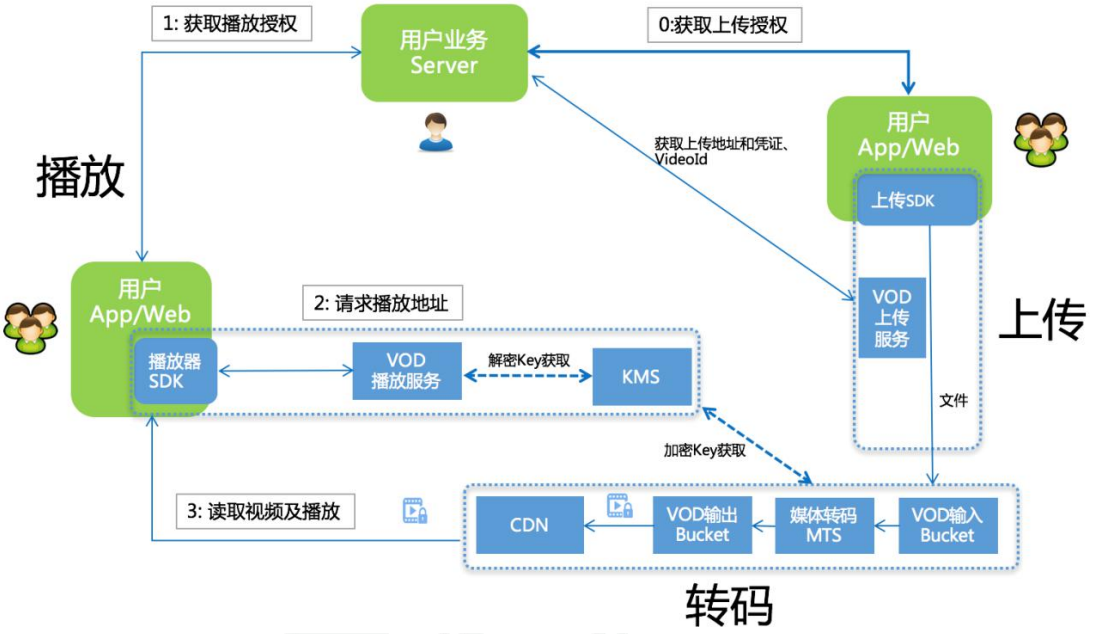
（1）问题分析：

考官主要是想考察你是否做过音频和视频之类的数据如何在后端保存，前端展示的相关业务。

（2）核心答案讲解：

像音频和视频之类的占内存数据，我们一般不会直接存储在数据库中，而是存储到一个专门的服务器，数据库只是存储该数据的路径。

在项目中，我们通过使用视频点播实现音视频上传到阿里云的 OSS（对象存储）服务器中存储，后端数据库中保存视频的 id；然后前端引入阿里云播放器的 js 文件，创建播放器对象进行操作。使用视频点播实现音视频上传、存储、处理和播放的整体流程如下：



用户获取上传授权。

VoD 下发 上传地址和凭证 及 VideoId。

用户上传视频保存视频 ID(VideoId)。

用户服务端获取播放凭证。

VoD 下发带时效的播放凭证。

用户服务端将播放凭证下发给客户端完成视频播放。

（3）问题扩展：

视频点播：阿里云视频点播（ApsaraVideo for VoD）是集音视频采集、编辑、上传、自动化转码处理、媒体资源管理、高效云剪辑处理、分发加速、视频播放于一体的一站式音

视频点播解决方案，整体服务构建在阿里云强大的基础设施服务之上，提供端到端的视频全链路服务，帮助企业和开发者快速搭建安全、弹性、高可定制的视频点播平台和应用。

MTS：媒体处理（ApsaraVideo Media Processing，原 MTS）是一种多媒体数据处理服务。它以经济、弹性和高可扩展的转换方法，将多媒体数据转码成适合在全平台播放的格式。并基于海量数据深度学习，对媒体的内容、文字、语音、场景多模态分析，实现智能审核、内容理解、智能编辑。

CDN：全称是 Content Delivery Network，即内容分发网络。CDN 是构建在网络之上的内容分发网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。CDN 的关键技术主要有内容存储和分发技术。

（4）结合项目中使用：

无

开发过程中从开发到上线的流程是怎么样的?

题目标签：开发流程

试题编号：MS006889

（1）问题分析：

主要是考察熟悉项目流程，开发一个需求的过程。

（2）核心答案讲解：

产品先出需求 然后开需求分析会 ，后台再梳理出接口文档 评估开发排期和联调测试上线完就接下一版。

（3）问题扩展：

从开发到上线这个过程就是上面产品迭代联调测试通过了，就可以部署上线。

（4）结合项目中使用：

每个功能都需要经历这个过程。

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=911225010F02F6E99C33DC5901307461

WSGI 和 nigix 的详细部署?

题目标签：WSGI 和 nginx

试题编号：MS006890，删除 MS006585

（1）问题分析：

考官主要是否参与过项目的部署。如参加过项目部署之后对 WSGI 和 Nginx 的理解和应用。

（2）核心答案讲解：

WSGI：全拼为 Python Web Server Gateway Interface，Python Web 服务器网关接口，是 Python 应用程序或框架和 Web 服务器之间的一种接口，被广泛接受。WSGI 没有官方的实现，因为 WSGI 更像一个协议，只要遵照这些协议，WSGI 应用(Application)都可以在任何服务器(Server)上运行。

项目默认会生成一个 wsgi.py 文件，确定了 settings 模块、application 对象。

application 对象：在 Python 模块中使用 application 对象与应用服务器交互。

settings 模块：用于进行项目配置。

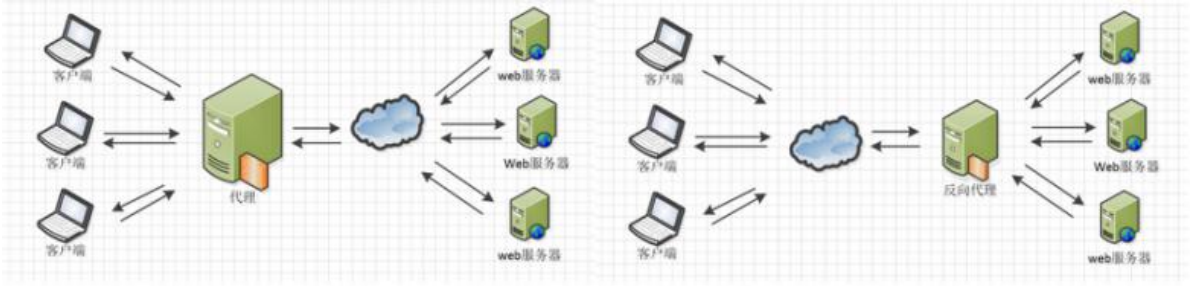
Nginx：是 Igor Sysoev 为俄罗斯访问量第二的 rambler.ru 站点设计开发的。从 2004 年发布至今，凭借开源的力量，已经接近成熟与完善。

Nginx 功能丰富，可作为 HTTP 服务器，也可作为反向代理服务器，邮件服务器。支持 FastCGI、SSL、Virtual Host、URL Rewrite、Gzip 等功能。并且支持很多第三方的模块扩展。

Nginx 的稳定性、功能集、示例配置文件和低系统资源的消耗让他后来居上，在全球活跃的网站中有 12.18%的使用比率，大约为 2220 万个网站。

Nginx 常用功能

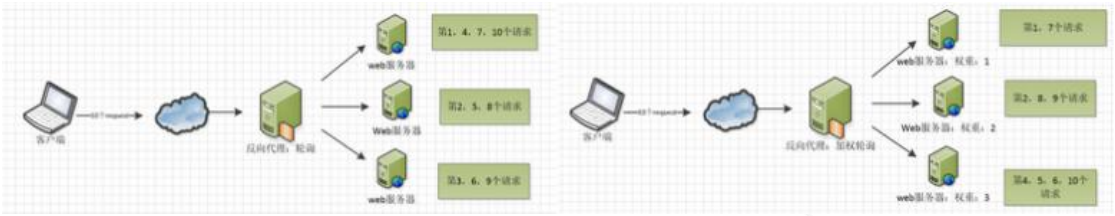
1）Http 代理，反向代理：作为 web 服务器最常用的功能之一，尤其是反向代理。



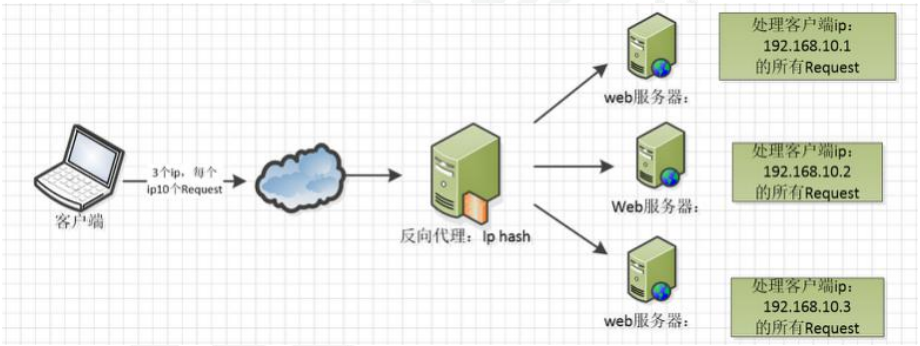
Nginx 在做反向代理时，提供性能稳定，并且能够提供配置灵活的转发功能。Nginx 可以根据不同的正则匹配，采取不同的转发策略，比如图片文件结尾的走文件服务器，动态页面走 web 服务器，只要你正则写的没问题，又有相对应的服务器解决方案，你就可以随心所欲的玩。并且 Nginx 对返回结果进行错误页跳转，异常判断等。如果被分发的服务器存在异常，他可以将请求重新转发给另外一台服务器，然后自动去除异常服务器。

2）负载均衡

Nginx 提供的负载均衡策略有 2 种：内置策略和扩展策略。内置策略为轮询，加权轮询，Ip hash。扩展策略，你可以参照所有的负载均衡算法，找出来做下实现



Ip hash 算法，对客户端请求的 ip 进行 hash 操作，然后根据 hash 结果将同一个客户端 ip 的请求分发给同一台服务器进行处理，可以解决 session 不共享的问题。



3）web 缓存

Nginx 可以对不同的文件做不同的缓存处理，配置灵活，并且支持 FastCGI_Cache，主要用于对 FastCGI 的动态程序进行缓存。配合着第三方的 ngx_cache_purge，对制定的 URL 缓存内容可以的进行增删管理。

（3）问题扩展：

uwsgi 和 uWSGI 的理解

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 协议是一个 uWSGI 服务器自有的协议，它用于定义传输信息的类型

uWSGI 是一个全功能的 HTTP 服务器，他要做的就是把 HTTP 协议转化成语言支持的网络协议。比如把 HTTP 协议转化成 WSGI 协议，让 Python 可以直接使用。

uwsgi 是一种 uWSGI 的内部协议，使用二进制方式和其他应用程序进行通信。

（4）结合项目中使用：

项目具体怎么实现了反向代理呢

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=D70601E57786D2EF9C33DC5901307461

nginx 是如何进行分流处理的?

题目标签：nginx

试题编号：MS006891

（1）问题分析：

考察 nginx 在真实的项目中是如何使用的，以及对分流的处理方式。

（2）核心答案讲解：

1）根据 IP 分流

2）根据 URL 分流

3）根据权重

4）根据响应时间

NGINX 负载均衡分发请求的几种方式

1）轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉，能自动剔除。

2）weight

指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况。

3）ip_hash

每个请求按访问 ip 的 hash 结果分配，这样每个访客固定访问一个后端服务器，可以解决 session 的问题。

4）fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

5）url_hash（第三方）

按访问 url 的 hash 结果来分配请求，使每个 url 定向到同一个后端服务器，后端服务器为缓存时比较有效。

```
upstream www.test1.com {
    ip_hash;
    server 172.16.125.76:8066 weight=10;
    server 172.16.125.76:8077 down;
    server 172.16.0.18:8066 max_fails=3 fail_timeout=30s;
    server 172.16.0.18:8077 backup;
}
```

根据服务器的本身的性能差别及职能，可以设置不同的参数控制。

（3）问题扩展：

down 表示负载过重或者不参与负载

weight 权重过大代表承担的负载就越大

backup 其它服务器时或 down 时才会请求 backup 服务器

max_fails 失败超过指定次数会暂停或请求转往其它服务器

fail_timeout 失败超过指定次数后暂停时间

（4）结合项目中使用：

无。

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=8EB79B5186070B379C33DC59013074l61

docker 里部署 django 项目?

题目标签：Docker

试题编号：MS006892，删除：MS006605

（1）问题分析：

考察 docker 的使用情况，以及如何将 Django 项目部署到 docker 并成功运行。

（2）核心答案讲解：

为什么使用 docker?

1）更高效的利用系统资源

2）可迁移性高

3）更快速的启动时间

4）一致的运行环境

开始部署：

apt install docker.io

在此之前你需要熟悉 docket 的基本用法

安装 django 镜像

docker search django

docker pull django

docker run --name django_uwsgi_nginx -p 80:80 -it django /bin/bash

进入 django 容器

由于是 ubuntu 系统环境，可以先对系统更新

apt update （嫌慢可以放上国内的源）

安装几个东西

apt install nginx　　apt install git (如果你的 django 项目放在了 git 上，可以使用 git clone 下载)

apt install net-tools （docker 网络管理包 包括 netstat 命令）

容器中的 django 是 1.x 版本的，需要更新到 2.0 版本的

pip3 uninstall django

pip3 install django

1）开始部署 uwsgi

在 django 项目的根录目下 创建 ini 文件 如 website.ini

写入：

```
socket          =127.0.0.1:9090    #9090 连接 nginx
chdir           = /root/website
module          =website.wsgi
master          = true
processes       = 4
vacuum          = true
```

尝试启动：

```
uwsgi --ini website.ini
```

2）配置 nginx

打开/etc/nginx/nginx.conf

在 http 代码块写入

```
server {
        listen 80;
        server_name xxxx(域名);
        location / {            #连接 uwsgi
           uwsgi_pass 127.0.0.1:9090;
           include uwsgi_params;
```

```
            uwsgi_param UWSGI_CHDIR /root/website;
            uwsgi_param UWSGI_SCRIPT website.wsgi;
        }
        location /static {
            alias /root/website/static;
        }
        access_log off;
    }
```

开启服务：

```
uwsgi --ini website.ini & nginx
```

访问域名

（3）问题扩展：

无

（4）结合项目中使用：

无

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=5670CB61D7630ED09C33DC5901307461

docker 私有仓库怎么搭建，常见的 dockfile 如何编写，实现什么功能?

题目标签：Docker

试题编号：MS006893

此题答案

（1）问题分析：

无

（2）核心答案讲解：

环境准备

环境：两个装有 Docker 的 Ubuntu 虚拟机

虚拟机一：192.168.112.132 用户开发机

虚拟机二：192.168.112.136 用作私有仓库

此处我们准备了两个虚拟机，分别都安装了 Docker，其中 132 机器用作开发机，136 机器用作 registry 私有仓库机器。环境准备好之后接下来我们就开始搭建私有镜像仓库。

搭建私有仓库

首先在 136 机器上下载 registry 镜像

$ sudo docker pull registry

下载完之后我们通过该镜像启动一个容器

$ sudo docker run -d -p 5000:5000 registry

默认情况下，会将仓库存放于容器内的/tmp/registry 目录下，这样如果容器被删除，则存放于容器中的镜像也会丢失，所以我们一般情况下会指定本地一个目录挂载到容器内的

/tmp/registry 下，如下：

$ sudo docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/registry registry

```
mhy@Linux:/opt/data$ sudo docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/registry registry
613c19bce6d9a7d8a26ff9d43c3922f044c2c6dd0e9a47a796b5d6d9e9b11cfa
mhy@Linux:/opt/data$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
613c19bce6d9        registry:latest     "docker-registry"   5 seconds ago       Up 4 seconds        0.0.0.0:5000->5000/tcp   determined_colden
mhy@Linux:/opt/data$
```

可以看到我们启动了一个容器，地址为：192.168.112.136:5000。

（3）问题扩展：

无。

（4）结合项目中使用：

无。

视频地址：

https://p.bokecc.com/qrplay.bo?uid=78665FEF083498AB&vid=5820F4BA74D9EDA39C33DC5901307461

Flask 框架中请求钩子的作用个，并写出支持的 4 种请求钩子

题目标签：Flask 请求钩子

试题编号：MS002396

（1）问题分析：

无

（2）核心答案讲解：

flask 中具有四种钩子被做成了修饰器,我们在后端可以进行调用做相关的操作.使用钩子函数时,我们需要借助 flask 的全局变量 g.g 作为中间变量,在钩子函数和视图函数中间传递数据.我们先引入全局变量 g

1）before_first_request

2）before_request

3）after_request

4）teardown_request

（3）问题扩展：

无

（4）结合项目中使用：

无

第四部分：数据库

DML 是什么，列举出属于 DML 的命令

题目标签：其他

试题编号：MS001121

ML（data manipulation language）：它们是 SELECT、UPDATE、INSERT、DELETE，就象它的名字一样，这 4 条命令是用来对数据库里的数据进行操作的语言。

数据库优化查询效率的方法

题目标签：其他

试题编号：MS001123

（1）问题分析

（2）核心问题讲解

合理使用索引；

避免或简化排序；

消除对大型表行数据的顺序存取；

避免相关

子查询

避免困难的正规表达式；

使用临时表　加速查询。

数据表有两种含义，一是指数据库最重要的组成部分之一，二是指电子元件，电子芯片等的数据手册（datasheet）。

数据表一般为产品或资料提供一个详细具体的数据资料，方便人们使用和工作时能够清楚方便的获得相应的数据信息。

没有数据表，关键字、主键、索引等也就无从谈起。在数据库画板中可以显示数据库中的所有数据表（即使不是用 PowerBuilder 创建的表），创建数据表，修改表的定义等数据表是数据库中一个非常重要的对象，是其他对象的基础。

（3）问题扩展

（4）结合项目中使用

为什么单线程的 Redis 比多线程的 memcache 的速度快?

题目标签：Redis

试题编号：MS006895

（1）问题分析：

考察数据库的使用情况以及对数据库的理解和在具体使用中的总结。

（2）核心答案讲解：

Redis 是一个开源的内存中的数据结构存储系统，它可以用作：数据库、缓存和消息中间件。

它支持多种类型的数据结构，如字符串（Strings），散列（Hash），列表（List），集合（Set），有序集合（Sorted Set 或者是 ZSet）与范围查询，Bitmaps，Hyperloglogs 和地理空间（Geospatial）索引半径查询。其中常见的数据结构类型有：String、List、Set、Hash、ZSet 这 5 种。

Redis 内置了复制（Replication），LUA 脚本（Lua scripting）， LRU 驱动事件（LRU eviction），事务（Transactions） 和不同级别的磁盘持久化（Persistence），并通过 Redis 哨兵（Sentinel）和自动分区（Cluster）提供高可用性（High Availability）。

Redis 也提供了持久化的选项，这些选项可以让用户将自己的数据保存到磁盘上面进行存储。根据实际情况，可以每隔一定时间将数据集导出到磁盘（快照），或者追加到命令日志中（AOF 只追加文件），他会在执行写命令时，将被执行的写命令复制到硬盘里面。您也可以关闭持久化功能，将 Redis 作为一个高效的网络的缓存数据功能使用。

Redis 不使用表，他的数据库不会预定义或者强制去要求用户对 Redis 存储的不同数据进行关联。

数据库的工作模式按存储方式可分为：硬盘数据库和内存数据库。Redis 将数据储存在内存里面，读写数据的时候都不会受到硬盘 I/O 速度的限制，所以速度极快。

Redis 为什么这么快

1）完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O(1)；

2）数据结构简单，对数据操作也简单，Redis 中的数据结构是专门进行设计的；

3）采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4）使用多路 I/O 复用模型，非阻塞 IO；

5）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；

以上几点都比较好理解，下边我们针对多路 I/O 复用模型进行简单的探讨：

多路 I/O 复用模型

多路 I/O 复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。

（3）问题扩展：

1）单进程多线程模型：MySQL、Memcached、Oracle（Windows 版本）；

2）多进程模型：Oracle（Linux 版本）；

3）Nginx 有两类进程，一类称为 Master 进程(相当于管理进程)，另一类称为 Worker 进程（实际工作进程）。启动方式有两种：

第一种：单进程启动：此时系统中仅有一个进程，该进程既充当 Master 进程的角色，也充当 Worker 进程的角色。

第二种：多进程启动：此时系统有且仅有一个 Master 进程，至少有一个 Worker 进程工作。

Master 进程主要进行一些全局性的初始化工作和管理 Worker 的工作；事件处理是在 Worker 中进行的。

（4）结合项目中使用：

无

Redis 的内部应答处理器和多路复用以及内部实现了队列功能的理解？

题目标签：Redis

试题编号：MS006896

（1）问题分析：

考察 Redis 的使用理解。

（2）核心答案讲解：

Redis 为什么是单线程的：

Redis 核心就是 如果我的数据全都在内存里，我单线程的去操作 就是效率最高的，为什么呢，因为多线程的本质就是 CPU 模拟出来多个线程的情况，这种模拟出来的情况就有一个代价，就是上下文的切换，对于一个内存的系统来说，它没有上下文的切换就是效率最高的。

Redis 用 单个 CPU 绑定一块内存的数据，然后针对这块内存的数据进行多次读写的时候，都是在一个 CPU 上完成的，所以它是单线程处理这个事。在内存的情况下，这个方案就是最佳方案。

什么是 I/O 多路复用：

I/O 多路复用，I/O 就是指的我们网络 I/O,多路指多个 TCP 连接(或多个 Channel)，复用指复用一个或少量线程。串起来理解就是很多个网络 I/O 复用一个或少量的线程来处理这些连接。

为什么 Redis 中要使用 I/O 多路复用这种技术呢?

首先，Redis 是跑在单线程中的，所有的操作都是按照顺序线性执行的，但是由于读写操作等待用户输入或输出都是阻塞的，所以 I/O 操作在一般情况下往往不能直接返回，这会导致某一文件的 I/O 阻塞导致整个进程无法对其它客户提供服务，而 I/O 多路复用就是为了解决这个问题而出现的。

阻塞式的 I/O 模型并不能满足这里的需求，我们需要一种效率更高的 I/O 模型来支撑 Redis 的多个客户（Redis-cli），这里涉及的就是 I/O 多路复用模型了。

Redis 服务采用 Reactor 的方式来实现文件事件处理器（每一个网络连接其实都对应一个文件描述符）

文件事件处理器使用 I/O 多路复用模块同时监听多个 FD，当 accept、read、write 和 close 文件事件产生时，文件事件处理器就会回调 FD 绑定的事件处理器。

虽然整个文件事件处理器是在单线程上运行的，但是通过 I/O 多路复用模块的引入，实现了同时对多个 FD 读写的监控，提高了网络通信模型的性能，同时也可以保证整个 Redis 服务实现的简单。

（3）问题扩展：

Redis 对于 I/O 多路复用模块的设计非常简洁，通过宏保证了 I/O 多路复用模块在不同平台上都有着优异的性能，将不同的 I/O 多路复用函数封装成相同的 API 提供给上层使用。

整个模块使 Redis 能以单进程运行的同时服务成千上万个文件描述符，避免了由于多进程应用的引入导致代码实现复杂度的提升，减少了出错的可能性。

（4）结合项目中使用：

无

MySQL 有哪些高可用构架解决方案，分别是如何做到高可用的?

题目标签：数据库设计

试题编号：MS002023

（1）问题分析

（2）核心问题讲解

MySQL 主从架构：一般初创企业比较常用，也便于后面步步的扩展；

MySQL+DRDB 架构：通过 DRBD 基于 block 块的复制模式，快速进行双主故障切换，

很大程度上解决主库单点故障问题

MySQL+MHA 架构：MHA 目前在 MySQL 高可用方案中应该也是比较成熟和常见的方案，它由日本人开发出来，在 MySQL 故障切换过程中，MHA 能做到快速自动切换操作，而且还能最大限度保持数据的一致性

MySQL+MMM 架构：MMM 即 Master-Master Replication Manager for MySQL（MySQL 主主复制管理器），是关于 MySQL 主主复制配置的监控、故障转移和管理的一套可伸缩的脚本套件（在任何时候只有一个节点可以被写入），这个套件也能基于标准的主从配置的任意数量的从服务器进行读负载均衡，所以你可以用它来在一组居于复制的服务器启动虚拟 ip，除此之外，它还有实现数据备份、节点之间重新同步功能的脚本。

（3）问题扩展

（4）结合项目中使用

什么是 MySQL 的触发器？（请看答案中标红部分，答案应该不全，请补充）

题目标签：MySQL 触发器

试题编号：MS006897

（1）问题分析：

考官主要想考察学员对于数据库的设计和操作，对于数据库操作有没有经验。

（2）核心答案讲解：

触发器（trigger）：监视某种情况，并触发某种操作，它是提供给程序员和数据分析员来保证数据完整性的一种方法，它是与表事件相关的特殊的存储过程，它的执行不是由程序调用，也不是手工启动，而是由事件来触发，例如当对一个表进行操作（ insert，delete，

update）时就会激活它执行。

触发器经常用于加强数据的完整性约束和业务规则等。 触发器创建语法四要素：

1）监视地点(table)

2）监视事件(insert/update/delete)

3）触发时间(after/before)

触发器SQL语法：

```
create trigger triggerName
after/before insert/update/delete on 表名
for each row    #这句话在mysql是固定的
begin
    sql语句;
end;
```

4）触发事件（insert/update/delete）

首先在 Navicat for MySQL 找到需要建立触发器对应的表，右键“设计表”。

（3）问题扩展：

触发器应该尽量避免使用

1）存储过程和触发器二者是有很大的联系的，我的一般理解就是触发器是一个隐藏的存储过程，因为它不需要参数，不需要显示调用，往往在你不知情的情况下已经做了很多操作。从这个角度来说，由于是隐藏的，无形中增加了系统的复杂性，非 DBA 人员理解起来数据库就会有困难，因为它不执行根本感觉不到它的存在。

2）再有，涉及到复杂的逻辑的时候，触发器的嵌套是避免不了的，如果再涉及几个存储过程，再加上事务等等，很容易出现死锁现象，再调试的时候也会经常性的从一个触发器转到另外一个，级联关系的不断追溯，很容易使人头大。其实，从性能上，触发器并没有提

升多少性能，只是从代码上来说，可能在 coding 的时候很容易实现业务。

3）在编码中存储过程显示调用很容易阅读代码，触发器隐式调用容易被忽略。

4）存储过程的致命伤在于移植性，存储过程不能跨库移植，比如事先是在 MySQL 数据库的存储过程，考虑性能要移植到 oracle 上面那么所有的存储过程都需要被重写一遍。

（4）结合项目中使用：

我们在做项目中一般情况下避免使用触发器。

触发器和存储过程本身难以开发和维护，不能高效移植。触发器完全可以用事务替代。存储过程可以用后端脚本替代。

MySQL 的隔离级别具体有哪些?

题目标签：MySQL 隔离

试题编号：MS006900，删除：MS006650

（1）问题分析：

考官主要想考察的是 MySQL 隔离级别的概念，以及具体内容。

（2）核心答案讲解：

事务的基本要素（ACID）

1）原子性（Atomicity）：事务开始后所有操作，要么全部做完，要么全部不做，不可能停滞在中间环节。事务执行过程中出错，会回滚到事务开始前的状态，所有的操作就像没有发生一样。也就是说事务是一个不可分割的整体，就像化学中学过的原子，是物质构成的基本单位。

2）一致性（Consistency）：事务开始前和结束后，数据库的完整性约束没有被破坏 。

比如 A 向 B 转账，不可能 A 扣了钱，B 却没收到。

3）隔离性（Isolation）：同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。比如 A 正在从一张银行卡中取钱，在 A 取钱的过程结束前，B 不能向这张卡转账。

4）持久性（Durability）：事务完成后，事务对数据库的所有更新将被保存到数据库，不能回滚。

事务的并发问题

1）脏读：事务 A 读取了事务 B 更新的数据，然后 B 回滚操作，那么 A 读取到的数据是脏数据

2）不可重复读：事务 A 多次读取同一数据，事务 B 在事务 A 多次读取的过程中，对数据作了更新并提交，导致事务 A 多次读取同一数据时，结果 不一致。

3）幻读：系统管理员 A 将数据库中所有学生的成绩从具体分数改为 ABCDE 等级，但是系统管理员 B 就在这个时候插入了一条具体分数的记录，当系统管理员 A 改结束后发现还有一条记录没有改过来，就好像发生了幻觉一样，这就叫幻读。

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是
串行化（serializable）	否	否	否

（3）问题扩展：

1）事务隔离级别为读提交时，写数据只会锁住相应的行

2）事务隔离级别为可重复读时，如果检索条件有索引（包括主键索引）的时候，默认加锁方式是 next-key 锁；如果检索条件没有索引，更新数据时会锁住整张表。一个间隙被事务加了锁，其他事务不能在这个间隙插入记录，这样可以防止幻读。

3）事务隔离级别为串行化时，读写数据都会锁住整张表

4）隔离级别越高，越能保证数据的完整性和一致性，但对并发性能的影响也越大。

（4）结合项目中使用：

1）读未提交：

A）打开一个客户端 A，并设置当前事务模式为 read uncommitted（未提交读），查询表 account 的初始值：

```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     450 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

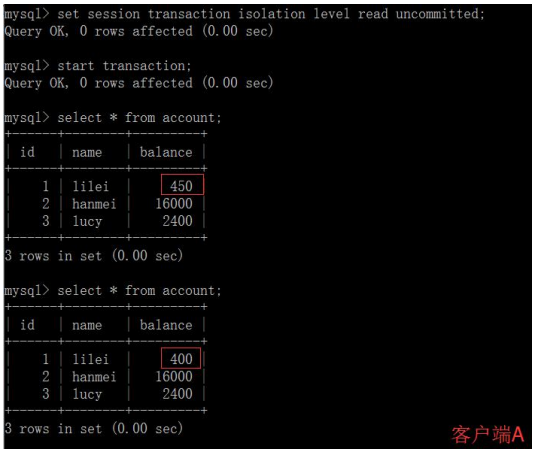
客户端A

B）在客户端 A 的事务提交之前，打开另一个客户端 B，更新表 account：

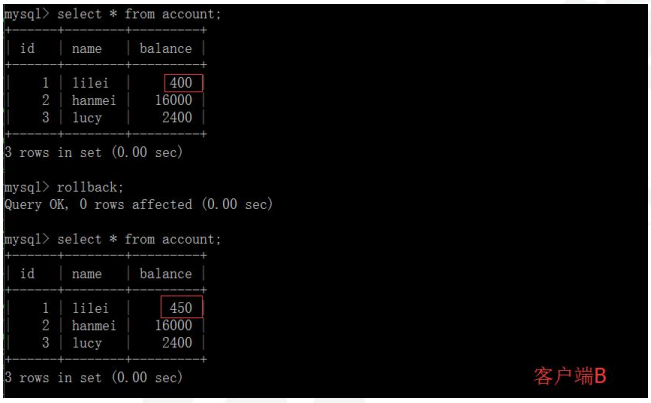
```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance = balance - 50 where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

客户端B

C）这时，虽然客户端 B 的事务还没提交，但是客户端 A 就可以查询到 B 已经更新的数据：



D）一旦客户端 B 的事务因为某种原因回滚，所有的操作都将会被撤销，那客户端 A 查询到的数据其实就是脏数据：



E）在客户端 A 执行更新语句 update account set balance = balance - 50 where id =1，lilei 的 balance 没有变成 350，居然是 400，是不是很奇怪，数据不一致啊，如果你这么想就太天真 了，在应用程序中，我们会用 400-50=350，并不知道其他会话回滚了，要想解决这个问题可以采用读已提交的隔离级别

```
mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)

mysql> update account set balance = balance - 50 where id =1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

客户端A

2）读已提交

A）打开一个客户端 A，并设置当前事务模式为 read committed（未提交读），查询表 account 的所有记录：

```
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     450 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

客户端A

B）在客户端 A 的事务提交之前，打开另一个客户端 B，更新表 account：

```
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance = balance - 50 where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

客户端B

C）这时，客户端 B 的事务还没提交，客户端 A 不能查询到 B 已经更新的数据，解决了脏读问题：

```
mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     450 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     450 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

客户端A

D）客户端 B 的事务提交

```
mysql> update account set balance = balance - 50 where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

客户端B

E）客户端 A 执行与上一步相同的查询，结果 与上一步不一致，即产生了不可重复读的问题

```
mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     450 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

不可重复读

客户端A

3）可重复读

A）打开一个客户端 A，并设置当前事务模式为 repeatable read，查询表 account 的所有记录

```
mysql> set session transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

客户端A

B）在客户端 A 的事务提交之前，打开另一个客户端 B，更新表 account 并提交

```
mysql> set session transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance = balance - 50 where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     350 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

客户端B

C）在客户端 A 查询表 account 的所有记录，与步骤（1）查询结果一致，没有出现不可重复读的问题。

MySQL 数据库索引的实现，索引的优缺点，索引的原理?

题目标签：MySQL 索引

试题编号：MS006901，删除：MS006667

（1）问题分析：

无

（2）核心答案讲解：

索引的优缺点

优点

1）大大加快数据的检索速度;

2）创建唯一性索引，保证数据库表中每一行数据的唯一性;

3）加速表和表之间的连接;

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

缺点

1）索引需要占物理空间。

2）当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度。

索引的实现原理

MySQL 支持诸多存储引擎，而各种存储引擎对索引的支持也各不相同，因此 MySQL 数据库支持多种索引类型，如 BTree 索引，B+Tree 索引，哈希索引，全文索引等等，

1）哈希索引:

只有 memory（内存）存储引擎支持哈希索引，哈希索引用索引列的值计算该值的 hashCode，然后在 hashCode 相应的位置存执该值所在行数据的物理位置，因为使用散列算法，因此访问速度非常快，但是一个值只能对应一个 hashCode，而且是散列的分布方式，因此哈希索引不支持范围查找和排序的功能。

2）全文索引：

FULLTEXT（全文）索引，仅可用于 MyISAM 和 InnoDB，针对较大的数据，生成全文索引非常的消耗时间和空间。对于文本的大对象，或者较大的 CHAR 类型的数据，如果使用普通索引，那么匹配文本前几个字符还是可行的，但是想要匹配文本中间的几个单词，那么就要使用 LIKE %word%来匹配，这样需要很长的时间来处理，响应时间会大大增加，这种情况，就可使用时 FULLTEXT 索引了，在生成 FULLTEXT 索引时，会为文本生成一份单词的清单，在索引时及根据这个单词的清单来索引。

（3）问题扩展：

无

（4）结合项目中使用：

无

MySQL 数据库如何分区、分表?

题目标签：数据库

试题编号：MS006902，删除：MS006669

（1）问题分析：

考官主要想考察学员对 MySQL 数据的理解和拓展，有没有深入的去理解过数据库，有

没有在项目开发中对数据库做过优化。

（2）核心答案讲解：

分区

就是把一张表的数据分成 N 个区块，在逻辑上看最终只是一张表，但底层是由 N 个物理区块组成的

分表

就是把一张表按一定的规则分解成 N 个具有独立存储空间的实体表。系统读写时需要根据定义好的规则得到对应的字表明，然后操作它

分表可以通过三种方式：MySQL 集群、自定义规则（根据一定的算法（如用 hash 的方式，也可以用求余（取模）的方式）让用户访问不同的表。）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。

（3）问题扩展：

什么时候考虑使用分区？

一张表的查询速度已经慢到影响使用的时候

sql 经过优化

数据量大

表中的数据是分段的

对数据的操作往往只涉及一部分数据，而不是所有的数据

分区解决的问题：

主要可以提升查询效率

什么时候考虑分表

一张表的查询速度已经慢到影响使用的时候

sql 经过优化

数据量大

当频繁插入或者联合查询时，速度变慢

分表解决的问题：

分表后，单表的并发能力提高了，磁盘 I/O 性能也提高了，写操作效率提高了

查询一次的时间短了

数据分布在不同的文件，磁盘 I/O 性能提高

读写锁影响的数据量变小

插入数据库需要重新建立索引的数据减少

（4）结合项目中使用：

说说你项目中如何对 MySQL 进行优化的，举具体的例子?

或者给你几张表，你设计一个优化策略。

数据库怎么优化查询效率?

题目标签：数据库优化

试题编号：MS006672，删除：MS006903

（1）问题分析

考官主要想考察学员对 MySQL 数据的理解和拓展，有没有深入的去理解过数据库，有没有在项目开发中对数据库做过优化。

（2）核心问题讲解

数据库设计方面：

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）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,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。

4）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。

5）应尽可能的避免更新索引数据列，因为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索引数据列，那么需要考虑是否应将该索引建为索引。

6）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。

7）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。

8）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）。

9）避免频繁创建和删除临时表，以减少系统表资源的消耗。

10）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。

11）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。

12）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。

SQL 语句方面：

1）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。

2）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

3）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

4）下面的查询也将导致全表扫描：

select id from t where name like '%abc%'

5）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

6）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

7）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

select id from t where datediff(day,createdate,’2005-11-30’)=0

–‘2005-11-30’生成的 id

应改为:

select id from t where name like ‘abc%’ select id from t where createdate>=’2005-11-30’ and createdate<’2005-12-1’

8）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。

9）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(...)

10）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)

11）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。

12）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。

13）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。

14）尽量避免大事务操作，提高系统并发能力。

（3）问题扩展

（4）结合项目中使用

结合项目中具体事例说说如何对查询效率做的优化

数据库的优化?

题目标签：数据库优化

试题编号：MS006680

（1）问题分析

考虑数据库的使用，在面对大量数据时如何保证数据库中的数据及时的更新、响应用户的请求、确保数据库的在高并发情况下仍能保证数据的完整性等

（2）核心问题讲解

1）选取最适用的字段属性

MySQL 可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上

面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

2）使用连接（JOIN）来代替子查询(Sub-Queries)

这个技术可以使用 SELECT 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。例如，我们要将客户基本信息表中没有任何订单的客户删除掉，就可以利用子查询先从销售信息表中将所有发出订单的客户 ID 取出来，然后将结果传递给主查询；使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的 SQL 操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接（JOIN）..替代。

3）使用联合(UNION)来代替手动创建的临时表

4）事务

5）锁定表

6）使用外键

7）使用索引

8）对查询语句进行优化

（3）问题扩展

都有哪些熟悉的数据库，常用的有哪些，每种数据库对应的优缺点等

（4）结合项目中使用

针对每种数据的优缺点来结合项目进行说明，在什么情况下使用哪种数据库，使用的原因等（比如：采用 Redis 数据库来对数据进行缓存，查询速度相对于其他的数据库来说比较快等）

MyISAM 与 InnoDB 的区别?

题目标签：MySQL 引擎

试题编号：MS006681

（1）问题分析

考察是否了解数据库引擎

（2）核心问题讲解

1）MyISAM：默认表类型，它是基于传统的 ISAM 类型，ISAM 是 Indexed Sequential Access Method (有索引的顺序访问方法) 的缩写，它是存储记录和文件的标准方法。不是事务安全的，而且不支持外键，如果执行大量的 select，insert MyISAM 比较适合。

2）InnoDB：支持事务安全的引擎，支持外键、行锁、事务是他的最大特点。如果有大量的 update 和 insert，建议使用 InnoDB，特别是针对多个并发和 QPS 较高的情况。

（3）问题扩展

（4）结合项目中使用

数据库分页以及参数

题目标签：数据库

试题编号：MS001464

（1）问题分析

（2）核心问题讲解

MySQL 分页需用到的参数:

pageSize 每页显示多少条数据

pageNumber 页数 从客户端传来

totalRecouds 表中的总记录数 select count (*) from 表名

totalPages 总页数

totalPages=totalRecouds%pageSize==0?totalRecouds/pageSize:totalRecouds/pageSize+1

pages 起始位置

pages= pageSize*(pageNumber-1)

SQL 语句:

select * from 表名 limit pages, pageSize;

MySQL 分页依赖于关键字 limit 它需两个参数:起始位置和 pageSize

起始位置=页大小*(页数-1)

起始位置=pageSize*(pageNumber -1)

（3）问题扩展

（4）结合项目中使用

Redis 并发竞争

题目标签：数据库

试题编号：MS001471

（1）问题分析：

（2）核心答案讲解：

Redis 的并发竞争问题，主要是发生在并发写竞争。

考虑到 Redis 没有像 db 中的 sql 语句，update val = val + 10 where ...，无法使用这种方式进行对数据的更新。

假如有某个 key = "price"， value 值为 10，现在想把 value 值进行+10 操作。正常逻辑下，就是先把数据 key 为 price 的值读回来，加上 10，再把值给设置回去。如果只有一个连接的情况下，这种方式没有问题，可以工作得很好，但如果有两个连接时，两个连接同时想对还 price 进行+10 操作，就可能会出现问题了。

例如：两个连接同时对 price 进行写操作，同时加 10，最终结果我们知道，应该为 30 才是正确。

考虑到一种情况：

T1 时刻，连接 1 将 price 读出，目标设置的数据为 10+10 = 20。

T2 时刻，连接 2 也将数据读出，也是为 10，目标设置为 20。

T3 时刻，连接 1 将 price 设置为 20。

T4 时刻，连接 2 也将 price 设置为 20，则最终结果是一个错误值 20。

解决方案

方案 1

利用 Redis 自带的 incr 命令，具体用法看这里 http://doc.Redisfans.com/string/incr.html。

方案 2

可以使用独占锁的方式，类似操作系统的 mutex 机制。（网上有例子，

http://blog.csdn.net/black_ox/article/details/48972085 不过实现相对复杂，成本较高）

方案 3

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。伪代码如下：

watch price

get price $price

$price = $price + 10

multi

set price $price

exec

方案 4

这个是针对客户端来的，在代码里要对 Redis 操作的时候，针对同一 key 的资源，就先进行加锁（java 里的 synchronized 或 lock）。

方案 5

利用 Redis 的 setnx 实现内置的锁。

（3）问题扩展：

（4）结合项目中使用：

MySQL，Redis 集群部署

题目标签：MySQL，Redis 集群部署

试题编号：MS006600 删除：MS006898

（1）问题分析

1）为什么要做 MySQL Redis 集群部署， 集群部署的优点

2）集群的缺点，考察你对数据库集群是否做过，是否理解的比较深

3）Redis 集群的类型

4）考官主要考察搭建数据库集群并且部署，如 MySQL 和 Redis 集群的具体搭建并且如何部署到服务器上

（2）核心问题讲解

MySQL 集群优点：

1）99.999%的高可用性

2）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

MySQL 集群缺点：

1）存在很多限制，比如：不支持外键

2）部署、管理、配置很复杂

3）占用磁盘空间大、内存大

4）备份和恢复不方便

5）重启的时候，数据节点将数据 load 到内存需要很长的时间

MySQL 集群方案

1）Repliaction 集群方案

特性：速度快、弱一致性、低价值、日志、新闻、帖子

2）PXC 集群方案（ Percona XtraDB Cluster

特性：速度慢、强一致性、高价值、订单、账户、财务

PXC 方案和 Replication 方案对比

1）先看看 PXC 方案

很明显 PXC 方案在任何一个节点写入的数据都会同步到其他节点，数据双向同步的（在任何节点上都可以同时读写）。

2）Replication 集群方案：

Replication 方案只能在 Master 数据库进行写操作，在 Slave 数据库进行读操作。如果在 Slave 数据库中写入数据，Master 数据库是不能知道的（单向同步的）。

3）PXC 数据的强一致性

PXC 采用同步复制，事务在所有集群节点要么同时提交，要么不提交。

Replication 采用异步复制，无法保证数据的一致性。

1）下面看看 PXC 写入操作

当一个写入请求到达 PXC 集群中的一个 MySQL（node1 数据库） 数据库时，node1 数据库会将该写入请求同步给集群中的其他所有数据库，等待所有数据库都成功提交事务后，node1 节点才会将写入成功的结果告诉给 node1 的客户端。

PXC 的强一致性对保存高价值数据时特别重要。

2）在看 Replication 集群写入操作：

当一个写入请求到达 Master 数据库时，Master 数据库执行写入操作，然后 Master 向客户端返回写入成功，同时异步的复制写入操作给 Slave 数据库，如果异步复制时出现问题，从数据库将无法执行写入操作，而客户端得到的是写入成功。这也是弱一致性的体现。

Redis 集群的类型：

主从复制

1）哨兵模式

2）Redis-Cluster 集群

3）Redis 集群的优点：

A）将数据自动切分（split）到多个节点的能力。

B）当集群中的一部分节点失效或者无法进行通讯时， 仍然可以继续处理命令请求的能力。

4）Redis 集群搭建

Redis 集群是一个可以在多个 Redis 节点之间进行数据共享的设施（installation）。

Redis 集群不支持那些需要同时处理多个键的 Redis 命令， 因为执行这些命令需要在多个 Redis 节点之间移动数据， 并且在高负载的情况下， 这些命令将降低 Redis 集群的性能， 并导致不可预测的行为。

R edis 集群通过分区（partition）来提供一定程度的可用性（availability）： 即使集群中有一部分节点失效或者无法进行通讯， 集群也可以继续处理命令请求。

Reds 集群的特性

Redis 集群数据共享

Redis 集群中的主从复制

Redis 集群的数据一致性

MySQL 集群部署方案

1）启动 MySQL 主从备份

2）通过使用 Mycat 中间件做分表以及路由

3）使用 haproxy 代理 MyCat 做负载均衡

4）keepalived 保证 haproxy 的高可用性，解决单点故障。

所需依赖：

1）pip install Redis-py-cluster

2）sudo gem install Redis

3）rubygems：sudo apt-get install rubygems

4）ruby-1.8.7：sudo apt-get install ruby

5）Redis.io/download">Redis-3.0.7

修改配置文件 Redis.conf

1）这里创建 3 个节点，所以复制 3 份 Redis.conf

2）分别命名为：Redis-6379.conf, Redis-6380.conf, Redis-6381.conf

3）分别修改其中如下地方，拿 Redis-6379.conf 来举例

开启 3 个 Redis-server 节点

1）./Redis-server ./Redis-6379.conf

2）./Redis-server ./Redis-6380.conf

3）./Redis-server ./Redis-6381.conf

创建 Redis 的集群

1）./Redis-trib.rb create 127.0.0.1:6379 127.0.0.1:6380 127.0.0.1:6381

2）Redis-py-cluster 测试客户端： 官网：

https://github.com/Grokzen/Redis-py-cluster

（3）问题扩展

如何应对缓存穿透和缓存雪崩问题，部署过程中有没有出现问题？

（4）结合项目中使用

无

第五部分：人工智能

用 Python 实现冒泡排序

题目标签：其他

试题编号：MS001157

（1）问题分析

（2）核心问题讲解

```
for(i = 0; i n-1; i++){
  for(j = 0; j n-1-i; j++)
      if(a[j] a[j+1]){
        temp = a[j];
        a[j] = a[j+1];
```

```
        a[j+1] = temp;
      }
  }
```

（3）问题扩展

（4）结合项目中使用

编写三种熟知的排序算法

题目标签：算法实现/elasticsearch

试题编号：MS002079

（1）问题分析：

无

（2）核心答案讲解：

冒泡排序:

```
def bubble_sort(lists):
    # 冒泡排序
    count = len(lists)
    for i in range(0, count):
        for j in range(i + 1, count):
            if lists[i] > lists[j]:
                #判断后值是否比前置大，如果大就将其交换
```

```
                lists[i], lists[j] = lists[j], lists[i]
    return lists


res=bubble_sort([1,209,31,4,555,6,765,9,5,4,7,89,6,5,34,3,57,96])print(res)
```

快速排序:

```
def quick_sort(lists, left, right):
    # 快速排序
    if left >= right:
        return lists
    key = lists[left]
    low = left
    high = right
    while left < right:
        while left < right and lists[right] >= key:
            right -= 1
        lists[left] = lists[right]
        while left < right and lists[left] <= key:
            left += 1
        lists[right] = lists[left]
    lists[right] = key
    quick_sort(lists, low, left - 1)
```

```
    quick_sort(lists, left + 1, high)
    return lists
```

选择排序:

```
def select_sort(lists):
    # 选择排序
    count = len(lists)
    for i in range(0, count):
        min = i
        for j in range(i + 1, count):
            if lists[min] > lists[j]:
                min = j
        print("--------",lists[min], lists[i])
        lists[min], lists[i] = lists[i], lists[min]
        print(lists[min], lists[i])
    return lists
res=select_sort([1,209,31,4,555,6,765,9,5,4,7,89,6,5,34,3,57,96])print(res)
```

(3) 问题扩展:

无

(4) 结合项目中使用:

无

输入字符串（以都好隔开的数字组成，如“15，20，39，4，60，90”），要求将字符串中数字采用冒泡排序法进行排序并输出结果，然后将每个数与最小值的差值小于 10 的数字的个数统计出来。

题目标签：算法实现/elasticsearch

试题编号：MS002367

（1）问题分析：

无

（2）核心答案讲解：

```
alist = [15,20,39,4,60,90]
#函数冒泡排序
#    参数 alist：被排序的列表
def bubbleSort(alist):
    for num in range(len(alist)-1,0,-1):
        for i in range(num):
            if alist[i] < alist[i+1]:
                #进行当前位置和下一个位置的交换
                alist[i] = alist[i]^alist[i+1]
                alist[i+1] = alist[i]^alist[i+1]
                alist[i] = alist[i]^alist[i+1]
    return alist
```

(3) 问题扩展：

无

(4) 结合项目中使用：

无

一线名企面试笔试真题

面试问题（包含笔试题）	问题分类	面试题考点（技术点）
实现一个二分查找算法	数据结构与算法	二分算法
对线程、进程、协程是怎么理解的	系统编程	线程、进程、协程
怎么在多进程里面做资源共享	系统编程	队列
socket 编程会写吗（socket 流程）	网络编程	socket
tcp3 次握手流程图画一下	网络编程	tcp 三次握手
多对多关系的表怎么建立他们之间的联系	数据库	多表关联
菜单表与菜单表之间是什么联系	数据库	自关联
查询的时候怎么查	数据库	select 语句
redis 的类型有哪些	数据库	redis
在操作 redis 的时候并发的时候自增、自减用哪个命令	数据库	redis
列出你 Linux 常用的哪些命令	Linux 系统使用	Linux 命令
Linux 查内存怎么查，用哪个命令查	Linux 系统使用	Linux 命令
查系统的负载呢	Linux 系统使用	Linux 命令
查日志用什么命令	Linux 系统使用	Linux 命令
celery 怎么控制它的并发率	工具组件	celery
nginx 和 uwsig 的区别	服务器	nginx、uwsig

git 用得熟吗	Git	git
你们项目是怎么部署的（部署流程）	业务实现	部署
前后端分离是怎么做的？（流程）	业务实现	前后端分离
讲一下 restful 风格有哪些	Web 基础	restful
前端怎么知道访问的接口有问题呢（前端怎么知道出错）	Web 基础	前后端分离
mysql 熟悉吗？你对这个表设计有了解过一些吗	数据库	表设计
有了解 mysql 的索引吗	数据库	索引
有用过消息队列吗	工具组件	消息队列
docker 有用过吗？你们项目有部署在 docker 里面吗	工具组件	docker
elasticha 是用来干嘛的？数据量有多少？	工具组件	es
用的是什么支付？有用过微信支付吗	业务实现	第三方支付
有没有用 GBDT 筛选过特征	数据挖掘与分析	GBDT 筛选
模型的调优你都用什么做指标	数据挖掘与分析	数据模型
flask 与 Django 框架的区别，项目使用 flask 或 tornado 框架，搭建微服务		框架
haystack 和 elasticsearch 对接遇到过什么问题		工具组件
什么是 http，有哪些状态码？比如说用户访问没有权限会返回什么状态码	Web 基础	http
切片	Python 语言	数据

生成器和迭代器	Python 语言	装饰器迭代器
python 的数据类型和 redis 的数据类型	数据库	数据类型
python 有哪些可变对象和不可变对象	Python 语言	数据对象
git	git	git
状态保持	Web 基础	状态保持
[i%2 for I in range(10)]和 (I % 2 for in range(10))输出结果分别是什么?	Python 语言	函数
python2 和 python3 有哪些显著的区别?	Python 语言	lambda 表达式
请描述 unicode、utf-8、gbk 等编码之间的关系?	Python 语言	正则表达式
请描述 with 的用法? 如果自己的类需要支持 with 语句, 应该如何书写?	Python 语言	运算符
什么是装饰器? 写一个装饰器, 可以打印输出方法执行时长的信息	Python 语言	运算符
如何暂停一个正在运行的程序, 把其放在后台 (不运行)	Linux 系统使用	linux 暂停程序
用一条命令查目前系统已启动的服务所监听的端口	运维	linux 监听端口
查找多有名称中包含“Test”的进程, 并且全部强制终止进程	Linux 系统使用	linux 查找进程终止进程
用 sed 修改 test.txt 第 23test 伟 TEST。	Linux 系统使用	linux 系统使用
枚举 git 克隆仓库、查看相关信息、获取最远端分支, 获取特定分支代码及添加 tag 的命令	Git	GIT 操作
两张关联表 a 和 b, 关联字段伟 id, 删除主表 a 中在副表	数据库	数据库关联查询

b 中没有的信息		
pyhton 如何实现单例模式保证线程安全	设计模式	设计模式
如何调用远程接口	业务实现	远程接口调用
如何优化 python 代码	业务实现	python 代码优化
简要描述 pyhton 的垃圾回收机制	Python 语言	垃圾回收
什么是 lambda 函数，有什么好处	Python 语言	匿名函数
单引号、双引号、三引号的却别，分别阐述 3 中印好的场景	Python 语言	python 基础
python 和多线程是一个好主意么，列举一些让代码并行运行的方法	Python 语言	多线程
这两个参数是什么意思：*args **kwargs？我们为什么要使用它们？	Python 语言	函数参数
下面这些事什么意思：@classmethod @staticmethod @property？	Python 语言	面向对象
匿名函数的排序	Python 语言	函数
py3 和 py2 的区别	Python 语言	版本的区别
链表的冒泡排序	数据结构与算法	数据结构
快排的原理？	数据结构与算法	并发量
设计模式你知道几种？用过哪些？	设计模式	设计模式
写一个函数,有一个整数参数 n 返回反转数字中的每一位数字后的整数(不使用内置函数)	数据结构与算法	数据结构

一个有整数构成的数组,找出数组中的第三大的数字,(不可用 sort(),注意性能)	数据结构与算法	排序算法
用你擅长的语言将一个文本中出现次数最多的单词找出来,文本中的单词以空格分隔	Python 语言	运算符
实现模拟 http 请求发送至"www.baidu.com",并判断返回码是否为 200	Web 基础	网络请求
用 python 实现生产者消费者模型	设计模式	设计模式
编写一个开平方函数,并输入一个为 double 类型的非负数,输出为输入值的开方的结果,要求保留小数点后 6 位	Python 语言	函数
常用的 I/O 多路复用技术有哪些?请阐明他们各自的特点和优点	Web 基础	多路复用
在 python 标准库中,在多个进程间共享数据的方法有哪些?他们是进程安全的么?	系统编程	进程
解释以下魔法方法的作用,以及它们在什么情况下被触发	Python 语言	魔法方法
python 语言实现冒泡排序	数据结构与算法	排序算法
使用伪代码描述快速排序的原理	数据结构与算法	算法
编写一个函数来查找字符串数组中最长的公共前缀子串，如果没有公共前缀，则返回空字符串""	Python 语言	字符串操作
编写一个函数来检查给定的链表是否包含有环？如果有则返回环的起始节点，否则返回 null?	Python 语言	数据结构
写出一个函数来打印出第 n 个素数	Python 语言	Python 基础
写一个函数来检查一个字符序列是否是回文	Python 语言	字符串操作

如何将字符串的第一个字母变成大写？	Python 语言	字符串操作
请设计一段程序，可以将一篇纯英文的文章中，使用的单词按出现频次有高到低排列，每个单词后应标明这个词出现的次数	Python 语言	算法
哈希冲突回避算法有哪几种，分别有什么特点	数据结构与算法	数据结构和算法
分别用装饰器类的共享属性实现单例模式	设计模式	设计模式
实现一个 data descriptor 需要的方法	Python 语言	描述器的使用
描述 MySQL 中 left join 和 inner join 的区别	数据库	数据库的连接查询
请写出你知道的 MySQL 存储引擎和区别	数据库	数据库的引擎
举例说明什么样的查询会导致互数据库死锁？	数据库	数据库
如何在 Python 中实现多线程？	网络编程	多线程的使用
请描述协程和线程的区别，以及各自的应用场景	网络编程	协程与线程的区别
请描述深拷贝和浅拷贝的区别，可以用代码演示	Python 语言	深浅拷贝
从加入购物车到购买到出货，一个流程是什么？	业务实现	订单出货流程
订单付款成功之后呢？干嘛去了，出货？	业务实现	后台管理
秒杀怎么设计的呢？	业务实现	秒杀
有用到 redis 吗？	数据库	redis
短信验证码怎么做？	业务实现	短信验证码

过期时间怎么设置？	数据库	redis
redis 除了做缓存，还做过什么的吗	业务实现	redis 消息订阅
mysql 用的熟悉吗？使用的 orm 吗？	数据库	mysql
平时有建表吗	数据库	mysql 建表
索引和 key 的区别？	数据库	索引
引擎用的哪个？	数据库	引擎
为什么要用 innodb？支持事务吗？	数据库	引擎
前后端分离怎么做的？有没有碰到什么问题？	系统架构	前后端分离
有用过 socket 编程吗？	网络编程	socket
密码有什么加密的？验证呢？	业务实现	登录注册
权限校验，有什么样的权限？用户分几种？这个怎么做？	业务实现	权限校验
如果用户直接访问那些接口的话？怎么去鉴权？	业务实现	权限校验
有做过单点登录吗？	业务实现	单点登录
有用过 supervisor？	工具组件	supervisor
linux 用的是什么，常用的是什么命令？	Linux 系统使用	基本命令
查看进程？实时查看文件？配置文件在哪？	Linux 系统使用	基本命令
捕获异常执行顺序	Python 语言	捕获异常
日志怎么写的？怎么拿到更多信息？日志的级别有哪些？	Python 语言	日志
平时怎么看堆栈信息？只有一行的话，怎么定位错误？没有存日志文件吗？	Python 语言	定位 bug
平时怎么去调试代码？	Python 语言	定位 bug
rabbitMQ 的概念，队列中信道 交换器 路由器 队列的关系	难	web 框架

多线程和多进程的使用场景，即 io 密集型一般用多线程，cpu 密集型一般用多进程	简单	系统编程
元类了解吗？说一下。		
QL 语句 只写 join 的话默认是使用哪种连接（左右内）？内		
如果锁表的情况通常是跟什么有关？索引		
什么叫设计模式？之前用过什么设计模式	设计模式	考察设计模式掌握
HTML5 的新内容标签有哪些	前端	HTML5
Doctype 的作用是什么	前端	HTML5
行内元素有哪些？块级元素有哪些？	前端	HTML5
请描述一下 cookies，sessionStorage 和 localStorage 的区别？	前端	HTML5
iframe 有哪些缺点？	前端	HTML5
如何实现浏览器内多个标签页之间的通信？	前端	HTML5
js 的基本数据类型有哪些？	前端	js
DOM 操作—怎样添加、移除、移动、复制、创建和查找节点？	前端	js
请描述一下你认识的 JSON	网络编程	网络
如何解决跨域的问题？（至少三种）	WEB 框架	网络
常见的 http 状态码有哪些？分别代表是什么意思？	网络编程	网络
介绍一下 CSS 的盒子模型？	前端	CSS
CSS3 新增伪类有哪些？	前端	CSS
CSS3 有哪些新特性？	前端	CSS
CSS 优先级算法如何计算？	前端	CSS
CSS 选择器有哪些？哪些属性可以继承？	前端	CSS
写一个函数，有一个整数参数 n 返回反转数字中的每一位数字后的整数(不使用内置函数)	数据结构与算法	数据结构

一个有整数构成的数组,找出数组中的第三大的数字,(不可用 sort(),注意性能)	数据结构与算法	排序算法
用你擅长的语言将一个文本中出现次数最多的单词找出来,文本中的单词以空格分隔	Python 语言	运算符
实现模拟 http 请求发送至"www.baidu.com",并判断返回码是否为 200	Web 基础	网络请求
用 python 实现生产者消费者模型	设计模式	设计模式
编写一个开平方函数,并输入一个为 double 类型的非负数,输出为输入值的开方的结果,要求保留小数点后 6 位	Python 语言	函数
常用的 I/O 多路复用技术有哪些?请阐明他们各自的特点和优点	Web 基础	多路复用
在 python 标准库中,在多个进程间共享数据的方法有哪些?他们是进程安全的么?	Python 语言	进程
解释以下魔法方法的作用,以及它们在什么情况下被触发	Python 语言	魔法方法
python 语言实现冒泡排序	数据结构与算法	排序算法
如何写 python 单元测试,请给出具体的例子	Python 语言	单元测试
python 中如何产生随机数	Python 语言	随机函数
在 linux 中如何查找文件名为 newtown.py 的文件	Linux 系统使用	linux 命令
简述 linux 环境变量的用途	Linux 系统使用	linux 环境变量
linux 中如何查询当前主机的内存使用情况	Linux 系统使用	linux 命令
_new_和 _init_的区别	Python 语言	魔法方法
使用 fifter 和 lambda 过滤 a 中大于 5 的数字 a=[1,2,3,4,5,6]	Python 语言	lambda 过滤操作
[x for x in range(10) if	Python 语言	列表生成式

x % 2 == 0]输出什么		
（x for x in range(10) if x % 2 == 0）输出什么	Python 语言	生成器
django models 的 Q（）和 F（）作用是什么	WEB 框架	django
说下装饰器的了解	Python 语言	装饰器
django 中间键的请求过程	WEB 框架	中间件
单例模式的应用	设计模式	单例模式
快排和冒泡算法实现	数据结构与算法	算法
项目中用到的模块	业务实现	导入模块
django 的底层原理	WEB 框架	原理
百度请求的全过程	网络编程	请求过程
垃圾回收机制的了解	Python 语言	垃圾回收机制
数据库的使用	数据库	数据库操作
高并发是怎么解决的	业务实现	高并发的问题
简单描述一下 TCP 和 UDP 的区别以及优缺点。	网络编程	TCP 和 UDP
函数装饰器有什么作用请列举出至少三个并举出一些实例？	Python 语言	装饰器
简单说一下浏览器通过简答浏 WSGI 请求动态资源的过程？	服务器	WSGI
简单描述一下浏览器访问 www.baidu.com 的过程？	网络编程	tcp
RESTful 开发 API 接口的规范？	WEB 框架	RESTful 的掌握
你们这个项目的集群与节点的结构，几个节点，分布式结构，都开了哪些进程服务？	服务器	集群
为什么使用 hash 存储购物车信息（包括浏览记录为啥用 list 存储,只要涉及到 redis 数据结构存储，可能会问为什么使用这种结构存储）	业务实现	redis 数据类型的用法
如何限制用户的访问量？	业务实现	并发限制

docker 配置 fdfs 是接口么比如多个服务器都要上传图像到 fdfs 怎么实现	工具组件	docker
订单支付如何实现同时操作成功，或者同时操作失败	业务实现	事物
django 的有外键的数据库查询	WEB 框架	Django 数据库
你是怎么使用 socket 的？	网络编程	socket
视图类有哪些类？	WEB 框架	view
如何查看 Docker 中的程序？	工具组件	docker
赋值，浅拷贝，深拷贝的区别？	Python 语言	深拷贝浅拷贝
聊一聊 celery	工具组件	celery
聊一聊消息队列	设计模式	消息队列
Redis 的缺点	数据库	Redis
Redis 在项目中哪些地方有用到	数据库	Redis
聊聊 ES	工具组件	ES
什么字段适合索引	数据库	MySQL
MySQL 引擎	数据库	MySQL
Django 和 Flask 框架的区别	WEB 框架	web 框架比较
mysql 主从数据如何同步的	数据库	MySQL
主从集群的作用	数据库	MySQL
数据存储量的扩大怎么回事	数据库	MySQL
集群是怎么从一个机器上查到其它集群里面的数据	数据库	MySQL
数据库索引有了解吗，	数据库	MySQL
B 树和 b+树的区别	数据库	MySQL
线程和协程的区别	Python 语言	线程
websocket 具体怎么实现的	Web 基础	websocket
升级成 websocket 后与 http 请求的区别	Web 基础	websocket
二分查找的思路，有重复的情况怎么办，最坏的情况是什么	数据结构与算法	二分查找
快速排序的思路，时间复杂度	数据结构与算法	快排
进程间通信有哪些方式	Python 语言	进程
ElasticSearch 的应用场景	工具组件	HTTP
mysql 也有 like 的模糊查询，	工具组件	TCP/IP

为什么要用 ElasticSearch		
微信公众平台怎么实现的	业务实现	Mysql 性能
restful 设计风格	Web 基础	restful 风格
如何高效往序列中插入数据		python 数据
部署的具体步骤		运维
数据结构		python
进程中栈和堆的区别	系统编程	考察系编程的掌握
算法的时间复杂度和空间复杂度是什么？	数据结构与算法	考察算法的掌握
悲观锁和乐观锁的区别，使用时如何选择？	WEB 框架	考察 WEB 基知识的掌握
使用 Python 将字符串"1,2,3,4,5"转换为字符串“5|4|3|2|1”	python 语言	py 方法
请分别描述 Python2.X 和 Python3.X 中 import 包时的路径额搜索顺序	python 语言	python 版本
用 Python 的正则表达式匹配时间信息	设计模式	正则
使用 Python 编写一个装饰器，打印被装饰函数的输入与输出	python 语言	装饰器
阐述 range 和 xrange 的区别，并且用 Python 仿写 xrange 函数	python 语言	列表推导式
列举几种你曾经常用的 Python 包并且解释其功能以及用法	python 语言	常用包
HTML5 的新的内容标签	前端	前端
Doctype 的作用是什么	前端	标签
行内元素有哪些？块级元素有哪些：空元素有哪些？	前端	前端标签
请描述一下 cookies,sessionStorage 和 localStorage 的区别？	python 语言	cookie。session
iframe 有哪些缺点？	项目业务	iframe
如何实现浏览器内多个标签页之间的通信？	项目业务	浏览器通信
分别从前端、后端、数据库阐述 web 项目的性能优化	项目业务	性能优化

简述同源策略	项目业务	同源策略
list 如何倒序切片	python 语言	python list 切片
什么是列表推导式	python 语言	列表推导式
深拷贝和浅拷贝	python 语言	深拷贝和浅拷贝
面向对象的特征	python 语言	面向对象的三大特征
如何理解多态	web 基础	对多态的理解
python 中 GIL 的作用	python 语言	GIL 锁
如何理解 python 元类	python 语言	对 python 元类的理解
线程安全	python 语言	线程安全
python 内存管理机制	python 语言	内存管理
python2 和 python3 字符编码方式的区别	python 语言	python2 和 3 的编码区别
如何利用 SciKit 包训练一个简单的线性回归模型	数据挖掘与分析	第三方包 scikit
例举几个常用的 python 分析数据包及其作用	数据挖掘与分析	常用数据分析包
如何利用 Numpy 对数列的前 n 项进行排序	数据挖掘与分析	numpy 的使用
如何检验一个数据集或者时间序列是随机分布的	数据挖掘与分析	数据集
Pandas 中使用的标准数据缺失标志是什么	数据挖掘与分析	pandas 库
python 导包关键字	Python 语言	导包
import 与 form import 的区别	Python 语言	导包
python 的类是怎么继承的	Python 语言	继承
如果继承两个类，他们有相同的方法会怎样？	Python 语言	继承
mro 顺序能改？	Python 语言	mro
实现一个功能，打印一个类里的所有全局变量	Python 语言	类属性
线程安全有了解吗？	系统编程	线程
python 的单例	Python 语言	单例
单元测试用什么框架	系统架构	单元测试
mysql 优化	数据库	数据库优化
什么是单点登录？	业务实现	单点登录
jwt 的优缺点	Web 基础	jwt
怎么清除 jwt？	Web 基础	jwt
什么是乐观锁	Web 基础	乐观锁
乐观锁与悲观锁的区别	Web 基础	乐观锁

msyql 为什么加索引就能快	数据库	索引
生成器的优缺点	Python 语言	生成器
说下微信登录流程	业务实现	微信登录
深拷贝与浅拷贝的去区别	Python 语言	深拷贝、浅拷贝
进程间的通信	系统编程	进程通信
linux 怎么查看所有的进程	Linux 系统使用	Linux 命令
说下 tcp 协议	网络编程	tcp
udp 与 tcp 的区别	网络编程	udp 与 tcp
为什么要三次握手?	网络编程	三次握手
http 与 https 的区别	网络编程	http 与 https
浏览器输入 URL 到服务器响应的过程	网络编程	http 请求过程
socket 实现 tcp 服务器	业务实现	socket
如何保证接口安全	业务实现	接口安全
1.flask 与 django 区别		
请简单描述一下微信二维码扫码登录原理。	业务实现	写出二维码的登录原理
前端有哪几种存储方式，他们之间的优缺点是什么?	其他	前端的存储方式
设计一个数据表（table）能够实现储存无限极目录树，并使用你最熟悉的语言，写一段简单 的代码，实现从表中获取并现实树结构！	数据库	数据表的实现
一根木棒烧完需要一小时，木棒数量不限，如果何才能计时 45 分钟。	业务实现	
哈希和加密的区别是什么，各自有什么用途！	数据库	哈希和加密的区别
utf-8 和 unicode 的区别是什么?	Web 基础	utf-8 和 unicode
描述数组，链表，队列、堆栈的区别?	数据结构与算法	数据结构
你对 Django 的认识！	WEB 框架	Django
什么是 flask！	WEB 框架	flask
python 的特点和优点是什么?	Python 语言	python 特点
python 常见的 PEP8 规范?	Python 语言	PEP8 规范

python 是如何进行内存管理的	Python 语言	内存管理
请用代码简答实现 stack。	业务实现	stack
简述 falsk 中蓝图的作用！	WEB 框架	蓝图的作用
请简要描述一下“re”模块的 split（），sub（），subn（）	Python 语言	re
xxx 项目，有个模块，统计全国各地区（省、市、县、乡、村）人员的就业和收入等相关情况 1、设计数据库表（写出表名字、表结构就可以）	业务实现	接口编写
设计接口，根据前端选择的地区和年份不同，统计对应人员的就业情况和人均收入情况（写出“接口名、入参、出参，统计的 sql）	业务实现	接口编写
3、如果需要按某种条件，导出相关数据到 excel，需要考虑哪些方面？	业务实现	接口编写
4、设置定时任务：从 www.xxx.com 网站抓取对应的数据，并更新数据库，你会采用哪些技术，更新数据库时，需要注意些什么？	业务实现	接口编写
5、如果系统中有一个对外的 api 接口，需要限流，你会怎么做？	业务实现	接口编写
两张关联表 a 和 b，关联字段伟 id，删除主表 a 中在副表 b 中没有的信息	数据库	数据库关联查询
pyhton 如何实现单例模式保证线程安全	设计模式	设计模式
如何调用远程接口	业务实现	远程接口调用
哈希和加密有什么区别，各自有什么用途	Python 语言	加密
实现一段程序，可以对一个整数 k（小于 20）进行拆分，	Python 语言	python 基础

得到所有的由从 1 到 k 的整数相加的形式。		
统计一段纯英文文章中，使用的单词出现频次从高到低排列，每个单词后应标明这个词出现的次数	Python 语言	文章单词出现次数统计
简述数据库索引的底层实现	数据库	索引
简述多进程、多线程、多协程之间的区别	Python 语言	多线程
redis 数据库的 zset 是如何实现有序的	数据库	redis
celery 运行中队列出现阻塞怎么解决	工具组件	celery
线上服务可能因为种种原因导致挂掉怎么办？	服务器	考察对服务器部署的掌握
描述数组、链表、队列、堆栈的区别？	数据结构与算法	考察对数据结构的掌握
跨域请求问题 django 怎么解决的（原理）	WEB 框架	考察 WEB 基知识的掌握
你常用的深度学习模型有哪些	深度学习	深度学习模型
介绍下 Seq2Seq 的原理	深度学习	Seq2Seq
图像中包含多个目标对象，如何识别标注，并返回他们的位置	深度学习	图像识别
如何解决模型过拟合	深度学习	过拟合
样本数据特征不均衡怎么处理	深度学习	特征不均衡
如何构建机器学习模型，特征不均衡的数据中	深度学习	特征不均衡
如何获取特征均匀的样本数据集	深度学习	特征
编程实现深度学习的卷积运算	深度学习	mysql
介绍下 sark，spark 的存储原理	深度学习	spark
图像识别的模型和算法了解多少	深度学习	图像识别
聊天机器人项目上线了没有	深度学习	nlp
应有的场景有哪些，实现的	深度学习	nlp

效果如何		
语言生成结果如何优化	深度学习	nlp
聊天机器人现状	深度学习	nlp
无人超市现状	深度学习	图像识别
如何检测图片中的汽车，并识别车型，如果有遮挡怎么办？	深度学习	图像识别
数字识别的流程。	深度学习	图像识别
深度学习中目标检测的常用方法，异同	深度学习	深度学习
说下高斯算子，Sobel 算子，拉普拉斯算子等，以及它们梯度方向上的区别。	深度学习	算子
常见的损失函数有哪些	深度学习	损失函数
怎么理解端到端	深度学习	端到端
多标签不平衡怎么处理，多任务不平衡怎么处理	深度学习	标签
如何处理梯度弥散问题？CNN-LSTM	深度学习	梯度弥散
Mysql 和 redis 如何做搭配	数据库	
Redis 的数据类型	数据库	
查询如何优化，如何做缓存？	数据库	
Socket 的流程	服务器	
Flask 和 django 的区别	WEB 框架	
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？	WEB 框架	考察对 web 框架的掌握
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	WEB 框架	考察对 web 框架的掌握
ngnix 的正向代理与反向代理？	服务器	考察对 ngnix 的掌握
mysql 有哪些索引	数据库	索引
各个索引有什么区别	数据库	索引
主键索引的数据结构是什么	数据库	主键索引
主键的数据结构是什么	数据库	主键

Mongodb 的索引和 MySQL 的索引的区别是什么	数据库	索引
数据库怎么做优化	数据库	优化
大表拆分了为什么能实现数据库优化，多查几个表不是会更慢吗	数据库	优化
做数据库优化的时候，你怎么知道是哪里需要优化	数据库	优化
怎么查看 MySQL 数据库的一项业务花了多长时间	数据库	优化
未发送的消息是怎么处理的	业务实现	发送消息
你的消息发送时间根据的是前端的时间还是后端的时间	业务实现	发送消息
你了解过 websocket 的原理和底层协议吗	Web 基础	websocket
es 全文检索原理	工具组件	es
你的地理位置是怎么存储的，查询的时候转换成了什么数据结构	业务实现	地理信息储存
那你说说 2dindex 的数据结构是什么	数据库	2dindex
说说页面静态化是怎么做的	业务实现	页面静态化
Celery 是干什么的	工具组件	celery
celery 任务队列是怎么存储的，失败了是怎么处理的，失败了后参数是存储在哪里的，怎么再次传递的	业务实现	celery
写一个冒泡排序	数据结构与算法	冒泡算法
写一个九九乘法表	业务实现	乘法表
用过 pygame 吗	Python 语言	pygame
列表去重	Python 语言	列表
a is b 与 a==b 的区别	Python 语言	对象引用
一个字典，根据 key 排序	Python 语言	字典排序
1 python 垃圾回收机制有哪些		
2 哈希表 是怎么实现的		
3 *args **kwargs 这种格式除了在传参还在哪里用到过		
4 实现打印当前目录下所有文件名不能用 os.walk 方法		

5 深拷贝，浅拷贝的原理区别		
6 常用的 GIT 命令有哪些		
7 评论一下 python 线程，再次反问那 python 线程是不是没什么用		
8 讲述一下 猴子补丁		
字典 m = {'a':0, 'b':1}，请用多种方式完成 key 和 value 的转换？	Python 语言	字典处理
a = ['aa', 'cd', 'ff', 'aaa', 'aac','ff','gg']，请对 a 进行去重并保持原来顺序不变？	Python 语言	列表操作
有一个文件 file.h ,从他得知 model.so 里面包含 里面包含一个名叫 funA 的方法，请问 python 怎么调用这个方法？	Python 语言	面向对象
a = [2, 5, 7, 8, 3, 9]，一行代码实现对列表 a 基数位置的元素进行乘以 5 后求和？	Python 语言	列表操作

def f(x,l=[]):
for i in range(x):
l.append(i*i)
print(l)
f(2)
f(3, [3, 2, 1])
f(3)
	Python 语言	列表操作

python 如何实现单例模式，保证线程安全	设计模式	多线程
简述 python2 与 python3 的区别	Python 语言	python 语言的区别
将 python 字典组成的列表以字典的键值排序	Python 语言	列表字典排序
写出一个类方法装饰器，使得装饰器内部可以调用实例方法	Python 语言	装饰器
简述深拷贝与钱拷贝的区别	Python 语言	深拷贝浅拷贝
mysql 中 innodb 的四种事务	数据库	mysql 数据库

级别		
进程中死锁出现的原因	Python 语言	进程锁
使用 linux 命令实现将 /root/www/html/int,sh 的软连接放到/usr/bin 目录下	Linux 系统使用	linux 软连接
列举出 OSI 标准的七层计算机网络协议，并指明 p2p 协议以及 UDP 协议位于七层协议中的哪一层	网络编程	网络协议
使用 python 实现折半查找有序列表中的某个元素	数据结构与算法	折半查找
请用 pyhton 实现洗牌算法	数据结构与算法	数据结构
在 sqlalchemy 中如何实现多对多关系	WEB 框架	sqlalchemy 用法
Django 中的信号量是什么	WEB 框架	Django
flask 使用 python3 需要注意什么？	WEB 框架	Flask
描述一下 python 中多线程和多进程的区别	系统编程	多进程多线程
python 是如何进行内存管理的	Python 语言	内存管理
描述一下 python 的 GC 机制	Python 语言	GC 机制
描述一下类装饰器和函数装饰器实现的过程	Python 语言	装饰器
用两个队列来实现一个栈	数据结构与算法	栈
你再项目中使用的 python 版本，python2 与 python3 的区别	Python 语言	python2 和 3 的对比
多线程下的单例对象的线程安全问题	系统编程	线程安全
考虑函数中参数默认值是列表的情况会出现什么问题	Python 语言	默认值
django 的中间件使用方法，如何自定义中间件，中间件的执行顺序按照列表执行，	WEB 框架	中间件
redis 中存储的数据的类型	数据库	数据类型
你觉得你这几个项目中那个是最具难度，难度在哪儿	项目业务	项目流程

如果带你的人给你分配的活,是错误的,今天必须要上线,你该怎么做	项目业务	项目业务
说一下 select 和 epoll 的区别	Python 语言	epoll 和 select 的区别
简述 python 垃圾回收机制	python 语言	python 的 GC 机制
写一个装饰器能够实现参数的传递	python 语言	装饰器的理解
写一个静态方法类方法实力方法，他们之间有什么不同	Python 语言	对类、静态、实例方法的理解
python 中的日志如何使用	Web 基础	loger 日志的理解
二叉树如何求两个叶节点的最近公共祖先	数据结构与算法	数据结构与算法
你常用的深度学习模型有哪些	深度学习	深度学习模型
介绍下 RNN 的原理	深度学习	RNN
CNN 的思想是什么?	深度学习	CNN
如何解决模型过拟合	深度学习	过拟合
filter 尺寸的选择	深度学习	图像识别
如何构建机器学习模型，特征不均衡的数据中	深度学习	图像识别
输出尺寸计算公式	深度学习	特征
编程实现深度学习的卷积运算	深度学习	卷积
pooling 池化的作用	深度学习	pooling
图像识别的模型和算法了解多少	深度学习	图像识别
RNN、LSTM、GRU 区别?	深度学习	深度学习
应有的场景有哪些，实现的效果如何	深度学习	图像识别
什么是梯度消失和梯度爆炸?	深度学习	梯度下降
常用的激活函数有哪些?	深度学习	激活函数
推导 BP 算法	深度学习	BP 算法
如何检测图片中的汽车，并识别车型，如果有遮挡怎么办?	深度学习	图像识别
为什么神经网络中深度网络的表现比广度网络表现好?	深度学习	神经网络

深度学习中目标检测的常用方法，异同	深度学习	深度学习
说下高斯算子，Sobel 算子，拉普拉斯算子等，以及它们梯度方向上的区别。	深度学习	算子
常见的损失函数有哪些	深度学习	损失函数
常用的参数更新方法有哪些？	深度学习	调参
常用边缘检测有哪些算子，各有什么特性？	深度学习	标签
梯度下降法找到的一定是下降最快的方向么？	深度学习	梯度下降
设计模式,写一段代码实现,A 变,所有注册过的 B 都要收到 A 改变的信息	Python 语言	设计模式
如何实现某个定时任务	业务实现	定时任务
JWT 是否可以解决跨域,请说明?	Web 基础	JWT
写一段代码实现二维码登录?	业务实现	二维码
WebSocket 能否使用 Django 实现?可以就写代码是实现它,不行说明原因	网络编程	socket
编函数,合并两个 dict	业务实现	python 语言基础
编写快速排序程序并在 main 函数中测试	Python 语言	单元测试
编写数的数据结构，随机生成 20 个节点，插入树中，构造一个树，然后实现树的前序遍历，输出结果，可以用循环方法或者递归方法。	数据结构与算法	算法
设计接口，根据前端选择的地区和年份不同，统计对应人员的就业情况和人均收入情况（写出：接口名，入参，出参，统计的 sql）	项目业务	接口设计
如果系统中有一个对外的 api 接口，徐亚限流，你会怎么做？	项目业务	接口设计

如果需要按某种条件，导出相关数据到 excel，需要考虑那些方面？	项目业务	接口设计
哈希和加密的区别是什么，各自有什么用途！	数据结构与算法	加密算法
utf-8 和 unicode 的区别是什么？	Python 语言	字符集
写爬虫是用多线程好，还是多进程好，为什么？	爬虫	多进程和多线程在爬虫中应用
请设计一段程序，可以将一篇纯英问的文章中，使用的单词按出现频次，由高到低排列，每个单词后应标明这个词出现的次数。	Python 语言	代码编写
“：”.join('1,2,3,4,5'.split('.'))的结果是多少？	Python 语言	字符串的操作
请用一行代码实现'1,2,3'变成['1','2','3']	Python 语言	Python 语法
已知列表 x=[3,1,6,5],那么执行语句 y=list（reversed（x））之后，y 的值为多少？	Python 语言	Python 语法
表达式 list（range（6））[::-2]的结果是多少？	Python 语言	Python 语法
a=[{i:j}for i,j in enumerate(range(5))],请写出 a 的最终结果。	Python 语言	Python 语法
列举几种你曾经常用的 Python 包并且解释其功能以及用法	Python 语言	常用的包
描述 TCP/IP 协议的层次结构，以及每一层中重要协议	Python 语言	网络层次的划分
什么是并发，如何解决	Web 基础	高并发及解决方案
linux 查看文件的命令有哪几种？有什么区别	Linux 系统使用	linux 命令
在 vi 编辑器里，那个命令能将光标移动到第 200 行	Python 语言	vi 操作
在 vi 编辑器里，如何进行字符串查找替换	Python 语言	vi 操作
vi 的三种模式如何切换	Python 语言	vi 操作

python 中如何快速的交换两个变量的值	Python 语言	变量使用
如何快速实现一个列表的倒叙	Python 语言	列表操作
项目中注册这块如何避免用户重复注册？	业务实现	如何保证用户只能注册一次
数据库中什么是基本表什么是视图	数据库	数据表的认识
写一个方法实现输出 1-100 之间的所有素数（素数指的是除了 1 和本身不能有其他因数）	Python 语言	函数的使用
给定一个整数组，快速实现升序排序，a=[9,4,5,3,6]，输出为 a=[3,4,5,6,9]	数据结构与算法	快排，冒泡等常用排序方法
动态创建类的方法及应用场景	Python 语言	类的创建
redis 如何实现持久化存储	数据库	redis 持久化
简述一下 celery 的组成	Web 基础	celery 的认识
购物车以及用户支付的业务逻辑	业务实现	业务逻辑
说一说对多进程的理解	Python 语言	进程
说一说数据库的理解	数据库	数据库
如何处理高并发	系统架构	高并发
MySQL 与 Redis 有何不同	数据库	MySQL
聊聊设计模式	设计模式	设计模式
单例模式用过吗	设计模式	单例
Django 的架构聊一下	WEB 框架	Django
如何进行测试工作的	其他	测试
库存不足如何处理	业务实现	秒杀业务
线程与进程的区别	Python 语言	进程
查看进程的 Linux 命令	Linux 系统使用	Linux
说说有哪些设计模式	设计模式	设计模式
说说数据库优化	数据库	MySQL
说说 MySQL 与 nosql 的区别	数据库	MySQL
为甚用 Redis 而不用 MySQL	数据库	数据库
为什么用 orm 框架	WEB 框架	orm
说说 HTTP 协议	Web 基础	HTTP
Git 工作流	Git	Git
Nginx 怎么做端口区别	服务器	Nginx

Django 中 model 的继承有多少钟？	WEB 框架	Django
手写冒泡排序	数据结构与算法	数据结构与算法
is 和==的区别	Python 语言	Python 基础
手写单例设计模式	设计模式	单例
提问 redis 的过期机制和淘汰机制		
怎么实现 grpc，大体说说推荐系统的实现流程		
项目中解决了哪些问题最有成就感		
es 是怎么管理索引，怎么清理的？		
有没有了解过 tensorflow		
大型软件测试包括哪四个步骤？	其他	软件测试基础
软件详细设计工具可分为哪三类？	其他	软件设计工具
数据流图的基本四种成分是什么？	其他	软件设计
基于软件的功能划分，软件可分为那三种？	其他	软件测试基础
python 中 list、set、dict、tuple 有什么区别，各自分别应用在什么场景下；	python 语言	python 基本数据类型
说明 session 和 cookie 区别和联系	web 基础	会话机制
说明一个 web 框架包含的主要功能和组件	web 基础	web 框架组成
说明 nginx 和其他反向代理服务器和 python web 程序的通讯方式	web 基础	nginx 和 python web 的通讯过程
常见的 python web 的部署方案	运维	项目部署
__inti__ 和 __new__的区别	python 语言	python 中 init 和 new 的区别
请使用代码实现单例模式	设计模式	单例模式
使用 yield 实现生成器	python 语言	生成器
说明 os sys 模块的作用，并列举常用的模块方法；	python 语言	os 和 sys 模块的作用

@classmethod @staticmethod @property 的意思	python 语言	类方法，静态方法，属性的使用
python 里 search 和 match 的区别	python 语言	search 和 match 的区别
解释一下，Gevent 和 threading、multipropcess 之间的联系	系统编程	gevent 和多线程，多进程
正则表达式里的贪婪和非贪婪模式	python 语言	正则表达式
解释 session 和 cookie 以及 xsrf 的防范	web 基础	会话机制
请设计一个系统消息功能，实现对个人，组，全员发送消息，用户可以看到未读消息的数量	业务实现	系统消息功能
cookie 和 session 的区别	web 基础	会话机制
http 和 https 的区别	web 基础	http 协议
git 创建分支并切换到分支上	git	git 命令
git 合并 dev 分支到 master 分支	git	git 命令
推送本地分支到远程分支上	git	git 命令
请列举你知道的 python 代码检测工具及他们的区别	Python 语言	代码检测
对单元测试的理解并列举 python 单元测相关工具和库	Python 语言	单元测试
python 如何捕获异常（输出异常）	Python 语言	python 异常处理
python 是如何进行内存管理的	Python 语言	内存管理
简述 cookie 和 session 之间的关系	Python 语言	cookid 和 session
mysql 存储引擎区别	数据库	mysql 引擎
mysql 如何进行多表查询	数据库	mysql 查询
简述 sql 注入的攻击原理及如何在代码层面防止 sql 注入	数据库	sql 注入
innoDB 有哪些特性	数据库	mysql 引擎

请列出一些 mysql 数据库查询优化的技巧	数据库	mysql 查询优化
列举常见的 HTTP 头及作用	网络编程	HTTP 状态码
列举常见的 http 状态码及意义	网络编程	HTTP 头
简述对 RESTfulAPI 设计规范的理解	网络编程	api 设计
简述 HTTP 缓存机制	网络编程	HTTTP 缓存
1. python 代码： '-'.join(x.split()[2:]) a) 解释上述代码涵义。（提示：x 可能为字符串，也可能为其它任意对象） b) 上面代码什么情况会报错？	Python 语言	字符串操作
2. 请给出一下列表操作的答案： （1） a=[1, 2, 3, 4, 5], a[::2]=? a[-2:] = ? （2）一行代码实现对列表 a 中的 偶数位置的各元素加 3 后求和 ?	Python 语言	列表操作
简述一下观察者模式	Python 语言	观察者模式
3. List = [-2, 1, 2, 3, -6], 如何实现将 List 中内容以绝对值大小 从小到大排序。 (1) 列表的 sort 方法和 sorted 的区别是什么？	Python 语言	列表操作
简述同步异步模型	Python 语言	同步异步

4. 有一篇英文文章保存在 a.txt 中，请用 python 实现统计这篇文章内每个单词的出现频率，并返回出现频率最高的前 10 个单词及其出现次数（只考虑空格，标点符号可忽略） （1） 追加需求：引号内元素需要算作一个单词，如何实现？	Python 语言	统计词频
写一个装饰器，用于打印函数执行时长	Python 语言	装饰器
6. Python 中 GIL 是什么，有什么作用及影响？线程，进程，协程的区别？	Python 语言	GIL
7. 请用尽可能简短的语言描述 深拷贝，装饰器，闭包，元类，描述符等概念。	Python 语言	概念
python 中的 GC 机制	Python 语言	GC 机制
如何进行内存管理	Python 语言	内存管理
for I in range(5,0,-1):print	Python 语言	列表生成
通过代码对任意列表中元素去重	Python 语言	列表去重
反扒策略及解决方法	爬虫	反扒
通常使用的爬虫程序分为哪些功能模块，每个模块使用哪些技术	爬虫	爬虫模块
爬取数据如何去重	爬虫	爬虫去重

Python 学科最新课程视频资源

Python 学科最新课程视频资源详见资源 word 文件。

传智播客™
www.itcast.cn

万人万薪 就业行动

扫描二维码领取更多独家资源