**Python基础篇**

**1：为什么学习Python**

家里有在这个IT圈子里面，也想让我接触这个圈子，然后给我建议学的Python，  
然后自己通过百度和向有学过Python的同学了解了Python，Python这门语言，入门比较简单，  
它简单易学，生态圈比较强大，涉及的地方比较多，特别是在人工智能，和数据分析这方面。在未来我觉得是往自动化，  
人工智能这方面发展的，所以学习了Python

**2：通过什么途径学习Python**

刚开始接触Python的时候，到网上里面跟着视频学基础，再后来网上到看技术贴，然后看到有人推荐廖雪峰的Python教程，

练项目到GitHub上面找一些小项目学习。

**3：谈谈对Python和其他语言的区别**

[复制代码](javascript:void(0);)

Python属于解释型语言，当程序运行时，是一行一行的解释，并运行，所以调式代码很方便，开发效率高，  
还有龟叔给Python定位是任其自由发展、优雅、明确、简单，所以在每个领域都有建树，所有它有着非常强大的第三方库，  
特点：  
语法简洁优美，功能强大，标准库与第三方库都非常强大，而且应用领域也非常广  
可移植性，可扩展性，可嵌入性  
缺点：  
　　运行速度慢，  
  
- 解释型

- python/php

- 编译型

- c/java/c#

- Python弱类型

[复制代码](javascript:void(0);)

 （1）与java相比：在很多方面，Python比Java要简单，比如java中所有变量必须声明才能使用，而Python不需要声明,用少量的代码构建出很多功能;（高效的高级数据结构）

（2）与php相比：python标准包直接提供了工具，并且相对于PHP代码更易于维护;

（3）Python与c相比：

Python 和 C Python这门语言是由C开发而来

　　对于使用：Python的类库齐全并且使用简洁，如果要实现同样的功能，Python 10行代码可以解决，C可能就需要100行甚至更多.  
　　对于速度：Python的运行速度相较与C，绝逼是慢了

**Python的优势：**

1、Python 易于学习;

2、用少量的代码构建出很多功能;（高效的高级数据结构）

3、Python 拥有最成熟的程序包资源库之一;

4、Python完全支持面向对象;

5、Python 是跨平台且开源的。

6、动态类型:

**4：简述解释型和编译型编程语言**

解释型：就是边解释边执行（Python，php）

编译型：编译后再执行（c、java、c#）

**5：Python的解释器种类以及相关特点？**

[复制代码](javascript:void(0);)

CPython  
  
是官方版本的解释器：CPython。是使用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。  
CPython是使用最广的Python解释器。教程的所有代码也都在CPython下执行。  
  
IPython  
IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强，但是执行Python代码的功能和CPython是完全一样的。CPython用>>>作为提示符，而IPython用In [序号]:作为提示符。

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

PyPy  
  
由Python写的解释器，它的执行速度是最快。PyPy采用JIT技术，对Python代码进行动态编译（注意不是解释），  
绝大部分Python代码都可以在PyPy下运行，但是PyPy和CPython有一些是不同的，这就导致相同的Python代码在两种解释器下执行可能会有不同的结果。  
  
Jython  
Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。  
  
IronPython  
IronPython和Jython类似，只不过IronPython是运行在.Net平台上的Python解释器，可以直接把Python代码编译成.Net的字节码。  
  
小结：  
　　Python的解释器很多，但使用最广泛的还是CPython。如果要和Java或.Net平台交互，最好的办法不是用Jython或IronPython，而是通过网络调用来交互，确保各程序之间的独立性。

[复制代码](javascript:void(0);)

**6：位和字节的关系**

1字节 = 8 位  
位（bit），数据存储是以“字节”（Byte）为单位，数据传输是以大多是以“位”（bit，又名“比特”）为单位，  
一个位就代表一个0或1（即一个二进制），二进制是构成存储器的最小单位，每8个位（bit，简写为b）组成一个字节（Byte，简写为B），  
字节是最小一级的信息单位

**7：b、B、KB、MB、GB的关系**

b --->位(bit)

B --->字节      一个字节等于8位

1B = 8 bit

1kb = 1024 B

1 MB = 1024 KB

1 GB = 1024 MB

**8:PE8规范**

[复制代码](javascript:void(0);)

1、使用4个空格而不是tab键进行缩进。

2、每行长度不能超过79

3、使用空行来间隔函数和类，以及函数内部的大块代码

4、必要时候，在每一行下写注释

5、使用文档注释，写出函数注释

6、在操作符和逗号之后使用空格，但是不要在括号内部使用

7、命名类和函数的时候使用一致的方式，比如使用CamelCase来命名类，

使用lower\_case\_with\_underscores来命名函数和方法

8、在类中总是使用self来作为默认

9、尽量不要使用魔法方法

10、默认使用UTF-8，甚至ASCII作为编码方式  
11、换行可以使用反斜杠，最好使用圆括号。  
12、不要在一句import中多个库，  
空格的使用

1. 各种右括号前不要加空格。
2. 逗号、冒号、分号前不要加空格。
3. 函数的左括号前不要加空格。如Func(1)
4. 序列的左括号前不要加空格。如list[2]
5. 操作符左右各加一个空格，不要为了对齐增加空格
6. 函数默认参数使用的赋值符左右省略空格
7. 不要将多句语句写在同一行，尽管使用‘；’允许
8. if/for/while语句中，即使执行语句只有一句，也必须另起一行

 函数命名使用全部小写的方式，常量命名使用大写，类属性（方法和变量）使用小写  
类的命名首字母大写

[复制代码](javascript:void(0);)

**9：通过代码实现如下转换(进制之间转换）**

[复制代码](javascript:void(0);)

# 二进制转换成十进制-->int

v = "0b1111011"

b = int(v,2)

print(b) # 123

# 十进制转换成二进制--->bin

v2 = 18

print(bin(int(v2)))

# 0b10010

# 八进制转换成十进制

v3 = "011"

print(int(v3))

# 11

# 十进制转换成八进制：---> oct

v4 = 30

print(oct(int(v4)))

# 0o36

# 十六进制转换成十进制：

v5 = "0x12"

print(int(v5,16))

# 18

# 十进制转换成十六进制：---> hex

v6 = 87

print(hex(int(v6)))

# 0x57

[复制代码](javascript:void(0);)

**10:请编写一个函数实现将IP地址转换成一个整数**

[复制代码](javascript:void(0);)

请编写一个函数实现将IP地址转换成一个整数。

如 10.3.9.12 转换规则为：

10 00001010

  3 00000011

9 00001001

  12 00001100

再将以上二进制拼接起来计算十进制结果：00001010 00000011 00001001 00001100 = ？

def v1(addr):

# 取每个数

id = [int(x) for x in addr.split(".")]

print(id)

return sum(id[i] << [24, 16, 8, 0][i] for i in range(4))

print(v1("127.0.0.1"))

# [127, 0, 0, 1]

# 2130706433

[复制代码](javascript:void(0);)

------------------------------------------------

**11、python递归的最大层数？998**

**12：求结果(and or or)**

[复制代码](javascript:void(0);)

1. 求结果：1 or 3

print(1 or 3) # 1

2. 求结果：1 and 3

print(1 and 3) # 3

3. 求结果：0 and 2 and 1

print(0 and 2 and 1) # 0

4. 求结果：0 and 2 or 1

print(0 and 2 or 1) # 1

5. 求结果：0 and 2 or 1 or 4

print(0 and 2 or 1 or 4) # 1

6. 求结果：0 or Flase and 1

print(0 or False and 1) # Flase

总结：

　　# x or y 如果 x为真，则值为x， 否则为y

　　# x and y 如果 x 为真，则值为 y，否则为 x

[复制代码](javascript:void(0);)

**运算符**

1. 求结果：2 & 5

print(2 & 5) # 10 & 101 => 000 => 0

2. 求结果：2 ^ 5

print(2 ^ 5) # 10 ^ 101 => 111 => 1\*2\*\*0+1\*2\*\*1+1\*2\*\*2=1+2+4=7

**13 ：ascii、unicode、utf-8、gbk 区别**

[复制代码](javascript:void(0);)

python2内容进行编码（默认ascii）,而python3对内容进行编码的默认为utf-8。

ascii 最多只能用8位来表示（一个字节），即：2\*\*8 = 256，所以，ASCII码最多只能表示 256 个符号。

unicode 万国码，任何一个字符==两个字节

utf-8 万国码的升级版 一个中文字符==三个字节 英文是一个字节 欧洲的是 2个字节

gbk 国内版本 一个中文字符==2个字节 英文是一个字节

gbk 转 utf-8 需通过媒介 unicode

[复制代码](javascript:void(0);)

**14:字节码和机器码的区别**

机器码，学名机器语言指令，有时也被称为原生码，是电脑的CPU可直接解读的数据。

字节码是一种中间状态（中间码）的二进制代码（文件）。需要直译器转译后才能成为机器码。

[复制代码](javascript:void(0);)

什么是机器码

机器码(machine code)，学名机器语言指令，有时也被称为原生码（Native Code），是电脑的CPU可直接解读的数据。

通常意义上来理解的话，机器码就是计算机可以直接执行，并且执行速度最快的代码。

总结：机器码是电脑CPU直接读取运行的机器指令，运行速度最快，但是非常晦涩难懂，也比较难编写

什么是字节码

字节码（Bytecode）是一种包含执行程序、由一序列 op 代码/数据对 组成的二进制文件。  
字节码是一种中间码，它比机器码更抽象，需要直译器转译后才能成为机器码的中间代码。

总结：字节码是一种中间状态（中间码）的二进制代码（文件）。需要直译器转译后才能成为机器码。

[复制代码](javascript:void(0);)

-----------

[复制代码](javascript:void(0);)

#is 比较的是内存地址

#== 比较的是值

# int 具有范围：-5---256

#对于int 小数据池

范围：-5----256 创建的相间的数字，都指向同一个内存地址

#对于字符串 （面试）

1、小数据池 如果有空格，那指向两个内存地址，

2、长度不能超过 20

3、不能用特殊字符

i = 'a'\*20

j = 'a'\*20

print(i is j) # True

i = "a"\*21

j = "a"\*21

print(i is j) # False

关于编码所占字节

unicode： 所有字符（无论英文、中文等） 1个字符：2个字节

gbk：一个字符，英文1个字节，中文两个字节

utf-8：英文1个字节、 欧洲：2个字节， 亚洲：3个字节

在utf-8中，一个中文字符占用3个字节

在gbk中一个汉字占用2个字节

黎诗 = utf-8(6字节)=48

黎诗 = gbk(4字节)=32

字节和位的关系。

　　#一个字节(byte) = 8 位(bit)

　　# 位为最小的单位

简述变量命名规范

　　#1、以字母，数字，下划线任由结合

　　#2、不能以命名太长，不使用拼音，中文

　　#3、不能以数字开头

　　#4、不能用关键词

[复制代码](javascript:void(0);)

**15:三元运算写法和应用场景？**

[复制代码](javascript:void(0);)

应用场景：简化if语句  
# 关于三元运算

# 结果+　if + 条件 + else + 结果

result='gt' if 1>3 else 'lt'

print(result) # lt

# 理解：如果条件为真，把if前面的值赋值给变量，否则把else后面的值赋值给变量。

lambda 表达式

temp = lambda x,y:x+y

print(temp(4,10)) # 14

可替代：

def foo(x,y):

return x+y

print(foo(4,10)) # 14

[复制代码](javascript:void(0);)

**16:Python3和Python2的区别？**

[复制代码](javascript:void(0);)

1：打印时，py2需要可以不需要加括号，py3 需要

python 2 ：print ('lili') , print 'lili'

python 3 : print ('lili')

python3 必须加括号

exec语句被python3废弃，统一使用exec函数

2：内涵

Python2：1，臃肿，源码的重复量很多。

　 2，语法不清晰，掺杂着C，php，Java，的一些陋习。

Python3：几乎是重构后的源码，规范，清晰，优美。

3、输出中文的区别

python2：要输出中文 需加 # -\*- encoding:utf-8 -\*-

Python3 ： 直接搞

4：input不同

python2 ：raw\_input

python3 ：input 统一使用input函数

5：指定字节

python2在编译安装时，可以通过参数-----enable-unicode=ucs2 或-----enable-unicode=ucs4分别用于指定使用2个字节、4个字节表示一个unicode；

python3无法进行选择，默认使用 ucs4

查看当前python中表示unicode字符串时占用的空间：

impor sys

print（sys.maxunicode）

#如果值是65535，则表示使用usc2标准，即：2个字节表示

#如果值是1114111，则表示使用usc4标准，即：4个字节表示

6：

py2：xrange

　　　　range

py3：range 统一使用range，Python3中range的机制也进行修改并提高了大数据集生成效率

7：在包的知识点里

包：一群模块文件的集合 + \_\_init\_\_

区别：py2 ： 必须有\_\_init\_\_

　　　py3：不是必须的了

8：不相等操作符"<>"被Python3废弃，统一使用"!="

9：long整数类型被Python3废弃，统一使用int

10：迭代器iterator的next()函数被Python3废弃，统一使用next(iterator)

11：异常StandardError 被Python3废弃，统一使用Exception

12：字典变量的has\_key函数被Python废弃，统一使用in关键词

13：file函数被Python3废弃，统一使用open来处理文件，可以通过io.IOBase检查文件类型

[复制代码](javascript:void(0);)

**17：用一行代码实现数值交换**

a = 1   
b = 2

a, b = b, a

**18：Python3和Python2中int和long区别**

在python3里，只有一种整数类型int,大多数情况下，和python２中的长整型类似。

**19：xrange和range的区别**

都在循环时使用，xrange内存性能更好，xrange用法与range完全相同，range一个生成list对象，xrange是生成器

要生成很大的数字序列的时候，用xrange会比range性能优很多，因为不需要一上来就开辟一块很大的内存空间。

在python2中：

range([start,] stop[, step])，根据start与stop指定的范围以及step设定的步长，生成一个序列

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif例子

xrange用法与range完全相同，所不同的是生成的不是一个数组，而是一个生成器。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif例子

由上面的示例可以知道：要生成很大的数字序列的时候，用xrange会比range性能优很多，因为不需要一上来就开辟一块很大的内存空间，这两个基本上都是在循环的时候用。

在 Python 3 中，range() 是像 xrange() 那样实现，xrange()被抛弃。

**20：文件操作时：xreadlines和readlines的区别？**

readlines     返回一个列表

xreadlines   返回一个生成器

**21： 列列举布尔值为False的常见值？**

0，“”，{}，[],（），set（）

0 Flask 负数 不成立的表达式 None 等

**22. 字符串、列表、元组、字典每个常用的5个方法？**

[复制代码](javascript:void(0);)

字符串：  
字符串用单引号(')或双引号(")括起来，不可变

1，find通过元素找索引，可切片，找不到返回-1

2，index，找不到报错。

3，split 由字符串分割成列表，默认按空格。

4，captalize 首字母大写，其他字母小写。

5，upper 全大写。

6，lower 全小写。

7，title，每个单词的首字母大写。

8，startswith 判断以什么为开头，可以切片，整体概念。

9，endswith 判断以什么为结尾，可以切片，整体概念。

10，format格式化输出  
#format的三种玩法 格式化输出  
res='{} {} {}'.format('egon',18,'male') ==> egon 18 male  
res='{1} {0} {1}'.format('egon',18,'male') ==> 18 egon 18  
*res='{name} {age} {sex}'.format(sex='male',name='egon',age=18)*

11,strip 默认去掉两侧空格，有条件， 12，lstrip,rstrip 14,center 居中，默认空格。   
15，count查找元素的个数，可以切片，若没有返回0   
16，expandtabs 将一个tab键变成8个空格，如果tab前面的字符长度不足8个，则补全8个，  
 17，replace（old，new,次数）   
18，isdigit 字符串由字母或数字组成 isalpha, 字符串只由字母组成 isalnum 字符串只由数字组成   
19,swapcase 大小写翻转   
20，for i in 可迭代对象。   
**字典：**  
1无序（不能索引）2：数据关联性强3:键值对，键值对。唯一一个映射数据类型。

#字典的键必须是可哈希的 不可变类型。

在同一个字典中，键(key)必须是唯一的。

列表是有序的对象集合，字典是无序的对象集合。两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取

key： 输出所有的键

clear：清空

dic：删除的键如果没有则报错

pop：键值对删，有返回，没有原来的键会报错（自行设置返回键就不会报错）

popitem：随机删键值对

del：删除的键如果没有则报错

改 update

查 用get时。不会报错# 没有可以返回设定的返回值

注意：

1、字典是一种映射类型，它的元素是键值对。

2、字典的关键字必须为不可变类型，且不能重复。

3、创建空字典使用 { }。

**列表：**  
索引，切片，加，乘，检查成员。

增加：有三种，

append：在后面添加。

Insert按照索引添加，

expend：迭代着添加。

list.extend(seq) - 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）

pop 删除 (pop 有返回值)

remove 可以按照元素去删

clear 清空列表

del 1、可以按照索引去删除 2、切片 3、步长（隔着删）

改 1、索引 2、切片：先删除，再迭代着添加

list.count(obj) - 统计某个元素在列表中出现的次数

list.index(obj) - 从列表中找出某个值第一个匹配项的索引位置

list.reverse() - 反向列表中元素

list.sort([func]) - 对原列表进行排序

注意：

1、List写在方括号之间，元素用逗号隔开。

2、和字符串一样，list可以被索引和切片。

3、List可以使用+操作符进行拼接。

4、List中的元素是可以改变的。

**元组：**  
（）元组的元素不能修改

1、cmp(tuple1, tuple2)：比较两个元组元素。

2、len(tuple)：计算元组元素个数。

3、max(tuple)：返回元组中元素最大值。

4、min(tuple)：返回元组中元素最小值。

5、tuple(seq)：将列表转换为元组。

注意

1、与字符串一样，元组的元素不能修改。

2、元组也可以被索引和切片，方法一样。

3、注意构造包含0或1个元素的元组的特殊语法规则。

4、元组也可以使用+操作符进行拼接。

**Set（集合）**  
：集合（set）是一个无序不重复元素的序列。

可以使用大括号 { } 或者 set() 函数创建集合，注意：创建一个空集合必须用 set() 而不是 { }，因为 { } 是用来创建一个空字典。

[复制代码](javascript:void(0);)

**23、 lambda表达式格式以及应用场景？**

[复制代码](javascript:void(0);)

匿名函数：为了解决那些功能很简单的需求而设计的一句话函数

函数名 = lambda 参数 ：返回值

#参数可以有多个，用逗号隔开

#匿名函数不管逻辑多复杂，只能写一行，且逻辑执行结束后的内容就是返回值

#返回值和正常的函数一样可以是任意数据类型

lambda 表达式

temp = lambda x,y:x+y

print(temp(4,10)) # 14

可替代：

def foo(x,y):

return x+y

print(foo(4,10)) # 14

[复制代码](javascript:void(0);)

**24. pass的作用**

pass是空语句，是为了保持程序结构的完整性。pass 不做任何事情，一般用做占位语句。

**25. \*arg和\*\*kwarg作用**

 \*args代表位置参数，它会接收任意多个参数并把这些参数作为元祖传递给函数。  
\*\*kwargs代表的关键字参数，返回的是字典，位置参数一定要放在关键字前面

**26. is和==的区别**

[复制代码](javascript:void(0);)

a = 'lishi'

str1 = "li"

str2 = "shi"

str3 = str1 + str2

print("a == str3",a == str3)

print("a is str3",a is str3)

print("id(a)",id(a))

print("id(str3)",id(str3))

# a == str3 True == ---> 只需要内容相等

# a is str3 False is ---> 只需要内存地址相等

# id(a) 38565848

# id(str3) 39110280

[复制代码](javascript:void(0);)

is 比较的是两个实例对象是不是完全相同，它们是不是同一个对象，占用的内存地址是否相同。

== 比较的是两个对象的内容是否相等，即内存地址可以不一样，内容一样就可以了。默认会调用对象的 \_\_eq\_\_()方法。

**27：谈谈Python的深浅拷贝？以及实现方法和应用场景。**

浅拷贝只是增加了一个指针指向一个存在的地址，

而深拷贝是增加一个指针并且开辟了新的内存，这个增加的指针指向这个新的内存，  
采用浅拷贝的情况，释放内存，会释放同一内存，深拷贝就不会出现释放同一内存的错误

一层的情况：

[复制代码](javascript:void(0);)

import copy

# 浅拷贝

li1 = [1, 2, 3]

li2 = li1.copy()

li1.append(4)

print(li1, li2) # [1, 2, 3, 4] [1, 2, 3]

# 深拷贝

li1 = [1, 2, 3]

li2 = copy.deepcopy(li1)

li1.append(4)

print(li1, li2) # [1, 2, 3, 4] [1, 2, 3]

[复制代码](javascript:void(0);)

多层的情况：

[复制代码](javascript:void(0);)

import copy

# 浅拷贝 指向共有的地址

li1 = [1, 2, 3,[4,5],6]

li2 = li1.copy()

li1[3].append(7)

print(li1, li2) # [1, 2, 3, [4, 5, 7], 6] [1, 2, 3, [4, 5, 7], 6]

# 深拷贝 重指向

li1 = [1, 2, 3,[4,5],6]

li2 = copy.deepcopy(li1)

li1[3].append(7)

print(li1, li2) # [1, 2, 3, [4, 5, 7], 6] [1, 2, 3, [4, 5], 6]

[复制代码](javascript:void(0);)

**28. Python垃圾回收机制？**

引用计数

标记清除

分代回收

**29. Python的可变类型和不可变类型？**

可变数据类型：列表、字典、可变集合

不可变数据类型：数字、字符串、元组、不可变集合

**30、求结果**

def multipliers():

return [lambda x:i\*x for i in range(4)]

print([m(2) for m in multipliers()])

[复制代码](javascript:void(0);)

def a():

return [lambda x:i\*x for i in range(4)]

b=a() #返回个列表函数

# b[2](1)

print(b[1](1))

# print(type(b),b)

print([m(1) for m in a()])

print([i\*i for i in [1,2,3]])

[3, 3, 3, 3]

[1, 4, 9]

'''

def multipliers():

return [lambda x:i\*x for i in range(4)]

print([m(2) for m in multipliers()])

#解释：

　　函数返回值为一个列表表达式，经过4次循环结果为包含四个lambda函数的列表，

由于函数未被调用，循环中的i值未被写入函数，经过多次替代，循环结束后i值为3，

故结果为：6,6,6,6

func=lambda x:x+1

print(func(1))

#2

print(func(2))

#3

#以上lambda等同于以下函数

def func(x):

return(x+1)

'''

[复制代码](javascript:void(0);)

请修改multipliers的定义来产生期望的结果（0,2,4,6）。

def multipliers():

return （lambda x:i\*x for i in range(4)） #返回一个生成器表达式

print([m(2) for m in multipliers()])

[复制代码](javascript:void(0);)

-面试题2：  
现有两个元组(('a'),('b')),(('c'),('d'))，请使用python中匿名函数生成列表[{'a':'c'},{'b':'d'}]

#匿名函数形式：

l1=(('a'),('b'))

l2=(('c'),('d'))

ret=map(lambda n:{n[0]:n[1]},zip(l1,l2))

print(list(ret))

#列表表达式形式：

l1=(('a'),('b'))

l2=(('c'),('d'))

print([{n[0]:n[1]} for n in zip(l1,l2)])

**31、求结果**

[复制代码](javascript:void(0);)

v = dict.fromkeys(['k1', 'k2'], [])

v['k1'].append(666)

print(v)

v['k1'] = 777

print(v)

结果：

{'k1': [666], 'k2': [666]}

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

解释：

Python 字典(Dictionary) fromkeys() 函数用于创建一个新字典，以序列seq中元素做字典的键，value为字典所有键对应的初始值，默认为None。

v1 = dict.fromkeys(['k1', 'k2'])

print(v1) # {'k1': None, 'k2': None}

v2 = dict.fromkeys(['k1', 'k2'], [])

print(v2) # {'k1': [], 'k2': []}

[复制代码](javascript:void(0);)

**32、列举常见的内置函数**

**abs（）**

返回数字的绝对值

**map**

[复制代码](javascript:void(0);)

根据函数对指定序列做映射  
map()函数接收两个参数，一个是函数，一个是可迭代对象，map将传入的函数依次作用到序列的每个元素，并把结果作为新的list返回。  
  
返回值：  
　　Python2 返回列表  
　　Python3 返回迭代器  
  
例子1：

def mul(x):

return x\*x

n=[1,2,3,4,5]

res=list(map(mul,n))

print(res) #[1, 4, 9, 16, 25]  
  
例子2：abs() 返回数字的绝对值  
ret = map(abs,[-1,-5,6,-7])  
print(list(ret))  
# [1, 5, 6, 7]

[复制代码](javascript:void(0);)

**filter**

[复制代码](javascript:void(0);)

filter()函数接收一个函数 f(函数)和一个list（可迭代对象），这个函数 f的作用是对每个元素进行判断，返回 True或 False，

filter()根据判断结果自动过滤掉不符合条件的元素，返回由符合条件元素组成的新list。

def is\_odd(x):

return x % 2 == 1

v=list(filter(is\_odd, [1, 4, 6, 7, 9, 12, 17]))

print(v) #[1, 7, 9, 17]

[复制代码](javascript:void(0);)

**map与filter总结**

[复制代码](javascript:void(0);)

# filter 与 map 总结

# 参数: 都是一个函数名 + 可迭代对象

# 返回值: 都是返回可迭代对象

# 区别:

# filter 是做筛选的，结果还是原来就在可迭代对象中的项

# map 是对可迭代对象中每一项做操作的，结果不一定是原来就在可迭代对象中的项

[复制代码](javascript:void(0);)

**isinstance\type**

[复制代码](javascript:void(0);)

isinstance() 函数来判断一个对象是否是一个已知的类型，类似 type()。

isinstance() 与 type() 区别：

type() 不会认为子类是一种父类类型，不考虑继承关系。

isinstance() 会认为子类是一种父类类型，考虑继承关系。

如果要判断两个类型是否相同推荐使用 isinstance()。

# 例一  
a = 2  
print(isinstance(a,int)) *# True  
print(isinstance(a,str)) # False*

# type() 与 isinstance() 区别

class A:

pass

class B(A):

pass

print("isinstance",isinstance(A(),A)) # isinstance True

print("type",type(A()) == A) # type True

print('isinstance',isinstance(B(),A) ) # isinstance True

print('type',type(B()) == A) # type False

[复制代码](javascript:void(0);)

**zip 拉链函数**

[复制代码](javascript:void(0);)

# zip 拉链函数，

# 将对象中对应的元素打包成一个个元组，

# 然后返回由这些元组组成的列表迭代器。

# 如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同。

print(list(zip([0,1,3],[5,6,7],['a','b'])))

# [(0, 5, 'a'), (1, 6, 'b')]

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。

>>>a = [1,2,3]

>>> b = [4,5,6]

>>> c = [4,5,6,7,8]

>>> zipped = zip(a,b) # 打包为元组的列表

[(1, 4), (2, 5), (3, 6)]

>>> zip(a,c) # 元素个数与最短的列表一致

[(1, 4), (2, 5), (3, 6)]

>>> zip(\*zipped) # 与 zip 相反，可理解为解压，返回二维矩阵式

[(1, 2, 3), (4, 5, 6)]

[复制代码](javascript:void(0);)

**reduce**

[复制代码](javascript:void(0);)

'''

reduce() 函数

reduce() 函数会对参数序列中元素进行累积

函数将一个数据集合(链表、元组等)中的所有数据进行下列操作

'''

注意：

Python3已经将reduce() 函数从全局名字空间里移除了，它现在被放置在 fucntools 模块里，如果想要使用它，则需要通过引入 functools 模块来调用 reduce() 函数：

from functools import reduce

def add(x,y):

return x + y

print(reduce(add,[1,2,3,4,5]))

# 15

print(reduce(lambda x, y: x+y, [1,2,3,4,5])) # 15

print(reduce(add,range(1,101)))

# 5050

[复制代码](javascript:void(0);)

**33.**[**filter、map、reduce的作用？**](https://www.cnblogs.com/Xrinehart/p/3506467.html)

**内置函数：map、reduce、filter的用法和区别**

**map**:根据函数对指定序列做映射

[复制代码](javascript:void(0);)

map

参数

接收两个参数：一个是函数，一个是序列（可迭代对象）

返回值

Python2 返回列表

Python3 返回迭代器

# 例子：

# abs() 函数返回数字的绝对值

# 新的内容的个数等于原内容的个数

# ret = map(abs,[-1,-5,6,-7])

# print(list(ret))

# [1, 5, 6, 7]

[复制代码](javascript:void(0);)

**filter:**过滤函数 新的内容少于等于原内容的时候。才能使用filter

[复制代码](javascript:void(0);)

filter() 函数用于过滤序列，过滤不符合条件的元素，返回由符合条件元素组成的心列表

参数：

function 函数

iterable 可迭代对象

返回值:

返回列表

# 筛选大于10的数

def is\_odd(x):

if x>10:

return True

ret = filter(is\_odd,[1,4,5,7,8,9,76]) # 为迭代器

print(list(ret))

# [76]

[复制代码](javascript:void(0);)

**reduce**:对于序列内所有元素进行累计操作

[复制代码](javascript:void(0);)

'''

reduce() 函数

reduce() 函数会对参数序列中元素进行累积

函数将一个数据集合(链表、元组等)中的所有数据进行下列操作

'''

from functools import reduce

def add(x,y):

return x + y

print(reduce(add,[1,2,3,4,5]))

# 15

print(reduce(lambda x, y: x+y, [1,2,3,4,5])) # 15

print(reduce(add,range(1,101)))

# 5050

[复制代码](javascript:void(0);)

**34、 一行代码实现9\*9乘法表**

print('\n'.join([' '.join(['%s\*%s=%-2s' % (j, i, i \* j) for j in range(1, i + 1)]) for i in range(1, 10)]))

**35. 如何安装第三方模块？以及用过哪些第三方模块？**

[复制代码](javascript:void(0);)

1：pip包管理器

2：源码下载

-下载

-解压

-python setup.py build

-python setup.py install

[复制代码](javascript:void(0);)

用过的第三方模块：requests,pymysql,DbUtils,SQLAlchemy等

**36、 常用模块都有那些？**

re模块，os模块，json模块，time模块，

爬虫里面的requests/beautifulsoup4（bs4）

**37. re的match和search区别？**

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。

re.search 扫描整个字符串并返回第一个成功的匹配。

**38. 什么是正则的贪婪匹配？**

匹配一个字符串没有节制，能匹配多少就去匹配多少，知道没有匹配的为止

**39. 求结果：**

a. [ i % 2 for i in range(10) ]

print([ i % 2 for i in range(10) ]) # [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

print([ i for i in range(10) ]) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print([ 10 % 2]) # [0]

# %是个运算符。

b. ( i % 2 for i in range(10) )

[复制代码](javascript:void(0);)

print(( i % 2 for i in range(10) ))

# <generator object <genexpr> at 0x00000000020CEEB8> 生成器

# 在Python中，有一种自定义迭代器的方式，称为生成器（Generator）。

# 定义生成器的两种方式：

# 1.创建一个generator，只要把一个列表生成式的[]改成()，就创建了一个generator：

# generator保存的是算法，每次调用next()，就计算出下一个元素的值，直到计算到最后一个元素，  
没有更多的元素时，抛出StopIteration的错误。

# 2.定义generator的另一种方法。如果一个函数定义中包含yield关键字，那么这个函数就不再是一个普通函数，  
而是一个generator

[复制代码](javascript:void(0);)

**40. 求结果：**

a. 1 or 2  
b. 1 and 2  
c. 1 < (2==2)  
d. 1 < 2 == 2

[复制代码](javascript:void(0);)

>>> 1 or 2

1

>>> 1 and 2

2

>>> 1 < (2==2)

False

>>> 1 < 2 == 2

True

[复制代码](javascript:void(0);)

**41、def func(a,b=[]) 这种写法有什什么坑？**

[复制代码](javascript:void(0);)

def func(a,b = []):

b.append(1)

print(a,b)

func(a=2)

func(2)

func(2)

'''

2 [1]

2 [1, 1]

2 [1, 1, 1]

函数的默认参数是一个list 当第一次执行的时候实例化了一个list

第二次执行还是用第一次执行的时候实例化的地址存储

所以三次执行的结果就是 [1, 1, 1] 想每次执行只输出[1] ，默认参数应该设置为None

'''

[复制代码](javascript:void(0);)

**42、如何实现 “1,2,3” 变成 [‘1’,’2’,’3’]**

list("1,2,3".split(','))

**43. 如何实现[‘1’,’2’,’3’]变成[1,2,3]**

[复制代码](javascript:void(0);)

[int(x) for x in ['1','2','3']]

python 里如何把['1','2','3'] 变成[1,2,3]

a = ['1','2','3']

b = [int(i) for i in a]

print(b)

# [1, 2, 3]

[复制代码](javascript:void(0);)

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

 补充：

[复制代码](javascript:void(0);)

a=[1,2,3,4,5]，b=a和b=a[:]，有区别么？

a = [1,2,3,4,5]

b = a

b1 = a[:]

print(b) # [1, 2, 3, 4, 5]

# print(b) # [1, 2, 3, 4, 5]

b.append(6)

print("a",a) # a [1, 2, 3, 4, 5, 6]

print("b",b) # b [1, 2, 3, 4, 5, 6] 传递引用

print("b1",b1) # b1 [1, 2, 3, 4, 5] 拷贝

[复制代码](javascript:void(0);)

# 一个列表A=[2，3，4]，Python如何将其转换成B=[(2,3),(3,4),(4,2)]？

# B = zip(A, A[1:]+A[:1])

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

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

**46. 一行代码实现删除列表中重复的值**

list(set([1, 2, 3, 4, 45, 1, 2, 343, 2, 2]))

**47. 如何在函数中设置一个全局变量**

python中的global语句是被用来声明全局变量的。

[复制代码](javascript:void(0);)

x = 2

def func():

global x

x = 1

return x

func()

print(x) # 1

[复制代码](javascript:void(0);)

**48. logging模块的作用？以及应用场景？**

[复制代码](javascript:void(0);)

logging

模块定义的函数和类为应用程序和库的开发实现了一个灵活的事件日志系统

作用：可以了解程序运行情况，是否正常  
　　　　在程序的出现故障快速定位出错地方及故障分析

[复制代码](javascript:void(0);)

**49. 请用代码简答实现stack**

* Stack() 创建一个新的空栈
* push(item) 添加一个新的元素item到栈顶
* pop() 弹出栈顶元素
* peek() 返回栈顶元素
* is\_empty() 判断栈是否为空
* size() 返回栈的元素个数

[复制代码](javascript:void(0);)

# 实现一个栈stack,后进先出

'''

class Stack:

def \_\_init\_\_(self):

self.items = []

def is\_empty(self):

# 判断是否为空

return self.items == []

def push(self,item):

# 加入元素

self.items.append(item)

def pop(self):

# 弹出元素

return self.items.pop()

def peek(self):

# 返回栈顶元素

return self.items[len(self.items)-1]

def size(self):

# 返回栈的大小

return len(self.items)

if \_\_name\_\_ == "\_\_main\_\_":

stack = Stack()

stack.push("H")

stack.push("E")

stack.push("L")

print(stack.size()) # 3

print(stack.peek()) # L

print(stack.pop()) # L

print(stack.pop()) # E

print(stack.pop()) # H

'''

[复制代码](javascript:void(0);)

**50. 常用字符串格式化哪几种？**

1.占位符%

%d 表示那个位置是整数；%f 表示浮点数；%s 表示字符串。

print('Hello,%s' % 'Python')

print('Hello,%d%s%.2f' % (666, 'Python', 9.99)) # 打印：Hello,666Python10.00

2.format

print('{k} is {v}'.format(k='python', v='easy')) # 通过关键字

print('{0} is {1}'.format('python', 'easy')) # 通过关键字

**51. 简述 生成器、迭代器、可迭代对象 以及应用场景？**

**迭代器**

含有\_\_iter\_\_和\_\_next\_\_方法 (包含\_\_next\_\_方法的可迭代对象就是迭代器)

**生成器**

：包括含有yield这个关键字，生成器也是迭代器，调动next把函数变成迭代器。

[复制代码](javascript:void(0);)

应用场景：

range/xrange

- py2： range(1000000) ,会立即创建，xrange(1000000)生成器

- py3：range（10000000）生成器

- redis获取值  
conn = Redis(...)  
  
　　　　def hscan\_iter(self, name, match=None, count=None):  
　　　　　　"""  
　　　　　　Make an iterator using the HSCAN command so that the client doesn't  
　　　　　　need to remember the cursor position.

　　　　　　``match`` allows for filtering the keys by pattern

　　　　　　``count`` allows for hint the minimum number of returns  
　　　　　　"""  
　　　　　　cursor = '0'  
　　　　　　while cursor != 0:  
　　　　　　　　# 去redis中获取数据：12  
　　　　　　　　# cursor，下一次取的位置  
　　　　　　　　# data：本地获取的12条数数据  
　　　　　　　　cursor, data = self.hscan(name, cursor=cursor,match=match, count=count)  
　　　　　　　　for item in data.items():  
　　　　　　　　　　yield item

stark组件  
  
def index(request):

　　　　data = [  
　　　　　　{'k1':1,'name':'alex'},  
　　　　　　{'k1':2,'name':'老男孩'},  
　　　　　　{'k1':3,'name':'小男孩'},  
　　　　]  
　　　　new\_data = []  
　　　　for item in data:  
　　　　　　item['email'] = "xxx@qq.com"  
　　　　　　new\_data.append(item)

　　　　return render(request,'xx.html',{'data':new\_data})

[复制代码](javascript:void(0);)

**可迭代对象**

 一个类内部实现\_\_iter\_\_方法且返回一个迭代器。

[复制代码](javascript:void(0);)

应用场景：

- wtforms中对form对象进行循环时候，显示form中包含的所有字段。

class LoginForm(Form):

name = simple.StringField(

label='用户名',

validators=[

validators.DataRequired(message='用户名不能为空.'),

validators.Length(min=6, max=18, message='用户名长度必须大于%(min)d且小于%(max)d')

],

widget=widgets.TextInput(),

render\_kw={'class': 'form-control'}

)

pwd = simple.PasswordField(

label='密码',

validators=[

validators.DataRequired(message='密码不能为空.'),

validators.Length(min=8, message='用户名长度必须大于%(min)d'),

validators.Regexp(regex="^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[$@$!%\*?&])[A-Za-z\d$@$!%\*?&]{8,}",

message='密码至少8个字符，至少1个大写字母，1个小写字母，1个数字和1个特殊字符')

],

widget=widgets.PasswordInput(),

render\_kw={'class': 'form-control'}

)

form = LoginForm()

for item in form:

print(item)

- 列表、字典、元组

[复制代码](javascript:void(0);)

**装饰器**

[复制代码](javascript:void(0);)

装饰器：

能够在不修改原函数代码的基础上，在执行前后进行定制操作，闭包函数的一种应用

场景：

- flask路由系统

- flask before\_request

- csrf

- django内置认证

- django缓存

# 手写装饰器；

import functools

def wrapper(func):

@functools.wraps(func) #不改变原函数属性

def inner(\*args, \*\*kwargs):

执行函数前

return func(\*args, \*\*kwargs)

执行函数后

return inner

1. 执行wapper函数，并将被装饰的函数当做参数。 wapper(index)

2. 将第一步的返回值，重新赋值给 新index = wapper(老index)

@wrapper #index=wrapper(index)

def index(x):

return x+100

[复制代码](javascript:void(0);)

调用装饰器其实是一个闭包函数，为其他函数添加附加功能，不修改被修改的源代码和不修改被修饰的方式，装饰器的返回值也是一个函数对象。  
比如：插入日志、性能测试、事物处理、缓存、权限验证等，有了装饰器，就可以抽离出大量与函数功能本身无关的雷同代码并继续重用。

**52. 用Python实现一个二分查找的函数。**

二分查找算法：简单的说，就是将一个列表先排序好，比如按照从小到大的顺序排列好，当给定一个数据，比如3，查找3在列表中的位置时，可以先找到列表中间的数li[middle]和3进行比较，当它比3小时，那么3一定是在列表的右边，反之，则3在列表的左边，比如它比3小，则下次就可以只比较[middle+1, end]的数，继续使用二分法，将它一分为二，直到找到3这个数返回或者列表全部遍历完成（3不在列表中）

优点：效率高，时间复杂度为O(logN)；   
缺点：数据要是有序的，顺序存储。

[复制代码](javascript:void(0);)

li = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

def search(someone, li):

l = -1

h = len(li)

while l + 1 != h:

m = int((l + h) / 2)

if li[m] < someone:

l = m

else:

h = m

p = h

if p >= len(li) or li[p] != someone:

print("元素不存在")

else:

str = "元素索引为%d" % p

print(str)

search(3, li) # 元素索引为2

[复制代码](javascript:void(0);)

**53. 谈谈你对闭包的理解？**

[复制代码](javascript:void(0);)

ef foo():

m=3

n=5

def bar():

a=4

return m+n+a

return bar

>>>bar = foo()

>>>bar()

12

[复制代码](javascript:void(0);)

说明：  
bar在foo函数的代码块中定义。我们称bar是foo的内部函数。  
在bar的局部作用域中可以直接访问foo局部作用域中定义的m、n变量。  
简单的说，这种内部函数可以使用外部函数变量的行为，就叫闭包。

闭包的意义与应用

**54. os和sys模块的作用？**

os模块负责程序与操作系统的交互，提供了访问操作系统底层的接口;  
sys模块负责程序与python解释器的交互，提供了一系列的函数和变量，用于操控python的运行时环境。

[复制代码](javascript:void(0);)

os与sys模块的官方解释如下：

os: This module provides a portable way of using operating system dependent functionality.

这个模块提供了一种方便的使用操作系统函数的方法。

sys: This module provides access to some variables used or maintained by the interpreter and to   
functions that interact strongly with the interpreter.

这个模块可供访问由解释器使用或维护的变量和与解释器进行交互的函数。

os 常用方法

os.remove() 删除文件

os.rename() 重命名文件

os.walk() 生成目录树下的所有文件名

os.chdir() 改变目录

os.mkdir/makedirs 创建目录/多层目录

os.rmdir/removedirs 删除目录/多层目录

os.listdir() 列出指定目录的文件

os.getcwd() 取得当前工作目录

os.chmod() 改变目录权限

os.path.basename() 去掉目录路径，返回文件名

os.path.dirname() 去掉文件名，返回目录路径

os.path.join() 将分离的各部分组合成一个路径名

os.path.split() 返回( dirname(), basename())元组

os.path.splitext() 返回 (filename, extension) 元组

os.path.getatime\ctime\mtime 分别返回最近访问、创建、修改时间

os.path.getsize() 返回文件大小

os.path.exists() 是否存在

os.path.isabs() 是否为绝对路径

os.path.isdir() 是否为目录

os.path.isfile() 是否为文件

sys 常用方法

sys.argv 命令行参数List，第一个元素是程序本身路径

sys.modules.keys() 返回所有已经导入的模块列表

sys.exc\_info() 获取当前正在处理的异常类,exc\_type、exc\_value、exc\_traceback当前处理的异常详细信息

sys.exit(n) 退出程序，正常退出时exit(0)

sys.hexversion 获取Python解释程序的版本值，16进制格式如：0x020403F0

sys.version 获取Python解释程序的版本信息

sys.maxint 最大的Int值

sys.maxunicode 最大的Unicode值

sys.modules 返回系统导入的模块字段，key是模块名，value是模块

sys.path 返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值

sys.platform 返回操作系统平台名称

sys.stdout 标准输出

sys.stdin 标准输入

sys.stderr 错误输出

sys.exc\_clear() 用来清除当前线程所出现的当前的或最近的错误信息

sys.exec\_prefix 返回平台独立的python文件安装的位置

sys.byteorder 本地字节规则的指示器，big-endian平台的值是'big',little-endian平台的值是'little'

sys.copyright 记录python版权相关的东西

sys.api\_version 解释器的C的API版本

总结：

os模块负责程序与操作系统的交互，提供了访问操作系统底层的接口;  
sys模块负责程序与python解释器的交互，提供了一系列的函数和变量，用于操控python的运行时环境。

[复制代码](javascript:void(0);)

**55. 如何生成一个随机数？**

import random

print(random.random()) # 用于生成一个0到1的随机符点数: 0 <= n < 1.0

print(random.randint(1, 1000)) # 用于生成一个指定范围内的整数

**56. 如何使用python删除一个文件？**

[复制代码](javascript:void(0);)

import os

file = r'D:\test.txt'

if os.path.exists(file):

os.remove(file)

print('delete success')

else:

print('no such file:%s' % file)

[复制代码](javascript:void(0);)

**57. 谈谈你对面向对象的理解**

**三大特性以及解释？**

面对对象是一种编程思想，以类的眼光来来看待事物的一种方式。将有共同的属性和方法的事物封装到同一个类下面。

继承：将多个类的共同属性和方法封装到一个父类下面，然后在用这些类来继承这个类的属性和方法

封装：将有共同的属性和方法封装到同一个类下面

* 第一层面：创建类和对象会分别创建二者的名称空间，我们只能用类名.或者obj.的方式去访问里面的名字，这本身就是一种封装
* 第二层面：类中把某些属性和方法隐藏起来(或者说定义成私有的)，只在类的内部使用、外部无法访问，或者留下少量接口（函数）供外部访问。

多态：Python天生是支持多态的。指的是基类的同一个方法在不同的派生类中有着不同的功能

**58. Python面向对象中的继承有什么特点**

[复制代码](javascript:void(0);)

继承概念的实现方式主要有2类：实现继承、接口继承。

实现继承是指使用基类的属性和方法而无需额外编码的能力；

接口继承是指仅使用属性和方法的名称、但是子类必须提供实现的能力(子类重构爹类方法)；

python 两种类：经典类 新式类

python3 新式类 —— 都默认继承object class Animal(object): == class Animal:

python2 经典类和新式类 并存

class Animal: 经典类 —— 继承顺序 个别使用方法

class Animal(object): 新式类

继承分为单继承和多继承

Python是支持多继承的

如果没有指定基类，python的类会默认继承object类，object是所有python类的基类，它提供了一些常见方法（如\_\_str\_\_）的实现。

[复制代码](javascript:void(0);)

补充继承的应用（面试题）

[复制代码](javascript:void(0);)

1、对象可以调用自己本类和父类的所有方法和属性， 先调用自己的 自己没有才调父类的。谁（对象）调用方法，方法中的self就指向谁

class Foo:

def \_\_init\_\_(self):

self.func()

def func(self):

print('Foo.func')

class Son(Foo):

def func(self):

print('Son.func')

s = Son()

# Son.func

========================================================

class A:

def get(self):

self.say()

def say(self):

print('AAAAA')

class B(A):

def say(self):

print('BBBBB')

b = B()

b.get() #输出结果为：BBBBB

[复制代码](javascript:void(0);)

**59. 面向对象深度优先和广度优先是什么？**

Python的类可以继承多个类，Python的类如果继承了多个类，那么其寻找方法的方式有两种

当类是经典类时，多继承情况下，会按照深度优先方式查找 py3

当类是新式类时，多继承情况下，会按照广度优先方式查找 py2

简单点说就是：经典类是纵向查找，新式类是横向查找

经典类和新式类的区别就是，在声明类的时候，新式类需要加上object关键字。在python3中默认全是新式类

**60. 面向对象中super的作用？**

用于子类继承基类的方法

[复制代码](javascript:void(0);)

class FooParent(object):

def \_\_init\_\_(self):

self.parent = 'I\'m the parent.'

print('Parent')

print('1111')

def bar(self, message):

print("%s from Parent" % message)

class FooChild(FooParent):

def \_\_init\_\_(self):

# super(FooChild,self) 首先找到 FooChild 的父类（就是类 FooParent），然后把类B的对象 FooChild 转换为类 FooParent 的对象

super(FooChild, self).\_\_init\_\_()

print('Child')

# def bar(self, message):

# # super(FooChild, self).bar(message)

# print('Child bar fuction')

# print(self.parent)

if \_\_name\_\_ == '\_\_main\_\_':

fooChild = FooChild()

fooChild.bar('HelloWorld')

[复制代码](javascript:void(0);)

**61. 是否使用过functools中的函数？其作用是什么？**

用于修复装饰器

[复制代码](javascript:void(0);)

import functools

def deco(func):

@functools.wraps(func) # 加在最内层函数正上方

def wrapper(\*args, \*\*kwargs):

return func(\*args, \*\*kwargs)

return wrapper

@deco

def index():

'''哈哈哈哈'''

x = 10

print('from index')

print(index.\_\_name\_\_)

print(index.\_\_doc\_\_)

# 加@functools.wraps

# index

# 哈哈哈哈

# 不加@functools.wraps

# wrapper

# None

[复制代码](javascript:void(0);)

**62. 列举面向对象中带双下划线的特殊方法，如：\_\_new\_\_、\_\_init\_\_**

* \_\_new\_\_：生成实例
* \_\_init\_\_：生成实例的属性
* \_\_call\_\_：实例对象加( )会执行def \_\_call\_\_:... 方法里边的内容。

\_\_del\_\_：析构方法，当对象在内存中被释放时，自动触发执行。如当 del obj 或者应用程序运行完毕时，执行该方法里边的内容。

\_\_enter\_\_和\_\_exit\_\_：出现with语句,对象的\_\_enter\_\_被触发,有返回值则赋值给as声明的变量；with中代码块执行完毕时执行\_\_exit\_\_里边的内容。

\_\_module\_\_：表示当前操作的对象在那个模块   obj.\_\_module\_\_  
\_\_class\_\_ ：表示当前操作的对象的类是什么     obj.\_\_class\_\_

\_\_doc\_\_：类的描述信息，该描述信息无法被继承

\_\_str\_\_：改变对象的字符串显示 print函数 --->obj.\_\_str\_\_()  
\_\_repr\_\_：改变对象的字符串显示 交互式解释器 --->obj.\_\_repr\_\_()  
\_\_format\_\_：自定制格式化字符串

\_\_slots\_\_:一个类变量 用来限制实例可以添加的属性的数量和类型

\_\_setitem\_\_,\_\_getitem,\_\_delitem\_\_:

[复制代码](javascript:void(0);)

class Foo:

def \_\_init\_\_(self,name):

self.name=name

def \_\_getitem\_\_(self, item):

print(self.\_\_dict\_\_[item])

def \_\_setitem\_\_(self, key, value):

self.\_\_dict\_\_[key]=value

def \_\_delitem\_\_(self, key):

print('del obj[key]时,我执行')

self.\_\_dict\_\_.pop(key)

def \_\_delattr\_\_(self, item):

print('del obj.key时,我执行')

self.\_\_dict\_\_.pop(item)

f1=Foo('sb')

f1['age']=18

f1['age1']=19

del f1.age1

del f1['age']

f1['name']='alex'

print(f1.\_\_dict\_\_)

[复制代码](javascript:void(0);)

\_\_get\_\_():调用一个属性时,触发  
\_\_set\_\_():为一个属性赋值时,触发  
\_\_delete\_\_():采用del删除属性时,触发

\_\_setattr\_\_,\_\_delattr\_\_,\_\_getattr\_\_ :

**63. 如何判断是函数还是方法？**

看他的调用者是谁，如果是类，就需要传入一个参数self的值，这时他就是一个函数，

如果调用者是对象，就不需要给self传入参数值，这时他就是一个方法

print(isinstance(obj.func, FunctionType))   # False

print(isinstance(obj.func, MethodType))    # True

[复制代码](javascript:void(0);)

class Foo(object):

def \_\_init\_\_(self):

self.name = 'lcg'

def func(self):

print(self.name)

obj = Foo()

print(obj.func) # <bound method Foo.func of <\_\_main\_\_.Foo object at 0x000001ABC0F15F98>>

print(Foo.func) # <function Foo.func at 0x000001ABC1F45BF8>

# ------------------------FunctionType, MethodType------------#

from types import FunctionType, MethodType

obj = Foo()

print(isinstance(obj.func, FunctionType)) # False

print(isinstance(obj.func, MethodType)) # True

print(isinstance(Foo.func, FunctionType)) # True

print(isinstance(Foo.func, MethodType)) # False

# ------------------------------------------------------------#

obj = Foo()

Foo.func(obj) # lcg

obj = Foo()

obj.func() # lcg

"""

注意：

方法，无需传入self参数

函数，必须手动传入self参数

"""

[复制代码](javascript:void(0);)

**64. 静态方法和类方法区别？**

尽管 classmethod 和 staticmethod 非常相似，但在用法上依然有一些明显的区别。classmethod 必须有一个指向类对象的引用作为第一个参数，而 staticmethod 可以没有任何参数。

举个栗子:

[复制代码](javascript:void(0);)

class Num:

# 普通方法：能用Num调用而不能用实例化对象调用

def one():

print ('1')

# 实例方法：能用实例化对象调用而不能用Num调用

def two(self):

print ('2')

# 静态方法：能用Num和实例化对象调用

@staticmethod

def three():

print ('3')

# 类方法：第一个参数cls长什么样不重要，都是指Num类本身，调用时将Num类作为对象隐式地传入方法

@classmethod

def go(cls):

cls.three()

Num.one() #1

#Num.two() #TypeError: two() missing 1 required positional argument: 'self'

Num.three() #3

Num.go() #3

i=Num()

#i.one() #TypeError: one() takes 0 positional arguments but 1 was given

i.two() #2

i.three() #3

i.go() #3

[复制代码](javascript:void(0);)

**65. 列举面向对象中的特殊成员以及应用场景**

[复制代码](javascript:void(0);)

\_\_call\_\_

\_\_new\_\_

\_\_init\_\_

\_\_doc\_\_

\_\_class\_\_

\_\_del\_\_

\_\_dict\_\_

\_\_str\_\_

在falsk源码用到......

[复制代码](javascript:void(0);)

**66. 1、2、3、4、5 能组成多少个互不相同且无重复的三位数**

60个

题意理解：组成后的数值不相同，且组合的三个位数之间数字不重复。

使用python内置的排列组合函数（不放回抽样排列）

product 笛卡尔积　　（有放回抽样排列）

permutations 排列　　（不放回抽样排列）

combinations 组合,没有重复　　（不放回抽样组合）

combinations\_with\_replacement 组合,有重复　　（有放回抽样组合）

import itertools

print(len(list(itertools.permutations('12345', 3)))) # 60

**67. 什么是反射？以及应⽤用场景？**

反射的核心本质就是以字符串的形式去导入个模块，利用字符串的形式去执行函数。

Django中的 CBV就是基于反射实现的。

**68. metaclass作用？以及应用场景？**

metaclass用来指定类是由谁创建的。

类的metaclass 默认是type。我们也可以指定类的metaclass值。在python3中：

[复制代码](javascript:void(0);)

class MyType(type):

def \_\_call\_\_(self, \*args, \*\*kwargs):

return 'MyType'

class Foo(object, metaclass=MyType):

def \_\_init\_\_(self):

return 'init'

def \_\_new\_\_(cls, \*args, \*\*kwargs):

return cls.\_\_init\_\_(cls)

def \_\_call\_\_(self, \*args, \*\*kwargs):

return 'call'

obj = Foo()

print(obj) # MyType

[复制代码](javascript:void(0);)



**69. 用尽量多的方法实现单例模式。**

[复制代码](javascript:void(0);)

1：使用模块

Python的模块就是天然的单例模式。

因为模块在第一次导入时，会生成 .pyc 文件，当第二次导入时，就会直接加载 .pyc 文件，而不会再次执行模块代码。

因此，我们只需把相关的函数和数据定义在一个模块中，就可以获得一个单例对象了。

例如：

class V1(object):

def foo(self)

pass

V1 = V1()

将上面代码保存在文件test.py,要使用时，直接在其他文件中导入此文件中的对象，这个对象既是单例模式的对象

如：from a import V1

2：使用装饰器

def Singleton(cls):

\_instance = {}

def \_singleton(\*args, \*\*kargs):

if cls not in \_instance:

\_instance[cls] = cls(\*args, \*\*kargs)

return \_instance[cls]

return \_singleton

@Singleton

class A(object):

a = 1

def \_\_init\_\_(self, x=0):

self.x = x

a1 = A(2)

a2 = A(3)

3：使用类

4：基于\_\_new\_\_方法实现

当我们实例化一个对象时，是先执行了类的\_\_new\_\_方法

当：（我们没写时，默认调用object.\_\_new\_\_），实例化对象；然后再执行类的\_\_init\_\_方法，对这个对象进行初始化，所有我们可以基于这个，实现单例模式

[复制代码](javascript:void(0);)

**70. 装饰器器的写法以及应用场景。**

含义：装饰器本质就是函数，为其他函数添加附加功能

原则：

不修改被修饰函数的代码

不修改被修饰函数的调用方式

应用场景：

无参装饰器在用户登录 认证中常见

有参装饰器在flask的路由系统中见到过

[复制代码](javascript:void(0);)

import functools

def wrapper(func):

@functools.wraps(func)

def inner(\*args, \*\*kwargs):

print('我是装饰器')

return func

return inner

@wrapper

def index():

print('我是被装饰函数')

return None

index()

# 应用场景

- 高阶函数

- 闭包

- 装饰器

- functools.wraps(func)

[复制代码](javascript:void(0);)

**71. 异常处理写法以及如何主动跑出异常（应用场景）**

[复制代码](javascript:void(0);)

# 触发异常

def temp\_convert(var):

try:

return int(var)

except ValueError as Argument:

print ("参数没有包含数字%s"%Argument)

# 调用函数

temp\_convert("xyz")

# 以10为基数的int()的无效文字:“xyz”

----------------------------------------------------------------------------

# raise语法

#raise [Exception [, args [, traceback]]]

# 语句中 Exception 是异常的类型，args 是自已提供的异常参数。

class Networkerror(RuntimeError):

def \_\_init\_\_(self, arg):

self.args = arg

try:

raise Networkerror("Bad hostname")

except Networkerror as e:

print(e.args)

[复制代码](javascript:void(0);)

**72、什么是面向对象的mro**

mro就是方法解析顺序

### 73. isinstance作用以及应用场景？

isinstance(对象，类)  判断这个对象是不是这个类或者这个类的子类的实例化

[复制代码](javascript:void(0);)

# # 判断a 属不属于A这个类（可以判断到祖宗类）

class A:

pass

class B(A):

pass

a = A()

b = B()

print(isinstance(b,A)) # ===> True 判断到祖宗类

# 任何与object都是True,内部都继承object

class A:pass

a = A() # 实例化

print(isinstance(a,object)) # True

[复制代码](javascript:void(0);)

应用场景：rest framework 认证的流程

scrapy-redis

### 74. 写代码并实现

Given an array of integers, return indices of the two numbers such that they add up to a  
specific target.You may assume that each input would have exactly one solution, and you may  
not use the same element twice.  
Example:  
Given nums = [2, 7, 11, 15], target = 9,  
Because nums[0] + nums[1] = 2 + 7 = 9,  
return [0, 1]

### 75. json序列化时，可以处理的数据类型有哪些？如何定制支持datetime类型？

### 76. json序列化时，默认遇到中文会转换成unicode，如果想要保留中文怎么办？

在序列化时，中文汉字总是被转换为unicode码，在dumps函数中添加参数ensure\_ascii=False即可解决。

### 77. 什么是断言？应用场景？

[复制代码](javascript:void(0);)

assert 是的作用？断言

条件成立（布尔值为True）则继续往下，否则跑出异常，一般用于：满足某个条件之后，才能执行，否则应该跑出异常。

写API的时候，继承GenericAPIView

class GenericAPIView(views.APIView):

"""

Base class for all other generic views.

"""

# You'll need to either set these attributes,

# or override `get\_queryset()`/`get\_serializer\_class()`.

# If you are overriding a view method, it is important that you call

# `get\_queryset()` instead of accessing the `queryset` property directly,

# as `queryset` will get evaluated only once, and those results are cached

# for all subsequent requests.

queryset = None

serializer\_class = None

# If you want to use object lookups other than pk, set 'lookup\_field'.

# For more complex lookup requirements override `get\_object()`.

lookup\_field = 'pk'

lookup\_url\_kwarg = None

# The filter backend classes to use for queryset filtering

filter\_backends = api\_settings.DEFAULT\_FILTER\_BACKENDS

# The style to use for queryset pagination.

pagination\_class = api\_settings.DEFAULT\_PAGINATION\_CLASS

def get\_queryset(self):

assert self.queryset is not None, (

"'%s' should either include a `queryset` attribute, "

"or override the `get\_queryset()` method."

% self.\_\_class\_\_.\_\_name\_\_

)

queryset = self.queryset

if isinstance(queryset, QuerySet):

# Ensure queryset is re-evaluated on each request.

queryset = queryset.all()

return queryset

[复制代码](javascript:void(0);)

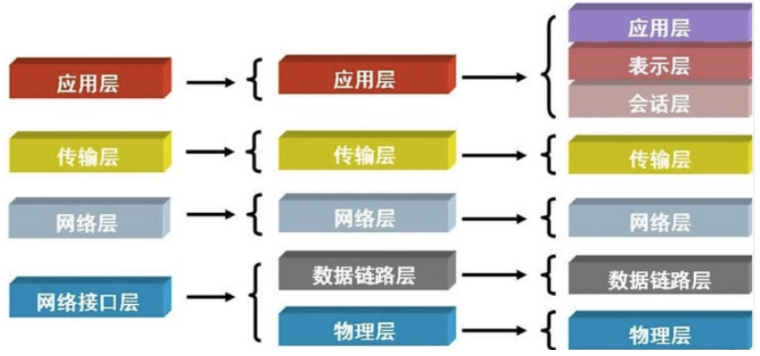
### 78. 有用过with statement吗？它的好处是什么？

### 79. 使用代码实现查看列举目录下的所有文件。

### 80. 简述 yield和yield from关键字

## 第二部分 网络编程和并发

### 81. 简述 OSI 七层协议。



物理层：主要是基于电器特性发送高低电压（电信号），高电压对应数字1，低电压对应数字0

数据链路层：定义了电信号的分组方式

网路层：引入一套新的地址用来区分不同的广播域/子网，这套地址即网络地址

传输层：建立端口到端口的通信

会话层：建立客户端与服务端连接

表示层：对来自应用层的命令和数据进行解释，按照一定格式传给会话层。如编码、数据格式转换、加密解密、压缩解压  
应用层：规定应用程序的数据格式

### 82. 什么是C/S和B/S架构？

c/s架构，就是client（客户端）与server（服务端）即：客户端与服务端的架构。

b/s架构，就是brosver（浏览器端）与sever（服务端）即：浏览器端与服务端架构

优点：统一了所有应用程序的入口、方便、轻量级

### 83. 简述 三次握手、四次挥手的流程。

[复制代码](javascript:void(0);)

三次握手：

第一次握手

1：客户端先向服务端发起一次询问建立连接的请求，并随机生成一个值作为标识

第二次握手

2：服务端向客户端先回应第一个标识，再重新发一个确认标识

第三次握手

3：客户端确认标识，建立连接，开始传输数据

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

四次挥手 ---> 断开连接

第一次挥手

客户端向服务端发起请求断开连接的请求

第二次挥手

服务端向客户端确认请求

第三次挥手

服务端向客户端发起断开连接请求

第四次挥手

客户端向服务端确认断开请求

[复制代码](javascript:void(0);)

### 84. TCP和UDP的区别？

TCP/UDP区别

TCP协议是面向连接，保证高可靠性传输层协议

UDP：数据丢失，无秩序的传输层协议（qq基于udp协议）

### 85. 为何基于tcp协议的通信比基于udp协议的通信更可靠？

tcp：可靠，因为只要对方回了确认收到信息，才发下一个，如果没收到确认信息就重发

UDP：不可靠，它是一直发数据，不需要对方回应

流式协议： TCP协议，可靠传输

数据报协议: UDP协议，不可传输

### 86. 什么是socket？简述基于tcp协议的套接字通信流程。

[复制代码](javascript:void(0);)

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。  
在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，  
对用户来说，一组简单的接口就是全部。

服务端：  
创建socket对象，  
绑定ip端口bind(),   
设置最大链接数listen(),   
accept()与客户端的connect()创建双向管道，等到联接，   
send(), recv(), 收发数据  
close()

客户端：  
创建socket对象，  
connect()与服务端accept()创建双向管道 ,   
 send(),  
 recv(),  
close()

[复制代码](javascript:void(0);)

### 87. 什么是粘包？ socket 中造成粘包的原因是什什么？ 哪些情况会发生粘包现象？

[复制代码](javascript:void(0);)

**只有TCP有粘包现象，UDP永远不会粘包  
粘包**：在获取数据时,出现数据的内容不是本应该接收的数据,如:对方第一次发送hello,第二次发送world,  
　　我方接收时,应该收两次,一次是hello,一次是world,但事实上是一次收到helloworld,一次收到空,这种现象叫粘包

**原因**

粘包问题主要还是因为接收方不知道消息之间的界限，不知道一次性提取多少字节的数据所造成的。

**什么情况会发生**：

1、发送端需要等缓冲区满才发送出去，造成粘包（发送数据时间间隔很短，数据了很小，会合到一起，产生粘包）

2、接收方不及时接收缓冲区的包，造成多个包接收（客户端发送了一段数据，服务端只收了一小部分，  
服务端下次再收的时候还是从缓冲区拿上次遗留的数据，产生粘包）

[复制代码](javascript:void(0);)

### 88. IO多路复的作用？

socketserver，多个客户端连接，单线程下实现并发效果，就叫多路复用。

与多进程和多线程技术相比，I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

### 89.select、poll、epoll 模型的区别？（属于多路复用IO的模型）

[复制代码](javascript:void(0);)

都是i/o多路复用的机制，监视多个socket是否发生变化，本质上都是同步i/o

select,poll实现需要自己不断轮询所有监测对象，直到对象发生变化，在这个阶段中，  
可能要睡眠和唤醒多次交替，而epoll也需要调用epoll\_wait不断轮询就绪链表，但是当对象发生变化时，  
会调用回调函数，将变化的对象放入就绪链接表中，并唤醒在epoll\_wait中进入睡眠的进程。  
虽然都会睡眠和唤醒，但是select和poll在被唤醒的时候要遍历整个监测对象集合，  
而epoll只要判断就绪链表是否为空即可，节省了大量cpu的时间

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

select、poll、epoll都是IO多路复用的机制，但select，poll，epoll本质上都是同步I/O，

因为他们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的.

FD(文件描述符)

select模型

优点：

1：可移植性好，在某些Unix系统不支持poll()

2：对于超时值提供了更好的精度：微妙，而poll是毫秒

缺点：

1:最大并发数限制，因为一个进程所打开的 FD （文件描述符）是有限制的，由 FD\_SETSIZE 设置，默认值是 1024/2048 ，因此 Select 模型的最大并发数就被相应限制了。

2：效率问题，select每次调用都会线性扫描全部的FD集合，所以将FD\_SETSIZE 改大，会越慢

3：需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大。

poll本质上和select 没有区别，它将用户传入的数组拷贝到内核空间，

它没有最大连接数的限制，原因是它基于链表来存储的但是同样有一个缺点：

大量的fd的数组被整体复制于用户态和内核地址空间，而不管这样的复制是不是有意义

[复制代码](javascript:void(0);)

### 90. 什么是防火墙以及作用？

[复制代码](javascript:void(0);)

防火墙是一个分离器、一个限制器，也是一个分析器，有效地监控了内部网和Internet之间的任何活动，保证了内部网络的安全

作用

防火墙是网络安全的屏障

可以强化网络安全策略

对网络存取和访问进行监控审计

防止内部信息的外泄

除了安全作用，防火墙还支持具有Internet服务特性的企业内部网络技术体系VPN（虚拟专用网）。

[复制代码](javascript:void(0);)

### 91. 简述 进程、线程、协程的区别 以及应用场景？

[复制代码](javascript:void(0);)

线程是指进程内的一个执行单元，

# 进程

进程拥有自己独立的堆和栈，既不共享堆，亦不共享栈，进程由操作系统调度。

# 线程

线程拥有自己独立的栈和共享的堆，共享堆，不共享栈，线程亦由操作系统调度

# 协程和线程

协程避免了无意义的调度，由此可以提高性能；但同时协程也失去了线程使用多CPU的能力

进程与线程的区别

（1）地址空间：线程是进程内的一个执行单位，进程内至少有一个线程，他们共享进程的地址空间，而进程有自己独立的地址空间

（2）资源拥有：进程是资源分配和拥有的单位，同一个进程内线程共享进程的资源

（3）线程是处理器调度的基本单位，但进程不是

（4）二者均可并发执行

（5）每个独立的线程有一个程序运行的入口

协程与线程

（1）一个线程可以有多个协程，一个进程也可以单独拥有多个协程，这样Python中则能使用多核CPU

（2）线程进程都是同步机制，而协程是异步

（3）协程能保留上一次调用时的状态

[复制代码](javascript:void(0);)

### 92. GIL锁是什么？

GIL本质就是一把互斥锁，既然是互斥锁，所有互斥锁的本质都一样，都是将并发运行变成串行，以此来控制同一时间内共享数据只能被一个任务所修改，进而保证数据安全。

GIL保护的是解释器级的数据，保护用户自己的数据则需要自己加锁处理

[复制代码](javascript:void(0);)

应用（总结）：

多线程用于IO密集型，如socket，爬虫，web

多进程用于计算密集型，如金融分析

1. 每个cpython进程内都有一个GIL

2. GIL导致同一进程内多个进程同一时间只能有一个运行

3. 之所以有GIL，是因为Cpython的内存管理不是线程安全的

4. 对于计算密集型用多进程，多IO密集型用多线程

[复制代码](javascript:void(0);)

### 93. Python中如何使用线程池和进程池？

### 94. threading.local的作用？

实现线程局部变量的传递。

ThreadLocal 最常用的地方：

为每个线程绑定一个资源（数据库连接，HTTP请求，用户身份信息等），这样一个线程的所有调用到的处理函数都可以非常方便地访问这些资源。

### 95. 进程之间如何进行通信？

### 96. 什么是并发和并行？

# 并发：同一时刻只能处理一个任务，但一个时段内可以对多个任务进行交替处理(一个处理器同时处理多个任务)

# 并行：同一时刻可以处理多个任务(多个处理器或者是多核的处理器同时处理多个不同的任务)

# 类比：并发是一个人同时吃三个馒头，而并行是三个人同时吃三个馒头。

### 97. 进程锁和线程锁的作用？

### 98. 解释什么是异步非阻塞？

[复制代码](javascript:void(0);)

非阻塞：不等待

即：遇到IO阻塞不等待(setblooking=False),（可能会报错->捕捉异常）

- sk=socket.socket()

- sk.setblooking(False)

异步：回调，当达到某个指定的状态之后，自动调用特定函数

实例

nb\_async.py 实现异步非阻塞的模块

[复制代码](javascript:void(0);)

异步体现在回调上，回调就是有消息返回时告知一声儿进程进行处理。非阻塞就是不等待，不需要进程等待下去，继续执行其他操作，不管其他进程的状态。

### 99. 路由器和交换机的区别

[复制代码](javascript:void(0);)

1：交换机：是负责内网里面的数据传递（arp协议）根据MAC地址寻址

路由器：在网络层，路由器根据路由表，寻找该ip的网段

2：路由器可以处理TCP/IP协议

3：路由器可以把一个IP分配给很多个主机使用，这些主机对外只表现出一个IP。

交换机可以把很多主机连起来，这些主机对外各有各的IP。

4：交换机是做端口扩展的，也就是让局域网可以连进来更多的电脑。

路由器是用来做网络连接，也就是；连接不同的网络

[复制代码](javascript:void(0);)

### 100.什么是域名解析？

在互联网上，所有的地址都是ip地址，现阶段主要是IPv4（比如：110.110.110.110）。

但是这些ip地址太难记了，所以就出现了域名（比如http://baidu.com）。

域名解析就是将域名，转换为ip地址的这样一种行为。

### 101.如何修改本地hosts件？

[复制代码](javascript:void(0);)

Hosts是一个没有扩展名的系统文件，可以用记事本等工具打开，其作用就是将一些常用的网址域名与其对应的IP地址建立一个关联“数据库”，  
当用户在浏览器中输入一个需要登录的网址时，系统会首先自动从Hosts文件中寻找对应的IP地址，  
一旦找到，系统会立即打开对应网页，如果没有找到，则系统会再将网址提交DNS域名解析服务器进行IP地址的解析。

浏览器访问网站，要首先通过DNS服务器把要访问的网站域名解析成一个唯一的IP地址，之后，浏览器才能对此网站进行定位并且访问其数据。

文件路径：C:\WINDOWS\system32\drivers\etc。

将127.0.0.1 www.163.com 添加在最下面

修改后用浏览器访问“www.163.com”会被解析到127.0.0.1，导致无法显示该网页。

[复制代码](javascript:void(0);)

### 102.生产者消费者模型应用场景及优势？

[复制代码](javascript:void(0);)

生产者与消费者模式是通过一个容器来解决生产者与消费者的强耦合关系，生产者与消费者之间不直接进行通讯，  
而是利用阻塞队列来进行通讯，生产者生成数据后直接丢给阻塞队列，消费者需要数据则从阻塞队列获取，  
实际应用中，生产者与消费者模式则主要解决生产者与消费者生产与消费的速率不一致的问题，达到平衡生产者与消费者的处理能力，而阻塞队列则相当于缓冲区。

应用场景：用户提交订单，订单进入引擎的阻塞队列中，由专门的线程从阻塞队列中获取数据并处理

优势：

1；解耦

假设生产者和消费者分别是两个类。如果让生产者直接调用消费者的某个方法，那么生产者对于消费者就会产生依赖（也就是耦合）。

将来如果消费者的代码发生变化，可能会影响到生产者。而如果两者都依赖于某个缓冲区，两者之间不直接依赖，耦合也就相应降低了。

2：支持并发

生产者直接调用消费者的某个方法，还有另一个弊端。由于函数调用是同步的（或者叫阻塞的），在消费者的方法没有返回之前，生产者只能一直等着

而使用这个模型，生产者把制造出来的数据只需要放在缓冲区即可，不需要等待消费者来取

3：支持忙闲不均

缓冲区还有另一个好处。如果制造数据的速度时快时慢，缓冲区的好处就体现出来了。

当数据制造快的时候，消费者来不及处理，未处理的数据可以暂时存在缓冲区中。等生产者的制造速度慢下来，消费者再慢慢处理掉。

[复制代码](javascript:void(0);)

### 103.什么是cdn？

目的是使用户可以就近到服务器取得所需内容，解决 Internet网络拥挤的状况，提高用户访问网站的响应速度。

cdn 即内容分发网络

### 104.LVS是什么及作用？

[复制代码](javascript:void(0);)

LVS ：Linux虚拟服务器

作用：LVS主要用于多服务器的负载均衡。

它工作在网络层，可以实现高性能，高可用的服务器集群技术。  
它廉价，可把许多低性能的服务器组合在一起形成一个超级服务器。  
它易用，配置非常简单，且有多种负载均衡的方法。  
它稳定可靠，即使在集群的服务器中某台服务器无法正常工作，也不影响整体效果。另外可扩展性也非常好。

[复制代码](javascript:void(0);)

### 105.Nginx是什么及作用？

### 106.keepalived是什么及作用?

### 107.haproxy是什么以及作用？

108.什么是负载均衡？

109.什么是rpc及应用场景？

110.简述 asynio模块的作用和应用场景。

111.简述 gevent模块的作用和应用场景。

112.twisted框架的使用和应用

[回到顶部](http://www.cnblogs.com/jassin-du/articles/8985293.html#_labelTop)

## 数据库和缓存（46题）

### 113.列举常见的关系型数据库和非关系型都有那些？

[复制代码](javascript:void(0);)

关系型数据库(需要有表结构)

mysql、oracle 、 spl、server、db2、sybase

非关系型数据库（是以key-value存储的，没有表结构）（NoSQL）

MongoDB

MongoDB 是一个高性能，开源，无模式的文档型数据库，开发语言是C++。它在许多场景下可用于替代传统的关系型数据库或键/值存储方式。

Redis

Redis 是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。目前由VMware主持开发工作。

[复制代码](javascript:void(0);)

### 114.MySQL常见数据库引擎及比较？

[复制代码](javascript:void(0);)

InnoDB

支持事务

支持表锁、行锁（for update）

表锁：select \* from tb for update

行锁：select id,name from tb where id=2 for update

myisam

查询速度快

全文索引

支持表锁

表锁：select \* from tb for update

NDB

高可用、 高性能、高可扩展性的数据库集群系统

Memory

默认使用的是哈希索引

[复制代码](javascript:void(0);)

### 115.简述数据库三大范式？

[复制代码](javascript:void(0);)

数据库的三大特性：

'实体':表

'属性'：表中的数据(字段)

'关系'：表与表之间的关系

----------------------------------------------------

# 数据库设计三大范式：

1：确保每列保持原子性（即数据库表中的所有字段值是不可分解的原子值）

2：确保表中的每列都是和主键相关（表中只能保存一种数据，不可以把多种数据保存在同一张表中）--->完全属于当前表的数据

3：确保每列都和主键直接相关，而不是间接相关（在一个数据库表中保存的数据只能与主键相关）----> 消除传递依赖（间接）

比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。  
而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段。

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

数据库五大约束'

1.primary KEY:设置主键约束；

2.UNIQUE：设置唯一性约束，不能有重复值；

3.DEFAULT 默认值约束

4.NOT NULL：设置非空约束，该字段不能为空；

5.FOREIGN key :设置外键约束。

[复制代码](javascript:void(0);)

### 116、什么是事务？MySQL如何支持事务？

[复制代码](javascript:void(0);)

事务用于将某些操作的多个SQL作为原子性操作，一旦有某一个出现错误，即可回滚到原来的状态，从而保证数据库数据完整性。

事务的特性：

原子性: 确保工作单元内的所有操作都成功完成，否则事务将被中止在故障点，和以前的操作将回滚到以前的状态。

一致性: 确保数据库正确地改变状态后，成功提交的事务。

隔离性: 使事务操作彼此独立的和透明的。

持久性: 确保提交的事务的结果或效果的系统出现故障的情况下仍然存在。

Mysql实现事务

InnoDB支持事务，MyISAM不支持

# 启动事务：

# start transaction；

# update from account set money=money-100 where name='a';

# update from account set money=money+100 where name='b';

# commit；

'start transaction 手动开启事务，commit 手动关闭事务'

[复制代码](javascript:void(0);)

### 117.简述数据库设计中一对多和多对多的应用场景？

FK(一对多)

下拉框里面的数据就需要用FK关联另一张表

M2M（多对多）

多选的下拉框，或者checkbox

### 118.如何基于数据库实现商城商品计数器？

### 119.常见SQL（必备）

[复制代码](javascript:void(0);)

group by 分组对聚合的条件进行筛选需要通过havhing

SQL的left join 、right join、inner join之间的区别

left join (左连接) 返回包括左表中的所有记录和右表中联结字段相等的记录

right join(右连接) 返回包括右表中的所有记录1和左表中联结字段相等的记录

inner join（内连接）： 只返回两个表中联结字段相等的行

[复制代码](javascript:void(0);)

https://www.cnblogs.com/wupeiqi/articles/5729934.html

### 120.简述触发器、函数、视图、存储过程？

[复制代码](javascript:void(0);)

触发器：

对数据库某张表的增加、删除，修改前后定义一些操作

函数：(触发函数是通过select)

聚合函数：max/sum/min/avg

时间格式化：date\_format

字符串拼接：concat

存储过程：

将SQL语句保存到数据库中，并命名，以后在代码调用时，直接调用名称即可

参数类型：

　　in 只将参数传进去

　　out 只拿结果

　　inout 既可以传，可以取

函数与存储过程区别：

本质上没区别。只是函数有如：只能返回一个变量的限制。而存储过程可以返回多个。而函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行。

视图：

视图是一个虚拟表，不是真实存在的（只能查，不能改）

[复制代码](javascript:void(0);)

### 121.MySQL索引种类

[复制代码](javascript:void(0);)

单列

功能

普通索引：加速查找

唯一索引：加速查找 + 约束：不能重复（只能有一个空，不然就重复了）

主键（primay key）：加速查找 + 约束：不能重复 + 不能为空

多列

　　联合索引（多个列创建索引）-----> 相当于单列的普通索引

　　联合唯一索引 -----> 相当于单列的唯一索引

　　ps：联合索引的特点：遵循最左前缀的规则

其他词语：

·· - 索引合并，利用多个单例索引查询；（例如在数据库查用户名和密码，分别给用户名和密码建立索引）

- 覆盖索引，在索引表中就能将想要的数据查询到；

[复制代码](javascript:void(0);)

### 122.索引在什么情况下遵循最左前缀的规则？

联合索引

### 123.主键和外键的区别？

主键是能确定一条记录的唯一标示。例如，身份证证号

外键：用于与另一张表的关联，是能确定另一张表记录的字段，用于保持数据的一致性

|  |  |  |
| --- | --- | --- |
|  | 主键 | 外键 |
| 定义 | 唯一标识一条记录，不能有重复的，不允许为空 | 表的外键是另一张表的主键，外键可以有重复的，可以为空 |
| 作用 | 用来保证数据完整性 | 用来与其他表建立联系的 |
| 个数 | 主键只能有一个 | 一个表可以有多个外键 |

### 124.MySQL常见的函数？

[复制代码](javascript:void(0);)

聚合函数

max/sum/min/avg

时间格式化

date\_format

字符串拼接

concat（当拼接了null，则返回null）

截取字符串

substring

返回字节个数

length

[复制代码](javascript:void(0);)

### 125.列举 创建索引但是无法命中索引的8种情况。

[复制代码](javascript:void(0);)

1.- like '%xx'

select \* from tb1 where name like '%cn';

2.- 使用函数

select \* from tb1 where reverse(name) = 'wupeiqi';

3.- or

select \* from tb1 where nid = 1 or email = 'seven@live.com';

特别的：当or条件中有未建立索引的列才失效，以下会走索引

select \* from tb1 where nid = 1 or name = 'seven';

select \* from tb1 where nid = 1 or email = 'seven@live.com' and name = 'alex'

4.- 类型不一致

如果列是字符串类型，传入条件是必须用引号引起来，不然...

select \* from tb1 where name = 999;

5.- !=

select \* from tb1 where name != 'alex'

特别的：如果是主键，则还是会走索引

select \* from tb1 where nid != 123

6.- >

select \* from tb1 where name > 'alex'

特别的：如果是主键或索引是整数类型，则还是会走索引

select \* from tb1 where nid > 123

select \* from tb1 where num > 123

7.- order by

select email from tb1 order by name desc;

当根据索引排序时候，选择的映射如果不是索引，则不走索引

特别的：如果对主键排序，则还是走索引：

select \* from tb1 order by nid desc;

8.- 组合索引最左前缀

如果组合索引为：(name,email)

name and email -- 使用索引

name -- 使用索引

email -- 不使用索引

[复制代码](javascript:void(0);)

### 126.如何开启慢日志查询？

[复制代码](javascript:void(0);)

修改配置文件

slow\_query\_log = OFF 是否开启慢日志记录

long\_query\_time = 2 时间限制，超过此时间，则记录

slow\_query\_log\_file = /usr/slow.log 日志文件

log\_queries\_not\_using\_indexes = OFF 为使用索引的搜索是否记录

下面是开启

slow\_query\_log = ON

long\_query\_time = 2

log\_queries\_not\_using\_indexes = OFF

log\_queries\_not\_using\_indexes = ON

注：查看当前配置信息：

　　 show variables like '%query%'

修改当前配置：

　　　　set global 变量名 = 值

[复制代码](javascript:void(0);)

### 127.数据库导入导出命令（结构+数据）？

[复制代码](javascript:void(0);)

导出现有数据库数据：（当有提示出入密码。-p就不用加密码）

mysqldump -u用户名 -p密码 数据库名称 >导出文件路径 # 结构+数据

mysqldump -u用户名 -p密码 -d 数据库名称 >导出文件 路径 # 结构

导入现有数据库数据：

mysqldump -uroot -p密码 数据库名称 < 文件路径

[复制代码](javascript:void(0);)

### 128.数据库优化方案？

[复制代码](javascript:void(0);)

1、创建数据表时把固定长度的放在前面（）

2、将固定数据放入内存： 例如：choice字段 （django中有用到，数字1、2、3…… 对应相应内容）

3、char 和 varchar 的区别(char可变, varchar不可变 )

4、联合索引遵循最左前缀(从最左侧开始检索)

5、避免使用 select \*

6、读写分离

　　　　- 实现：两台服务器同步数据

　　　　- 利用数据库的主从分离：主，用于删除、修改、更新；从，用于查；

读写分离:利用数据库的主从进行分离：主，用于删除、修改更新；从，用于查

7、分库

　　　　- 当数据库中的表太多，将某些表分到不同的数据库，例如：1W张表时

　　　　- 代价：连表查询

8、分表

　　　　- 水平分表：将某些列拆分到另外一张表，例如：博客+博客详情

　　　　- 垂直分表：讲些历史信息分到另外一张表中，例如：支付宝账单

9、加缓存

　　　　- 利用redis、memcache （常用数据放到缓存里，提高取数据速度）

如果只想获取一条数据

- select \* from tb where name=‘alex’ limit 1

[复制代码](javascript:void(0);)

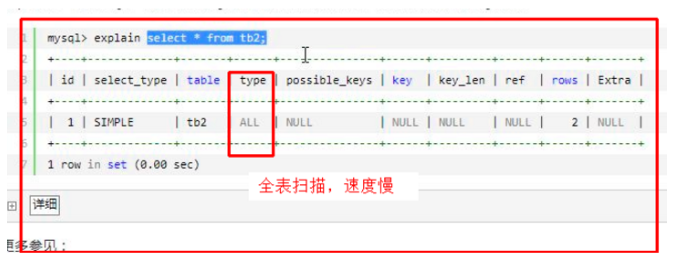
### 129.char和varchar的区别？

char 和 varchar 的区别(char可变, varchar不可变 )

### 130.简述MySQL的执行计划？

查看有没有命中索引，让数据库帮看看运行速度快不快

explain select \* from table;



当type为all时，是为全表索引

### 131.在对name做了唯一索引前提下，简述以下区别：

    select \* from tb where name = ‘Oldboy-Wupeiqi’     
        select \* from tb where name = ‘Oldboy-Wupeiqi’ limit 1

[复制代码](javascript:void(0);)

是这样的的，用where条件过滤出符合条件的数据的同时，进行计数，  
比如limit 1，那么在where过滤出第1条数据后，他就会直接把结果select出来返回给你，整个过程就结束了。

没做唯一索引的话,前者查询会全表扫描,效率低些  
limit 1,只要找到对应一条数据,就不继续往下扫描.

然而 name 字段添加唯一索引了,加不加limit 1,意义都不大;

[复制代码](javascript:void(0);)

### 132.1000w条数据，使用limit offset 分页时，为什么越往后翻越慢？如何解决？

[复制代码](javascript:void(0);)

答案一：

先查主键，在分页。

select \* from tb where id in (

select id from tb where limit 10 offset 30

)

答案二：

按照也无需求是否可以设置只让用户看200页

答案三：

记录当前页 数据ID最大值和最小值

在翻页时，根据条件先进行筛选；筛选完毕之后，再根据limit offset 查询。

select \* from (select \* from tb where id > 22222222) as B limit 10 offset 0

如果用户自己修改页码，也可能导致慢；此时对url种的页码进行加密（rest framework ）

[复制代码](javascript:void(0);)

### 133.什么是索引合并？

[复制代码](javascript:void(0);)

1、索引合并是把几个索引的范围扫描合并成一个索引。

2、索引合并的时候，会对索引进行并集，交集或者先交集再并集操作，以便合并成一个索引。

3、这些需要合并的索引只能是一个表的。不能对多表进行索引合并。

简单的说，索引合并，让一条sql可以使用多个索引。对这些索引取交集，并集，或者先取交集再取并集。  
从而减少从数据表中取数据的次数，提高查询效率。

[复制代码](javascript:void(0);)

### 134.什么是覆盖索引？

在索引表中就能将想要的数据查询到

### 135.简述数据库读写分离？

- 实现：两台服务器同步数据

　　　　- 利用数据库的主从分离：主，用于删除、修改、更新；从，用于查；

[复制代码](javascript:void(0);)

方式一:是视图里面用using方式可以进行指定到哪个数据读写

from django.shortcuts import render,HttpResponse

from app01 import models

def index(request):

models.UserType.objects.using('db1').create(title='普通用户')

　　# 手动指定去某个数据库取数据

result = models.UserType.objects.all().using('db1')

print(result)

return HttpResponse('...')

方式二：写配置文件

class Router1:

　　# 指定到某个数据库取数据

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth models go to auth\_db.

"""

if model.\_meta.model\_name == 'usertype':

return 'db1'

else:

return 'default'

　　　# 指定到某个数据库存数据

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth models go to auth\_db.

"""

return 'default'

再写到配置

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

},

'db1': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

DATABASE\_ROUTERS = ['db\_router.Router1',]

[复制代码](javascript:void(0);)

### 136.简述数据库分库分表？（水平、垂直）

[复制代码](javascript:void(0);)

1、分库

当数据库中的表太多，将某些表分到不同数据库，例如：1W张表时

代价：连表查询跨数据库，代码变多

# 2、分表

水平分表：将某些列拆分到另一张表，例如：博客+博客详情

垂直分表：将某些历史信息，分到另外一张表中，例如：支付宝账单

[复制代码](javascript:void(0);)

### 137.redis和memcached比较？

[复制代码](javascript:void(0);)

区别

1：redis不仅支持简单的key\_value类型，还支持字典，字符串，列表，集合，有序集合类型

2：内存使用效率对比，使用简单的key-value存储的话，  
 Memcached的内存利用率更高而如果Redis采用hash结构来做key-value存储，由于其组合式的压缩，其内存利用率会高于Memcached。

3.性能对比：由于Redis只使用单核，而Memcached可以使用多核，.  
 所以平均每一个核上Redis在存储小数据时比Memcached性能更高。而在100k以上的数据中，Memcached性能要高于Redis，

4.Redis虽然是基于内存的存储系统，但是它本身是支持内存数据的持久化的，而且提供两种主要的持久化策略：RDB快照和AOF日志。  
 而memcached是不支持数据持久化操作的。

5.集群管理不同，Memcached本身并不支持分布式，因此只能在客户端通过像一致性哈希这样的分布式算法来实现Memcached的分布式存储。

[复制代码](javascript:void(0);)

### 138.redis中数据库默认是多少个db 及作用？

Redis默认支持16个数据库，可以通过配置databases来修改这一数字。客户端与Redis建立连接后会自动选择0号数据库，不过可以随时使用SELECT命令更换数据库

Redis支持多个数据库，并且每个数据库的数据是隔离的不能共享，并且基于单机才有，如果是集群就没有数据库的概念。

### 139.[python操作redis的模块？](https://www.cnblogs.com/Eva-J/p/5152841.html)

[复制代码](javascript:void(0);)

- 连接

- 直接连接：

import redis

r = redis.Redis(host='10.211.55.4', port=6379)

r.set('foo', 'Bar')

print r.get('foo')

- 连接池：

import redis

pool = redis.ConnectionPool(host='10.211.55.4', port=6379)

r = redis.Redis(connection\_pool=pool)

r.set('foo', 'Bar')

print r.get('foo')

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

- 如果一个列表在redis中保存了10w个值，我需要将所有值全部循环并显示，请问如何实现？

一个一个取值，列表没有iter方法，但能自定义

　　　　 def list\_scan\_iter(name,count=3):

start = 0

while True:

result = conn.lrange(name, start, start+count-1)

start += count

if not result:

break

for item in result:

yield item

for val in list\_scan\_iter('num\_list'):

print(val)

　　场景：投票系统，script-redis

[复制代码](javascript:void(0);)

### 141.redis如何实现主从复制？以及数据同步机制？

[复制代码](javascript:void(0);)

优势：

- 高可用

- 分担主压力

注意：

- slave设置只读

从的配置文件添加以下记录，即可：

slaveof 1.1.1.1 3306

[复制代码](javascript:void(0);)

### 142.redis中的sentinel的作用？

[复制代码](javascript:void(0);)

帮助我们自动在主从之间进行切换

检测主从中 主是否挂掉，且超过一半的sentinel检测到挂了之后才进行进行切换。

如果主修复好了，再次启动时候，会变成从。

启动主redis:

redis-server /etc/redis-6379.conf 启动主redis

redis-server /etc/redis-6380.conf 启动从redis

在linux中：

找到 /etc/redis-sentinel-8001.conf 配置文件，在内部：

- 哨兵的端口 port = 8001

- 主redis的IP，哨兵个数的一半/1

找到 /etc/redis-sentinel-8002.conf 配置文件，在内部：

- 哨兵的端口 port = 8002

- 主redis的IP, 1

启动两个哨兵

[复制代码](javascript:void(0);)

### 143.如何实现redis集群？

[复制代码](javascript:void(0);)

redis集群、分片、分布式redis

redis-py-cluster

集群方案：

- redis cluster 官方提供的集群方案。

- codis，豌豆荚技术团队。

- tweproxy，Twiter技术团队。

redis cluster的原理？

- 基于分片来完成。

- redis将所有能放置数据的地方创建了 16384 个哈希槽。

- 如果设置集群的话，就可以为每个实例分配哈希槽：

- 192.168.1.20【0-5000】

- 192.168.1.21【5001-10000】

- 192.168.1.22【10001-16384】

- 以后想要在redis中写值时，

set k1 123

将k1通过crc16的算法，将k1转换成一个数字。然后再将该数字和16384求余，如果得到的余数 3000，那么就将该值写入到 192.168.1.20 实例中。

[复制代码](javascript:void(0);)

### 144.redis中默认有多少个哈希槽？

16384

### 145.简述redis的有哪几种持久化策略及比较？

[复制代码](javascript:void(0);)

RDB：每隔一段时间对redis进行一次持久化。

- 缺点：数据不完整

- 优点：速度快

AOF：把所有命令保存起来，如果想到重新生成到redis，那么就要把命令重新执行一次。

- 缺点：速度慢，文件比较大

- 优点：数据完整

[复制代码](javascript:void(0);)

### 146.列举redis支持的过期策略。

[复制代码](javascript:void(0);)

voltile-lru： 从已设置过期时间的数据集（server.db[i].expires）中挑选最近频率最少数据淘汰

volatile-ttl： 从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru： 从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random： 从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

[复制代码](javascript:void(0);)

### 147.MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中都是热点数据？

[复制代码](javascript:void(0);)

相关知识：redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略（回收策略）。redis 提供 6种数据淘汰策略：

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

[复制代码](javascript:void(0);)

### 148.写代码，基于redis的列表实现 先进先出、后进先出队列、优先级队列。

[复制代码](javascript:void(0);)

参看script—redis源码

from scrapy.utils.reqser import request\_to\_dict, request\_from\_dict

from . import picklecompat

class Base(object):

"""Per-spider base queue class"""

def \_\_init\_\_(self, server, spider, key, serializer=None):

"""Initialize per-spider redis queue.

Parameters

----------

server : StrictRedis

Redis client instance.

spider : Spider

Scrapy spider instance.

key: str

Redis key where to put and get messages.

serializer : object

Serializer object with ``loads`` and ``dumps`` methods.

"""

if serializer is None:

# Backward compatibility.

# TODO: deprecate pickle.

serializer = picklecompat

if not hasattr(serializer, 'loads'):

raise TypeError("serializer does not implement 'loads' function: %r"

% serializer)

if not hasattr(serializer, 'dumps'):

raise TypeError("serializer '%s' does not implement 'dumps' function: %r"

% serializer)

self.server = server

self.spider = spider

self.key = key % {'spider': spider.name}

self.serializer = serializer

def \_encode\_request(self, request):

"""Encode a request object"""

obj = request\_to\_dict(request, self.spider)

return self.serializer.dumps(obj)

def \_decode\_request(self, encoded\_request):

"""Decode an request previously encoded"""

obj = self.serializer.loads(encoded\_request)

return request\_from\_dict(obj, self.spider)

def \_\_len\_\_(self):

"""Return the length of the queue"""

raise NotImplementedError

def push(self, request):

"""Push a request"""

raise NotImplementedError

def pop(self, timeout=0):

"""Pop a request"""

raise NotImplementedError

def clear(self):

"""Clear queue/stack"""

self.server.delete(self.key)

class FifoQueue(Base):

"""Per-spider FIFO queue"""

def \_\_len\_\_(self):

"""Return the length of the queue"""

return self.server.llen(self.key)

def push(self, request):

"""Push a request"""

self.server.lpush(self.key, self.\_encode\_request(request))

def pop(self, timeout=0):

"""Pop a request"""

if timeout > 0:

data = self.server.brpop(self.key, timeout)

if isinstance(data, tuple):

data = data[1]

else:

data = self.server.rpop(self.key)

if data:

return self.\_decode\_request(data)

class PriorityQueue(Base):

"""Per-spider priority queue abstraction using redis' sorted set"""

def \_\_len\_\_(self):

"""Return the length of the queue"""

return self.server.zcard(self.key)

def push(self, request):

"""Push a request"""

data = self.\_encode\_request(request)

score = -request.priority

# We don't use zadd method as the order of arguments change depending on

# whether the class is Redis or StrictRedis, and the option of using

# kwargs only accepts strings, not bytes.

self.server.execute\_command('ZADD', self.key, score, data)

def pop(self, timeout=0):

"""

Pop a request

timeout not support in this queue class

"""

# use atomic range/remove using multi/exec

pipe = self.server.pipeline()

pipe.multi()

pipe.zrange(self.key, 0, 0).zremrangebyrank(self.key, 0, 0)

results, count = pipe.execute()

if results:

return self.\_decode\_request(results[0])

class LifoQueue(Base):

"""Per-spider LIFO queue."""

def \_\_len\_\_(self):

"""Return the length of the stack"""

return self.server.llen(self.key)

def push(self, request):

"""Push a request"""

self.server.lpush(self.key, self.\_encode\_request(request))

def pop(self, timeout=0):

"""Pop a request"""

if timeout > 0:

data = self.server.blpop(self.key, timeout)

if isinstance(data, tuple):

data = data[1]

else:

data = self.server.lpop(self.key)

if data:

return self.\_decode\_request(data)

# TODO: Deprecate the use of these names.

SpiderQueue = FifoQueue

SpiderStack = LifoQueue

SpiderPriorityQueue = PriorityQueue

[复制代码](javascript:void(0);)

### 149.如何基于redis实现消息队列？

[复制代码](javascript:void(0);)

# 通过发布订阅模式的PUB、SUB实现消息队列

# 发布者发布消息到频道了，频道就是一个消息队列。

# 发布者：

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

conn.publish('104.9MH', "hahahahahaha")

# 订阅者：

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

pub = conn.pubsub()

pub.subscribe('104.9MH')

while True:

msg= pub.parse\_response()

print(msg)

对了，redis 做消息队列不合适

业务上避免过度复用一个redis，用它做缓存、做计算，还做任务队列，压力太大，不好。

[复制代码](javascript:void(0);)

### 150.如何基于redis实现发布和订阅？以及发布订阅和消息队列的区别？

[复制代码](javascript:void(0);)

发布和订阅，只要有任务就给所有订阅者没人一份  
 发布者：

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

conn.publish('104.9MH', "hahaha")

订阅者：

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

pub = conn.pubsub()

pub.subscribe('104.9MH')

while True:

msg= pub.parse\_response()

print(msg)

[复制代码](javascript:void(0);)

### 151.什么是codis及作用？

Codis 是一个分布式 Redis 解决方案, 对于上层的应用来说, 连接到 Codis Proxy 和连接原生的 Redis Server 没有明显的区别   
(不支持的命令列表), 上层应用可以像使用单机的 Redis 一样使用, Codis 底层会处理请求的转发, 不停机的数据迁移等工作,  
 所有后边的一切事情, 对于前面的客户端来说是透明的, 可以简单的认为后边连接的是一个内存无限大的 Redis 服务.

### 152.什么是twemproxy及作用？

是 Twtter 开源的一个 Redis 和 Memcache 代理服务器，主要用于管理 Redis 和 Memcached 集群，  
减少与Cache 服务器直接连接的数量。

### 153.写代码实现redis事务操作。

[复制代码](javascript:void(0);)

import redis

pool = redis.ConnectionPool(host='10.211.55.4', port=6379)

conn = redis.Redis(connection\_pool=pool)

# pipe = r.pipeline(transaction=False)

pipe = conn.pipeline(transaction=True)

# 开始事务

pipe.multi()

pipe.set('name', 'bendere')

pipe.set('role', 'sb')

# 提交

pipe.execute()

注意：咨询是否当前分布式redis是否支持事务

[复制代码](javascript:void(0);)

### 154.redis中的watch的命令的作用？

[复制代码](javascript:void(0);)

在Redis的事务中，WATCH命令可用于提供CAS(check-and-set)功能。  
假设我们通过WATCH命令在事务执行之前监控了多个Keys，倘若在WATCH之后有任何Key的值发生了变化，  
EXEC命令执行的事务都将被放弃，同时返回Null multi-bulk应答以通知调用者事务执行失败。

面试题：你如何控制剩余的数量不会出问题？

方式一：- 通过redis的watch实现

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

# conn.set('count',1000)

val = conn.get('count')

print(val)

with conn.pipeline(transaction=True) as pipe:

# 先监视，自己的值没有被修改过

conn.watch('count')

# 事务开始

pipe.multi()

old\_count = conn.get('count')

count = int(old\_count)

print('现在剩余的商品有:%s',count)

input("问媳妇让不让买？")

pipe.set('count', count - 1)

# 执行，把所有命令一次性推送过去

pipe.execute()

方式二 - 数据库的锁

[复制代码](javascript:void(0);)

### 155.基于redis如何实现商城商品数量计数器？

[复制代码](javascript:void(0);)

import redis

conn = redis.Redis(host='192.168.1.41',port=6379)

conn.set('count',1000)

with conn.pipeline() as pipe:

# 先监视，自己的值没有被修改过

conn.watch('count')

# 事务开始

pipe.multi()

old\_count = conn.get('count')

count = int(old\_count)

if count > 0: # 有库存

pipe.set('count', count - 1)

# 执行，把所有命令一次性推送过去

pipe.execute()

[复制代码](javascript:void(0);)

### 156.简述redis分布式锁和redlock的实现机制。

[复制代码](javascript:void(0);)

在不同进程需要互斥地访问共享资源时，分布式锁是一种非常有用的技术手段。   
有很多三方库和文章描述如何用Redis实现一个分布式锁管理器，但是这些库实现的方式差别很大  
，而且很多简单的实现其实只需采用稍微增加一点复杂的设计就可以获得更好的可靠性。   
用Redis实现分布式锁管理器的算法，我们把这个算法称为RedLock。

实现

- 写值并设置超时时间

- 超过一半的redis实例设置成功，就表示加锁完成。

- 使用：安装redlock-py

from redlock import Redlock

dlm = Redlock(

[

{"host": "localhost", "port": 6379, "db": 0},

{"host": "localhost", "port": 6379, "db": 0},

{"host": "localhost", "port": 6379, "db": 0},

]

)

# 加锁，acquire

my\_lock = dlm.lock("my\_resource\_name",10000)

if my\_lock:

# J进行操作

# 解锁,release

dlm.unlock(my\_lock)

else:

print('获取锁失败')

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

redis分布式锁？

# 不是单机操作，又多了一/多台机器

# redis内部是单进程、单线程，是数据安全的(只有自己的线程在操作数据)

----------------------------------------------------------------

\A、B、C，三个实例(主)

1、来了一个'隔壁老王'要操作，且不想让别人操作，so，加锁；

加锁：'隔壁老王'自己生成一个随机字符串，设置到A、B、C里(xxx=666)

2、来了一个'邻居老李'要操作A、B、C，一读发现里面有字符串，擦，被加锁了，不能操作了，等着吧~

3、'隔壁老王'解决完问题，不用锁了，把A、B、C里的key：'xxx'删掉；完成解锁

4、'邻居老李'现在可以访问，可以加锁了

# 问题：

1、如果'隔壁老王'加锁后突然挂了，就没人解锁，就死锁了，其他人干看着没法用咋办？

2、如果'隔壁老王'去给A、B、C加锁的过程中，刚加到Ａ，'邻居老李'就去操作C了，加锁成功or失败？

3、如果'隔壁老王'去给A、B、C加锁时，C突然挂了，这次加锁是成功还是失败？

4、如果'隔壁老王'去给A、B、C加锁时，超时时间为5秒，加一个锁耗时3秒，此次加锁能成功吗？

# 解决

1、安全起见，让'隔壁老王'加锁时设置超时时间，超时的话就会自动解锁(删除key：'xxx')

2、加锁程度达到（1/2）+1个就表示加锁成功，即使没有给全部实例加锁；

3、加锁程度达到（1/2）+1个就表示加锁成功，即使没有给全部实例加锁；

4、不能成功，锁还没加完就过期，没有意义了，应该合理设置过期时间

# 注意

使用需要安装redlock-py

----------------------------------------------------------------

from redlock import Redlock

dlm = Redlock(

    [

        {"host": "localhost", "port": 6379, "db": 0},

        {"host": "localhost", "port": 6379, "db": 0},

        {"host": "localhost", "port": 6379, "db": 0},

    ]

)

# 加锁，acquire

my\_lock = dlm.lock("my\_resource\_name",10000)

if my\_lock:

    # 进行操作

    # 解锁,release

    dlm.unlock(my\_lock)

else:

    print('获取锁失败')

\通过sever.eval(self.unlock\_script)执行一个lua脚本，用来删除加锁时的key

[复制代码](javascript:void(0);)

### 157.什么是一致性哈希？Python中是否有相应模块？

一致性哈希

一致性hash算法（DHT）可以通过减少影响范围的方式，解决增减服务器导致的数据散列问题，从而解决了分布式环境下负载均衡问题；

如果存在热点数据，可以通过增添节点的方式，对热点区间进行划分，将压力分配至其他服务器，重新达到负载均衡的状态。

Python模块--hash\_ring，即Python中的一致性hash

### 158.如何高效的找到redis中所有以aaa开头的key？

[复制代码](javascript:void(0);)

redis 有一个keys命令。

# 语法：KEYS pattern

# 说明：返回与指定模式相匹配的所用的keys。

该命令所支持的匹配模式如下：

1、?：用于匹配单个字符。例如，h?llo可以匹配hello、hallo和hxllo等；

2、\*：用于匹配零个或者多个字符。例如，h\*llo可以匹配hllo和heeeello等；

2、[]：可以用来指定模式的选择区间。例如h[ae]llo可以匹配hello和hallo，但是不能匹配hillo。同时，可以使用“/”符号来转义特殊的字符

# 注意

KEYS 的速度非常快，但如果数据太大，内存可能会崩掉，

如果需要从一个数据集中查找特定的key，最好还是用Redis的集合结构(set)来代替。

## 第四部分 前端、框架和其他（155题）

### 1.谈谈你对http协议的认识。

[复制代码](javascript:void(0);)

　　浏览器本质,socket客户端遵循Http协议

　　HTTP协议本质：通过\r\n分割的规范+ 请求响应之后断开链接 == > 无状态、 短连接

具体：

　　Http协议是建立在tcp之上的，是一种规范，它规范定了发送的数据的数据格式，

然而这个数据格式是通过\r\n 进行分割的，请求头与请求体也是通过2个\r\n分割的，响应的时候，

响应头与响应体也是通过\r\n分割，并且还规定已请求已响应就会断开链接

即---> 短连接、无状态

[复制代码](javascript:void(0);)

### 2.谈谈你对websocket协议的认识。

[复制代码](javascript:void(0);)

websocket是给浏览器新建的一套（类似与http）协议，协议规定：（\r\n分割）浏览器和服务器连接之后不断开，

　　　　以此完成：服务端向客户端主动推送消息。

websocket协议额外做的一些操作

握手 ----> 连接钱进行校验

加密 ----> payload\_len=127/126/<=125 --> mask key

[复制代码](javascript:void(0);)

本质

创建一个连接后不断开的socket

当连接成功之后：

客户端（浏览器）会自动向服务端发送消息，包含： Sec-WebSocket-Key: iyRe1KMHi4S4QXzcoboMmw==

服务端接收之后，会对于该数据进行加密：base64(sha1(swk + magic\_string))

构造响应头：

HTTP/1.1 101 Switching Protocols\r\n

Upgrade:websocket\r\n

Connection: Upgrade\r\n

Sec-WebSocket-Accept: 加密后的值\r\n

WebSocket-Location: ws://127.0.0.1:8002\r\n\r\n

发给客户端（浏览器）

建立：双工通道，接下来就可以进行收发数据

发送数据是加密，解密，根据payload\_len的值进行处理

payload\_len <= 125

payload\_len == 126

payload\_len == 127

获取内容：

mask\_key

数据

根据mask\_key和数据进行位运算，就可以把值解析出来。

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

### 3.什么是magic string ？

客户端向服务端发送消息时，会有一个'sec-websocket-key'和'magic string'的随机字符串(魔法字符串)

# 服务端接收到消息后会把他们连接成一个新的key串，进行编码、加密，确保信息的安全性

### 4.如何创建响应式布局？

[复制代码](javascript:void(0);)

响应式布局是通过@media实现的

@media （min-width：768px）{

.pg-header{

background-color:green;

}

}

@media (min-width:992px){

.pg-header{

background-color:pink;

}

}

代码

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Title</title>

<style>

body{

margin: 0;

}

.pg-header{

background-color: red;

height: 48px;

}

@media (min-width: 768px) {

.pg-header{

background-color: aqua;

}

}

@media (min-width: 992px) {

.pg-header{

background-color: blueviolet;

}

}

</style>

</head>

<body>

<div class="pg-header"></div>

</body>

</html>

[复制代码](javascript:void(0);)

### 5.你曾经使用过哪些前端框架？

jQuery

- BootStrap

- Vue.js(与vue齐名的前端框架React和Angular)

### 6.什么是ajax请求？并使用jQuery和XMLHttpRequest对象实现一个ajax请求。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 基于原生AJAX - Demo

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 基于jQueryAjax - Demo

http://www.cnblogs.com/wupeiqi/articles/5703697.html

### 7.如何在前端实现轮训？

[复制代码](javascript:void(0);)

轮询：通过定时器让程序每隔n秒执行一次操作。

[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html lang="zh-cn">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Title</title>

</head>

<body>

<h1>请选出最帅的男人</h1>

<ul>

{% for k,v in gg.items() %}

<li>ID:{{ k }}, 姓名：{{ v.name }} ，票数：{{ v.count }}</li>

{% endfor %}

</ul>

<script>

setInterval(function () {

location.reload();

},2000)

</script>

</body>

</html>

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

### 8.如何在前端实现长轮训？

[复制代码](javascript:void(0);)

客户端向服务器发送请求，服务器接到请求后hang住连接，等待30秒，30s过后再重新发起请求，

直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。

[复制代码](javascript:void(0);)

<!DOCTYPE html>

<html lang="zh-cn">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Title</title>

</head>

<body>

<h1>请选出最帅的男人</h1>

<ul>

{% for k,v in gg.items() %}

<li style="cursor: pointer" id="user\_{{ k }}" ondblclick="vote({{ k }});">ID:{{ k }}, 姓名：{{ v.name }} ，票数：<span>{{ v.count }}</span></li>

{% endfor %}

</ul>

<script src="/static/jquery-3.3.1.min.js"></script>

<script>

$(function () {

get\_new\_count();

});

function get\_new\_count() {

$.ajax({

url: '/get\_new\_count',

type:'GET',

dataType:'JSON',

success:function (arg) {

if (arg.status){

// 更新票数

var gid = "#user\_" + arg.data.gid;

$(gid).find('span').text(arg.data.count);

}else{

// 10s内没有人投票

}

get\_new\_count();

}

})

}

function vote(gid) {

$.ajax({

url: '/vote',

type:'POST',

data:{gid:gid},

dataType:"JSON",

success:function (arg) {

}

})

}

</script>

</body>

</html>

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

### 9.vuex的作用？

[复制代码](javascript:void(0);)

多组件之间共享：vuex

补充luffyvue

1:router-link / router-view

2:双向绑定，用户绑定v-model

3：循环展示课程：v-for

4：路由系统，添加动态参数

5：cookie操作：vue-cookies

6：多组件之间共享：vuex

7：发送ajax请求：axios (js模块)

[复制代码](javascript:void(0);)

### 10.vue中的路由的拦截器的作用？

vue-resource的interceptors拦截器的作用正是解决此需求的妙方。  
在每次http的请求响应之后，如果设置了拦截器如下，会优先执行拦截器函数，获取响应体，然后才会决定是否把response返回给then进行接收

### 11.axios的作用？

发送ajax请求：axios (js模块)

### 12.列举vue的常见指令。

[复制代码](javascript:void(0);)

1、v-if指令:判断指令，根据表达式值得真假来插入或删除相应的值。

2、v-show指令:条件渲染指令，无论返回的布尔值是true还是false，元素都会存在在html中，只是false的元素会隐藏在html中，并不会删除.

3、v-else指令:配合v-if或v-else使用。

4、v-for指令:循环指令，相当于遍历。

5、v-bind:给DOM绑定元素属性。

6、v-on指令:监听DOM事件。

[复制代码](javascript:void(0);)

### 13.简述jsonp及实现原理？

JSONP

jsonp是json用来跨域的一个东西。原理是通过script标签的跨域特性来绕过同源策略。

JSONP的简单实现模式，或者说是JSONP的原型：创建一个回调函数，然后在远程服务上调用这个函数并且将JSON 数据形式作为参数传递，

完成回调。

### 14.什么是cors ？

[复制代码](javascript:void(0);)

CORS

浏览器将CORS请求分成两类：简单请求和赋复杂请求

简单请求(同时满足以下两大条件)

（1）请求方法是以下三种方法之一:

HEAD

GET

POST

（2）HTTP的头信息不超出以下几种字段：

Accept

Accept-Language

Content-Language

Last-Event-ID

Content-Type ：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain

凡是不同时满足上面两个条件，就属于非简单请求

[复制代码](javascript:void(0);)

### 15.列举Http请求中常见的请求方式？

[复制代码](javascript:void(0);)

GET、POST、  
PUT、patch（修改数据）

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

DELETE  
  
传值代码f

Request.QueryString方法针对控件id  
Request.Form方法针对控件名称name

[复制代码](javascript:void(0);)

### 16.列举Http请求中的状态码？

[复制代码](javascript:void(0);)

分类：

1\*\* 信息，服务器收到请求，需要请求者继续执行操作

2\*\* 成功，操作被成功接收并处理

3\*\* 重定向，需要进一步的操作以完成请求

4\*\* 客户端错误，请求包含语法错误或无法完成请求

5\*\* 服务器错误，服务器在处理请求的过程中发生了错误

常见的状态码

200 -请求成功  
202 -已接受请求，尚未处理  
204 -请求成功，且不需返回内容

301 - 资源（网页等）被永久转移到其他url  
400 - 请求的语义或是参数有错  
403 - 服务器拒绝请求

404 - 请求资源（网页）不存在

500 - 内部服务器错误  
502 - 网关错误，一般是服务器压力过大导致连接超时  
503 - 由于超载或系统维护，服务器暂时的无法处理客户端的请求。

[复制代码](javascript:void(0);)

### 17.列举Http请求中常见的请求头？

- user-agent

- host

- referer

- cookie

- content-type

### 18.看图写结果（js）：



 李杰

### 看图写结果（js）：



 武沛奇

### 看图写结果：（js）



 老男孩

### 看图写结果：（js）



 undefined

### 看图写结果：（js）



  武沛奇

### 看图写结果：（js）



 Alex

### 19.django、flask、tornado框架的比较？

[复制代码](javascript:void(0);)

对于django，大而全的框架它的内部组件比较多，内部提供：ORM、Admin、中间件、Form、ModelForm、Session、  
缓存、信号、CSRF；功能也都挺完善的

- flask，微型框架，内部组件就比较少了，但是有很多第三方组件来扩展它，  
　　比如说有那个wtform（与django的modelform类似，表单验证）、flask-sqlalchemy（操作数据库的）、  
　　flask-session、flask-migrate、flask-script、blinker可扩展强，第三方组件丰富。所以对他本身来说有那种短小精悍的感觉

- tornado，异步非阻塞。

django和flask的共同点就是，他们2个框架都没有写socket，所以他们都是利用第三方模块wsgi。

但是内部使用的wsgi也是有些不同的：django本身运行起来使用wsgiref，而flask使用werkzeug wsgi

还有一个区别就是他们的请求管理不太一样：django是通过将请求封装成request对象，再通过参数传递，而flask是通过上下文管理机制

Tornado

# 是一个轻量级的Web框架，异步非阻塞+内置WebSocket功能。

'目标'：通过一个线程处理N个并发请求(处理IO)。

'内部组件

    #内部自己实现socket

    #路由系统

    #视图

 #模板

　　 #cookie

    #csrf

[复制代码](javascript:void(0);)

### 20.什么是wsgi？

[复制代码](javascript:void(0);)

是web服务网关接口，是一套协议。

是通过以下模块实现了wsgi协议：

- wsgiref

- werkzurg

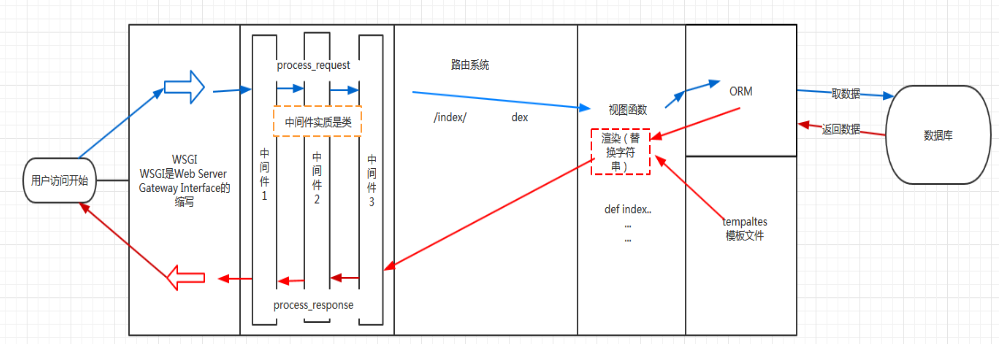
- uwsgi 关于部署

以上模块本质：编写socket服务端，用于监听请求，当有请求到来，则将请求数据进行封装，然后交给web框架处理。

[复制代码](javascript:void(0);)

### 21.django请求的生命周期？

用户请求进来先走到 wsgi 然后将请求交给 jango的中间件 穿过django中间件（方法是process\_request）   
接着就是 路由匹配 路由匹配成功之后就执行相应的 视图函数   
在视图函数中可以调用orm做数据库操作 再从模板路径 将模板拿到 然后在后台进行模板渲染   
模板渲染完成之后就变成一个字符串 再把这个字符串经过所有中间件（方法：process\_response） 和wsgi 返回给用户



### 22.列举django的内置组件？

[复制代码](javascript:void(0);)

form 组件

- 对用户请求的数据进行校验

- 生成HTML标签

PS：

- form对象是一个可迭代对象。

- 问题：choice的数据如果从数据库获取可能会造成数据无法实时更新

- 重写构造方法，在构造方法中重新去数据库获取值。

- ModelChoiceField字段

from django.forms import Form

from django.forms import fields

from django.forms.models import ModelChoiceField

class UserForm(Form):

name = fields.CharField(label='用户名',max\_length=32)

email = fields.EmailField(label='邮箱')

ut\_id = ModelChoiceField(queryset=models.UserType.objects.all())

依赖：

class UserType(models.Model):

title = models.CharField(max\_length=32)

def \_\_str\_\_(self):

return self.title

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

信号、

django的信号其实就是django内部为开发者预留的一些自定制功能的钩子。

只要在某个信号中注册了函数，那么django内部执行的过程中就会自动触发注册在信号中的函数。

如：

pre\_init # django的modal执行其构造方法前，自动触发

post\_init # django的modal执行其构造方法后，自动触发

pre\_save # django的modal对象保存前，自动触发

post\_save # django的modal对象保存后，自动触发

场景:

在数据库某些表中添加数据时，可以进行日志记录。

CSRF、

目标：防止用户直接向服务端发起POST请求。  
对所有的post请求做验证/ 将jango生成的一串字符串发送给我们，一种是从请求体发过来，一种是放在隐藏的标签里面用的是process\_view

方案：先发送GET请求时，将token保存到：cookie、Form表单中（隐藏的input标签），  
以后再发送请求时只要携带过来即可。  
   
  
ContentType   
contenttype是django的一个组件（app），  
为我们找到django程序中所有app中的所有表并添加到记录中。  
 可以使用他再加上表中的两个字段实现：一张表和N张表创建FK关系。 - 字段：表名称 - 字段：数据行ID   
应用：路飞表结构优惠券和专题课和学位课关联。

[复制代码](javascript:void(0);)

中间件

对所有的请求进行批量处理，在视图函数执行前后进行自定义操作。  
应用：用户登录校验  
问题：为甚么不使用装饰器？  
如果不使用中间件，就需要给每个视图函数添加装饰器，太繁琐

权限组件:

用户登录后，将权限放到session中，然后再每次请求进来在中间件里，根据当前的url去session中匹配，  
判断当前用户是否有权限访问当前url,有权限就继续访问，没有就返回，

（检查的东西就可以放到中间件中进行统一处理）在process\_request方法里面做的，

　我们的中间件是放在session后面，因为中间件需要到session里面取数据

[复制代码](javascript:void(0);)

session

cookie与session区别

（a）cookie是保存在浏览器端的键值对，而session是保存的服务器端的键值对，但是依赖cookie。  
（也可以不依赖cookie，可以放在url，或请求头但是cookie比较方便）

（b）以登录为例，cookie为通过登录成功后，设置明文的键值对，并将键值对发送客户端存，明文信息可能存在泄漏，不安全；  
　　session则是生成随机字符串，发给用户，并写到浏览器的cookie中，同时服务器自己也会保存一份。

（c）在登录验证时，cookie：根据浏览器发送请求时附带的cookie的键值对进行判断，如果存在，则验证通过；  
　　session：在请求用户的cookie中获取随机字符串，根据随机字符串在session中获取其对应的值进行验证

[复制代码](javascript:void(0);)

cors跨域（场景：前后端分离时，本地测试开发时使用）

如果网站之间存在跨域，域名不同，端口不同会导致出现跨域，但凡出现跨域，浏览器就会出现同源策略的限制

解决：在我们的服务端给我们响应数据，加上响应头---> 在中间件加的

[复制代码](javascript:void(0);)

缓存/   
常用的数据放在缓存里面，就不用走视图函数，请求进来通过所有的process\_request,会到缓存里面查数据，有就直接拿，  
　　　　　　　　没有就走视图函数  
　　　　　　关键点：1：执行完所有的process\_request才去缓存取数据  
　　　　　　　　　　2：执行完所有的process\_response才将数据放到缓存

**关于缓存问题**

1:为什么放在最后一个process\_request才去缓存

因为需要验证完用户的请求，才能返回数据

2:什么时候将数据放到缓存中

第一次走中间件，缓存没有数据，会走视图函数，取数据库里面取数据，

当走完process\_response,才将数据放到缓存里，因为，走process\_response的时候可能给我们的响应加处理

[复制代码](javascript:void(0);)

**为什么使用缓存**

将常用且不太频繁修改的数据放入缓存。

以后用户再来访问，先去缓存查看是否存在，如果有就返回

否则，去数据库中获取并返回给用户（再加入到缓存，以便下次访问）

### 23.列举django中间件的5个方法？以及django中间件的应用场景？

process\_request(self,request) 先走request 通过路由匹配返回

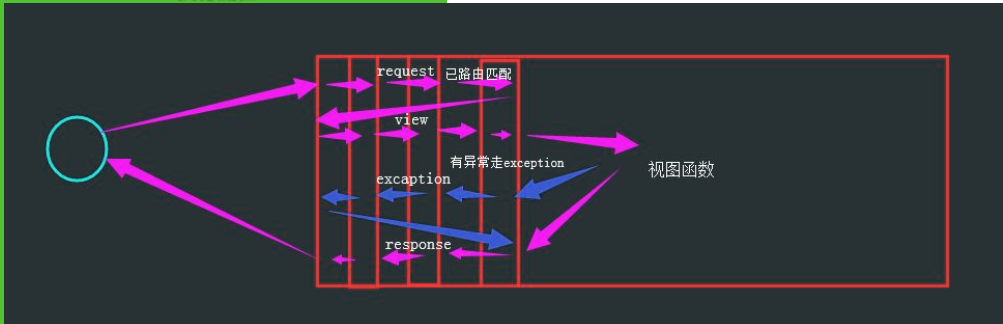
process\_view(self, request, callback, callback\_args, callback\_kwargs) 再返回执行view

process\_template\_response(self,request,response) 当视图函数的返回值

process\_exception(self, request, exception) 当视图函数的返回值对象中有render方法时，该方法才会被调用

process\_response(self, request, response)

执行流程



### 24.简述什么是FBV和CBV？

[复制代码](javascript:void(0);)

**FBV 基于函数**

# FBV 写法

# urls.py

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

# views.py

def login(request):

if request.method == "POST":

print(request.POST)

return render(request,"login.html")

# HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>登录页面</title>

</head>

<body>

<form action="{% url 'login' %}" method="post" enctype="multipart/form-data">

<input type="text" name="user2">

<input type="file" name="file">

<input type="submit" value="提交">

</form>

</body>

</html>

**CBV 基于类**

# urls.py

url(r'^login/$',views.Login.as\_view(), name="login"),

# views.py

from django.views import View

class Login(View): # 类首字母大写

def get(self,request):

return render(request,"login.html")

def post(self,request):

print(request.POST)

return HttpResponse("OK")

加装饰器

=================================

class IndexView(View):

# 如果是crsf相关，必须放在此处

def dispach(self,request):

# 通过反射执行post/get

@method\_decoretor(装饰器函数)

def get(self,request):

pass

def post(self,request):

pass

路由：IndexView.as\_view()

[复制代码](javascript:void(0);)

### 25.FBV与CBV的区别

- 没什么区别，因为他们的本质都是函数。CBV的.as\_view()返回的view函数，view函数中调用类的dispatch方法，

在dispatch方法中通过反射执行get/post/delete/put等方法。D

非要说区别的话：

- CBV比较简洁，GET/POST等业务功能分别放在不同get/post函数中。FBV自己做判断进行区分。

### 26.django的request对象是在什么时候创建的？

当请求一个页面时, Django会建立一个包含请求元数据的 HttpRequest 对象.   
当Django 加载对应的视图时, HttpRequest对象将作为视图函数的第一个参数.   
每个视图会返回一个HttpResponse对象.

### 27.如何给CBV的程序添加装饰器？

[复制代码](javascript:void(0);)

添加装饰器

方式一：

from django.views import View

from django.utils.decorators import method\_decorator ---> 需要引入memethod\_decorator

def auth(func):

def inner(\*args,\*\*kwargs):

return func(\*args,\*\*kwargs)

return inner

class UserView(View):

@method\_decorator(auth)

def get(self,request,\*args,\*\*kwargs):

return HttpResponse('...')

方式二：

- csrf的装饰器要加到dispath前面

from django.views import View

from django.utils.decorators import method\_decorator

from django.views.decorators.csrf import csrf\_exempt,csrf\_protect ---> 需要引入 csrf\_exempt

class UserView(View):

@method\_decorator(csrf\_exempt)

def dispatch(self, request, \*args, \*\*kwargs):

return HttpResponse('...')

或者：

from django.views import View

from django.utils.decorators import method\_decorator

from django.views.decorators.csrf import csrf\_exempt,csrf\_protect

@method\_decorator(csrf\_exempt,name='dispatch') ---> 指定名字

class UserView(View):

def dispatch(self, request, \*args, \*\*kwargs):

return HttpResponse('...')

[复制代码](javascript:void(0);)

### 28.列举django orm 中所有的方法（QuerySet对象的所有方法）

[复制代码](javascript:void(0);)

返回QuerySet对象的方法有：

all()

filter()

exclude()

order\_by()

reverse()

distinct()

特殊的QuerySet：

values() 返回一个可迭代的字典序列

values\_list() 返回一个可迭代的元组序列

返回具体对象的：

get()

first()

last()

返回布尔值的方法有：

exists()

返回数字的方法有：

count()

[复制代码](javascript:void(0);)

### 29.only和defer的区别？

[复制代码](javascript:void(0);)

def defer(self, \*fields):

models.UserInfo.objects.defer('username','id')

或

models.UserInfo.objects.filter(...).defer('username','id')

#映射中排除某列数据

def only(self, \*fields):

#仅取某个表中的数据

models.UserInfo.objects.only('username','id')

或

models.UserInfo.objects.filter(...).only('username','id')

[复制代码](javascript:void(0);)

### 30.select\_related和prefetch\_related的区别？

[复制代码](javascript:void(0);)

# 他俩都用于连表查询，减少SQL查询次数

\select\_related

select\_related主要针一对一和多对一关系进行优化，通过多表join关联查询，一次性获得所有数据，

存放在内存中，但如果关联的表太多，会严重影响数据库性能。

def index(request):

obj = Book.objects.all().select\_related("publisher")

return render(request, "index.html", locals())

\prefetch\_related

prefetch\_related是通过分表，先获取各个表的数据，存放在内存中，然后通过Python处理他们之间的关联。

def index(request):

obj = Book.objects.all().prefetch\_related("publisher")

return render(request, "index.html", locals())

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

def select\_related(self, \*fields)

性能相关：表之间进行join连表操作，一次性获取关联的数据。

model.tb.objects.all().select\_related()

model.tb.objects.all().select\_related('外键字段')

model.tb.objects.all().select\_related('外键字段\_\_外键字段')

def prefetch\_related(self, \*lookups)

性能相关：多表连表操作时速度会慢，使用其执行多次SQL查询在Python代码中实现连表操作。

# 获取所有用户表

# 获取用户类型表where id in (用户表中的查到的所有用户ID)

models.UserInfo.objects.prefetch\_related('外键字段')

from django.db.models import Count, Case, When, IntegerField

Article.objects.annotate(

numviews=Count(Case(

When(readership\_\_what\_time\_\_lt=treshold, then=1),

output\_field=CharField(),

))

)

students = Student.objects.all().annotate(num\_excused\_absences=models.Sum(

models.Case(

models.When(absence\_\_type='Excused', then=1),

default=0,

output\_field=models.IntegerField()

)))

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

# 1次SQL

# select \* from userinfo

objs = UserInfo.obejcts.all()

for item in objs:

print(item.name)

# n+1次SQL

# select \* from userinfo

objs = UserInfo.obejcts.all()

for item in objs:

# select \* from usertype where id = item.id

print(item.name,item.ut.title)

select\_related（）

# 1次SQL

# select \* from userinfo inner join usertype on userinfo.ut\_id = usertype.id

objs = UserInfo.obejcts.all().select\_related('ut') 连表查询

for item in objs:

print(item.name,item.ut.title)

.prefetch\_related()

# select \* from userinfo where id <= 8

# 计算：[1,2]

# select \* from usertype where id in [1,2]

objs = UserInfo.obejcts.filter(id\_\_lte=8).prefetch\_related('ut')

for obj in objs:

print(obj.name,obj.ut.title)

[复制代码](javascript:void(0);)

### 31.filter和exclude的区别？

[复制代码](javascript:void(0);)

def filter(self, \*args, \*\*kwargs)

# 条件查询(符合条件)

# 查出符合条件

# 条件可以是：参数，字典，Q

def exclude(self, \*args, \*\*kwargs)

# 条件查询(排除条件)

# 排除不想要的

# 条件可以是：参数，字典，Q

[复制代码](javascript:void(0);)

### 32.列举django orm中三种能写sql语句的方法。

[复制代码](javascript:void(0);)

原生SQL ---> connection

from django.db import connection, connections

cursor = connection.cursor() # cursor = connections['default'].cursor()

cursor.execute("""SELECT \* from auth\_user where id = %s""", [1])

row = cursor.fetchone() # fetchall()/fetchmany(..)

靠近原生SQL-->extra\raw

extra

- extra

def extra(self, select=None, where=None, params=None, tables=None, order\_by=None,

select\_params=None)

# 构造额外的查询条件或者映射，如：子查询

Entry.objects.extra(select={'new\_id': "select col from sometable where othercol > %s"},

select\_params=(1,))

Entry.objects.extra(where=['headline=%s'], params=['Lennon'])

Entry.objects.extra(where=["foo='a' OR bar = 'a'", "baz = 'a'"])

Entry.objects.extra(select={'new\_id': "select id from tb where id > %s"}, s

elect\_params=(1,), order\_by=['-nid'])

- raw

def raw(self, raw\_query, params=None, translations=None, using=None):

# 执行原生SQL

models.UserInfo.objects.raw('select \* from userinfo')

# 如果SQL是其他表时，必须将名字设置为当前UserInfo对象的主键列名

models.UserInfo.objects.raw('select id as nid,name as title from 其他表')

# 为原生SQL设置参数

models.UserInfo.objects.raw('select id as nid from userinfo where nid>%s', params=[12,])

# 将获取的到列名转换为指定列名

name\_map = {'first': 'first\_name', 'last': 'last\_name', 'bd': 'birth\_date', 'pk': 'id'}

Person.objects.raw('SELECT \* FROM some\_other\_table', translations=name\_map)

# 指定数据库

models.UserInfo.objects.raw('select \* from userinfo', using="default")

[复制代码](javascript:void(0);)

### 33.django orm 中如何设置读写分离？

[复制代码](javascript:void(0);)

方式一：手动使用queryset的using方法

from django.shortcuts import render,HttpResponse

from app01 import models

def index(request):

models.UserType.objects.using('db1').create(title='普通用户')

　　# 手动指定去某个数据库取数据

result = models.UserType.objects.all().using('db1')

print(result)

return HttpResponse('...')

方式二：写配置文件

class Router1:

　　# 指定到某个数据库取数据

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth models go to auth\_db.

"""

if model.\_meta.model\_name == 'usertype':

return 'db1'

else:

return 'default'

　　　# 指定到某个数据库存数据

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth models go to auth\_db.

"""

return 'default'

再写到配置

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

},

'db1': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

DATABASE\_ROUTERS = ['db\_router.Router1',]

[复制代码](javascript:void(0);)

### 34.F和Q的作用?

[复制代码](javascript:void(0);)

F:主要用来获取原数据进行计算。

Django 支持 F() 对象之间以及 F() 对象和常数之间的加减乘除和取模的操作。

修改操作也可以使用F函数,比如将每件商品的价格都在原价格的基础上增加10

from django.db.models import F

from app01.models import Goods

Goods.objects.update(price=F("price")+10) # 对于goods表中每件商品的价格都在原价格的基础上增加10元  
F查询专门对对象中某列值的操作，不可使用\_\_双下划线！

Q:用来进行复杂查询  
    Q查询可以组合使用 “&”, “|” 操作符，当一个操作符是用于两个Q的对象,它产生一个新的Q对象，  
　　Q对象可以用 “~” 操作符放在前面表示否定，也可允许否定与不否定形式的组合。  
　　Q对象可以与关键字参数查询一起使用，不过一定要把Q对象放在关键字参数查询的前面。

Q(条件1) | Q(条件2) 或

Q(条件1) & Q(条件2) 且

Q(条件1) & ~Q(条件2) 非

[复制代码](javascript:void(0);)

### 35.values和values\_list的区别？

def values(self, \*fields):

# 获取每行数据为字典格式

def values\_list(self, \*fields, \*\*kwargs):

# 获取每行数据为元祖

### 36.如何使用django orm批量创建数据？

[复制代码](javascript:void(0);)

def bulk\_create(self, objs, batch\_size=None):

# 批量插入

# batch\_size表示一次插入的个数

objs = [

models.DDD(name='r11'),

models.DDD(name='r22')

]

models.DDD.objects.bulk\_create(objs, 10)

[复制代码](javascript:void(0);)

### 37.django的Form和ModeForm的作用？

[复制代码](javascript:void(0);)

- 作用：

- 对用户请求数据格式进行校验

- 自动生成HTML标签

- 区别：

- Form，字段需要自己手写。

class Form(Form):

xx = fields.CharField(.)

xx = fields.CharField(.)

xx = fields.CharField(.)

xx = fields.CharField(.)

- ModelForm，可以通过Meta进行定义

class MForm(ModelForm):

class Meta:

fields = "\_\_all\_\_"

model = UserInfo

- 应用：只要是客户端向服务端发送表单数据时，都可以进行使用，如：用户登录注册

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

方式一:重写构造方法，在构造方法中重新去数据库获取值

class UserForm(Form):

name = fields.CharField(label='用户名',max\_length=32)

email = fields.EmailField(label='邮箱')

ut\_id = fields.ChoiceField(

# choices=[(1,'普通用户'),(2,'IP用户')]

choices=[]

)

def \_\_init\_\_(self,\*args,\*\*kwargs):

super(UserForm,self).\_\_init\_\_(\*args,\*\*kwargs)

self.fields['ut\_id'].choices = models.UserType.objects.all().values\_list('id','title')

方式二: ModelChoiceField字段

from django.forms import Form

from django.forms import fields

from django.forms.models import ModelChoiceField

class UserForm(Form):

name = fields.CharField(label='用户名',max\_length=32)

email = fields.EmailField(label='邮箱')

ut\_id = ModelChoiceField(queryset=models.UserType.objects.all())

依赖：

class UserType(models.Model):

title = models.CharField(max\_length=32)

def \_\_str\_\_(self):

return self.title

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

在django2.0后，定义外键和一对一关系的时候需要加on\_delete选项，此参数为了避免两个表里的数据不一致问题，不然会报错：

TypeError: \_\_init\_\_() missing 1 required positional argument: 'on\_delete'

举例说明：

user=models.OneToOneField(User)

owner=models.ForeignKey(UserProfile)

需要改成：

user=models.OneToOneField(User,on\_delete=models.CASCADE) --在老版本这个参数（models.CASCADE）是默认值

owner=models.ForeignKey(UserProfile,on\_delete=models.CASCADE) --在老版本这个参数（models.CASCADE）是默认值

参数说明：

on\_delete有CASCADE、PROTECT、SET\_NULL、SET\_DEFAULT、SET()五个可选择的值

CASCADE：此值设置，是级联删除。

PROTECT：此值设置，是会报完整性错误。

SET\_NULL：此值设置，会把外键设置为null，前提是允许为null。

SET\_DEFAULT：此值设置，会把设置为外键的默认值。

SET()：此值设置，会调用外面的值，可以是一个函数。

一般情况下使用CASCADE就可以了。

[复制代码](javascript:void(0);)

### 40.django中csrf的实现机制？

[复制代码](javascript:void(0);)

目的：防止用户直接向服务端发起POST请求

- 用户先发送GET获取csrf token: Form表单中一个隐藏的标签 + token

- 发起POST请求时，需要携带之前发送给用户的csrf token；

- 在中间件的process\_view方法中进行校验。

在html中添加{%csrf\_token%}标签

[复制代码](javascript:void(0);)

### 41.django如何实现websocket？

django中可以通过channel实现websocket

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

[复制代码](javascript:void(0);)

**//方式一给每个ajax都加上上请求头**

function Do1(){

$.ajax({

url:"/index/",

data:{id:1},

type:'POST',

　　　　　　　data:{csrfmiddlewaretoken:'{{ csrf\_token }}',name:'alex'}

success:function(data){

console.log(data);

}

});

}

**方式二：需要先下载jQuery-cookie，才能去cookie中获取token**

function Do1(){

$.ajax({

url:"/index/",

data:{id:1},

type:'POST',

headers:{

'X-CSRFToken':$.cookie('csrftoken') // 去cookie中获取

},

success:function(data){

console.log(data);

}

});

}

**方式三：搞个函数ajaxSetup，当有多的ajax请求，即会执行这个函数**

$.ajaxSetup({

beforeSend:function (xhr,settings) {

xhr.setRequestHeader("X-CSRFToken",$.cookie('csrftoken'))

}

});

函数版本

<body>

<input type="button" onclick="Do1();" value="Do it"/>

<input type="button" onclick="Do2();" value="Do it"/>

<input type="button" onclick="Do3();" value="Do it"/>

<script src="/static/jquery-3.3.1.min.js"></script>

<script src="/static/jquery.cookie.js"></script>

<script>

$.ajaxSetup({

beforeSend: function(xhr, settings) {

xhr.setRequestHeader("X-CSRFToken", $.cookie('csrftoken'));

}

});

function Do1(){

$.ajax({

url:"/index/",

data:{id:1},

type:'POST',

success:function(data){

console.log(data);

}

});

}

function Do2(){

$.ajax({

url:"/index/",

data:{id:1},

type:'POST',

success:function(data){

console.log(data);

}

});

}

function Do3(){

$.ajax({

url:"/index/",

data:{id:1},

type:'POST',

success:function(data){

console.log(data);

}

});

}

</script>

</body>

[复制代码](javascript:void(0);)

### 43.django中如何实现orm表中添加数据时创建一条日志记录。

 给信号注册函数

[复制代码](javascript:void(0);)

使用django的信号机制，可以在添加、删除数据前后设置日志记录

pre\_init # Django中的model对象执行其构造方法前,自动触发

post\_init # Django中的model对象执行其构造方法后,自动触发

pre\_save # Django中的model对象保存前,自动触发

post\_save # Django中的model对象保存后,自动触发

pre\_delete # Django中的model对象删除前,自动触发

post\_delete # Django中的model对象删除后,自动触发

[复制代码](javascript:void(0);)

### 44.django缓存如何设置？

[复制代码](javascript:void(0);)

jango中提供了6种缓存方式：

　　开发调试（不加缓存）

　　内存

　　文件

　　数据库

　　Memcache缓存（python-memcached模块）

　　Memcache缓存（pylibmc模块）

安装第三方组件支持redis：

　　django-redis组件

设置缓存

# 全站缓存（中间件）

MIDDLEWARE\_CLASSES = (

‘django.middleware.cache.UpdateCacheMiddleware’, #第一

'django.middleware.common.CommonMiddleware',

‘django.middleware.cache.FetchFromCacheMiddleware’, #最后

)

# 视图缓存

from django.views.decorators.cache import cache\_page

import time

@cache\_page(15) #超时时间为15秒

def index(request):

t=time.time() #获取当前时间

return render(request,"index.html",locals())

# 模板缓存

{% load cache %}

<h3 style="color: green">不缓存:-----{{ t }}</h3>

{% cache 2 'name' %} # 存的key

<h3>缓存:-----:{{ t }}</h3>

{% endcache %}

[复制代码](javascript:void(0);)

### 45.django的缓存能使用redis吗？如果可以的话，如何配置？

[复制代码](javascript:void(0);)

pip install django-redis

apt-get install redis-serv

在setting添加配置文件

CACHES = {

"default": {

"BACKEND": "django\_redis.cache.RedisCache", # 缓存类型

"LOCATION": "127.0.0.1:6379", # ip端口

"OPTIONS": {

"CLIENT\_CLASS": "django\_redis.client.DefaultClient", #

"CONNECTION\_POOL\_KWARGS": {"max\_connections": 100} # 连接池最大连接数

# "PASSWORD": "密码",

}

}

}

使用

from django.shortcuts import render,HttpResponse

from django\_redis import get\_redis\_connection

def index(request):

# 根据名字去连接池中获取连接

conn = get\_redis\_connection("default")

conn.hset('n1','k1','v1') # 存数据

return HttpResponse('...')

[复制代码](javascript:void(0);)

### 46.django路由系统中name的作用？

[复制代码](javascript:void(0);)

反向解析路由字符串

路由系统中name的作用：反向解析  
url**(r'^home'**, views.home, name**='home')**在模板中使用：**{ %**url **'home' %}**在视图中使用：reverse**(**“home”）

[复制代码](javascript:void(0);)

### 47.django的模板中filter和simple\_tag的区别？

[复制代码](javascript:void(0);)

filter : 类似管道,只能接受两个参数第一个参数是|前的数据

simple\_tag : 类似函数

1、模板继承：**{ %**extends **'layouts.html' %}**2、自定义方法  
    **'filter'**：只能传递两个参数，可以在if、for语句中使用  
    **'simple\_tag'**：可以无线传参，不能在if for中使用  
    **'inclusion\_tags'**：可以使用模板和后端数据  
3、防xss攻击： **'|safe'**、**'mark\_safe'**

[复制代码](javascript:void(0);)

### 48.[django-debug-toolbar的作用？](https://blog.csdn.net/weixin_39198406/article/details/78821677)

一、查看访问的速度、数据库的行为、cache命中等信息。

二、尤其在Mysql访问等的分析上大有用处(sql查询速度)

### 49.django中如何实现单元测试？

对于每一个测试方法都会将setUp()和tearDown()方法执行一遍

会单独新建一个测试数据库来进行数据库的操作方面的测试，默认在测试完成后销毁。

在测试方法中对数据库进行增删操作，最后都会被清除。也就是说，在test\_add中插入的数据，在test\_add测试结束后插入的数据会被清除。

django单元测试时为了模拟生产环境，会修改settings中的变量，例如, 把DEBUG变量修改为True, 把ALLOWED\_HOSTS修改为[\*]。

### 50.解释orm中 db first 和 code first的含义？

db first: 先创建数据库，再更新表模型

code first：先写表模型，再更新数据库

https://www.cnblogs.com/jassin-du/p/8988897.html

### 51.django中如何根据数据库表生成model中的类？

1、修改seting文件，在setting里面设置要连接的数据库类型和名称、地址

2、运行下面代码可以自动生成models模型文件

- python manage.py inspectdb

3、创建一个app执行下下面代码：

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

### 52.使用orm和原生sql的优缺点？

[复制代码](javascript:void(0);)

SQL：

# 优点：

执行速度快

# 缺点：

编写复杂，开发效率不高

---------------------------------------------------------------------------

ORM：

# 优点：

让用户不再写SQL语句，提高开发效率

可以很方便地引入数据缓存之类的附加功能

# 缺点：

在处理多表联查、where条件复杂查询时，ORM的语法会变得复杂。

没有原生SQL速度快

[复制代码](javascript:void(0);)

### 53.简述MVC和MTV

MVC：model、view(模块)、controller(视图)

MTV：model、tempalte、view

### 54.django的contenttype组件的作用？

contenttype是django的一个组件(app)，它可以将django下所有app下的表记录下来

可以使用他再加上表中的两个字段,实现一张表和N张表动态创建FK关系。

- 字段：表名称

- 字段：数据行ID

应用：路飞表结构优惠券和专题课和学位课关联

### 55.谈谈你对restfull 规范的认识？

[复制代码](javascript:void(0);)

restful其实就是一套编写接口的'协议'，规定如何编写以及如何设置返回值、状态码等信息。

# 最显著的特点：

# 用restful:

给用户一个url，根据method不同在后端做不同的处理

比如：post创建数据、get获取数据、put和patch修改数据、delete删除数据。

# 不用restful:

给调用者很多url，每个url代表一个功能，比如：add\_user/delte\_user/edit\_user/

# 当然，还有协议其他的，比如：

'版本'来控制让程序有多个版本共存的情况，版本可以放在 url、请求头（accept/自定义）、GET参数

'状态码'200/300/400/500

'url中尽量使用名词'restful也可以称为“面向资源编程”

'api标示'

api.luffycity.com

www.luffycity.com/api/

[复制代码](javascript:void(0);)

### 56.接口的幂等性是什么意思？

[复制代码](javascript:void(0);)

'一个接口通过1次相同的访问，再对该接口进行N次相同的访问时，对资源不造影响就认为接口具有幂等性。'

GET， #第一次获取结果、第二次也是获取结果对资源都不会造成影响，幂等。

POST， #第一次新增数据，第二次也会再次新增，非幂等。

PUT， #第一次更新数据，第二次不会再次更新，幂等。

PATCH，#第一次更新数据，第二次不会再次更新，非幂等。

DELTE，#第一次删除数据，第二次不在再删除，幂等。

[复制代码](javascript:void(0);)

### 57.什么是RPC？

'远程过程调用协议'

是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。

进化的顺序: 现有的RPC,然后有的RESTful规范

### 58.Http和Https的区别？

[复制代码](javascript:void(0);)

#Http: 80端口

#https: 443端口

# http信息是明文传输，https则是具有安全性的ssl加密传输协议。

#- 自定义证书

- 服务端：创建一对证书

- 客户端：必须携带证书

#- 购买证书

- 服务端： 创建一对证书，。。。。

- 客户端： 去机构获取证书，数据加密后发给咱们的服务单

- 证书机构:公钥给改机构

[复制代码](javascript:void(0);)

### 59.为什么要使用django rest framework框架？

[复制代码](javascript:void(0);)

# 在编写接口时可以不使用django rest framework框架，

# 不使用：也可以做，可以用django的CBV来实现，开发者编写的代码会更多一些。

# 使用：内部帮助我们提供了很多方便的组件，我们通过配置就可以完成相应操作，如：

'序列化'可以做用户请求数据校验+queryset对象的序列化称为json

'解析器'获取用户请求数据request.data，会自动根据content-type请求头的不能对数据进行解析

'分页'将从数据库获取到的数据在页面进行分页显示。

# 还有其他组件：

'认证'、'权限'、'访问频率控制

[复制代码](javascript:void(0);)

### 60.django rest framework框架中都有那些组件？

[复制代码](javascript:void(0);)

#- 路由，自动帮助开发者快速为一个视图创建4个url

www.oldboyedu.com/api/v1/student/$

www.oldboyedu.com/api/v1/student(?P<format>\w+)$

www.oldboyedu.com/api/v1/student/(?P<pk>\d+)/$

www.oldboyedu.com/api/v1/student/(?P<pk>\d+)(?P<format>\w+)$

#- 版本处理

- 问题：版本都可以放在那里？

- url

- GET

- 请求头

#- 认证

- 问题：认证流程？

#- 权限

- 权限是否可以放在中间件中？以及为什么？

#- 访问频率的控制

匿名用户可以真正的防止？无法做到真正的访问频率控制，只能把小白拒之门外。

如果要封IP，使用防火墙来做。

登录用户可以通过用户名作为唯一标示进行控制，如果有人注册很多账号，则无法防止。

#- 视图

#- 解析器 ，根据Content-Type请求头对请求体中的数据格式进行处理。request.data

#- 分页

#- 序列化

- 序列化

- source

- 定义方法

- 请求数据格式校验

#- 渲染器

[复制代码](javascript:void(0);)

### 61.django rest framework框架中的视图都可以继承哪些类

[复制代码](javascript:void(0);)

a. 继承APIView（最原始）但定制性比较强

这个类属于rest framework中的顶层类，内部帮助我们实现了只是基本功能：认证、权限、频率控制，

但凡是数据库、分页等操作都需要手动去完成，比较原始。

class GenericAPIView(APIView)

def post(...):

pass

b.继承GenericViewSet（ViewSetMixin，generics.GenericAPIView）

　　首先他的路由就发生变化

如果继承它之后，路由中的as\_view需要填写对应关系

　　在内部也帮助我们提供了一些方便的方法：

　　get\_queryset

　　get\_object

　　get\_serializer

　　get\_serializer\_class

　　get\_serializer\_context

　　filter\_queryset

注意：要设置queryset字段，否则会抛出断言的异常。

代码

只提供增加功能 只继承GenericViewSet

class TestView(GenericViewSet):

　　serialazer\_class = xxx

　　def creat(self,\*args,\*\*kwargs):

　　　　pass # 获取数据并对数据

c. 继承 modelviewset --> 快速快发

　　　　-ModelViewSet(增删改查全有+数据库操作)

　　　　-mixins.CreateModelMixin（只有增）,GenericViewSet

　　　　-mixins.CreateModelMixin,DestroyModelMixin,GenericViewSet

　　对数据库和分页等操作不用我们在编写，只需要继承相关类即可。

示例：只提供增加功能

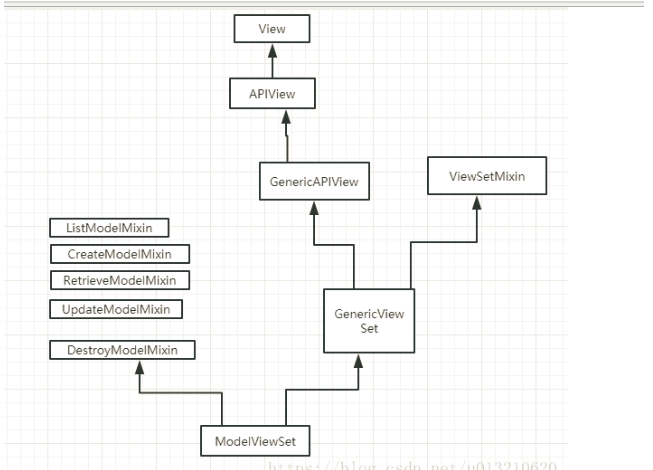
class TestView(mixins.CreateModelMixin,GenericViewSet):

　　　　serializer\_class = XXXXXXX

\*\*\*

　　modelviewset --> 快速开发，复杂点的genericview、apiview

[复制代码](javascript:void(0);)



### 62.简述 django rest framework框架的认证流程。

[复制代码](javascript:void(0);)

- 如何编写？写类并实现authenticators

　　请求进来认证需要编写一个类，类里面有一个authenticators方法，我们可以自定义这个方法，可以定制3类返回值。

　　成功返回元组，返回none为匿名用户，抛出异常为认证失败。

源码流程：请求进来先走dispatch方法，然后封装的request对象会执行user方法，由user触发authenticators认证流程

- 方法中可以定义三种返回值：

- （user,auth），认证成功

- None , 匿名用户

- 异常 ，认证失败

- 流程：

- dispatch

- 再去request中进行认证处理

[复制代码](javascript:void(0);)

### 63.django rest framework如何实现的用户访问频率控制？

[复制代码](javascript:void(0);)

# 对匿名用户，根据用户IP或代理IP作为标识进行记录，为每个用户在redis中建一个列表

{

throttle\_1.1.1.1:[1526868876.497521,152686885.497521...]，

throttle\_1.1.1.2:[1526868876.497521,152686885.497521...]，

throttle\_1.1.1.3:[1526868876.497521,152686885.497521...]，

}

每个用户再来访问时，需先去记录中剔除过期记录，再根据列表的长度判断是否可以继续访问。

'如何封IP'：在防火墙中进行设置

--------------------------------------------------------------------------

# 对注册用户，根据用户名或邮箱进行判断。

{

throttle\_xxxx1:[1526868876.497521,152686885.497521...]，

throttle\_xxxx2:[1526868876.497521,152686885.497521...]，

throttle\_xxxx3:[1526868876.497521,152686885.497521...]，

}

每个用户再来访问时，需先去记录中剔除过期记录，再根据列表的长度判断是否可以继续访问。

\如1分钟：40次，列表长度限制在40，超过40则不可访问

[复制代码](javascript:void(0);)

### 64.Flask框架的优势？

Flask自由、灵活，可扩展性强，透明可控，第三方库的选择面广，

开发时可以结合最流行最强大的Python库，

### 65.Flask框架依赖组件

# 依赖jinja2模板引擎

# 依赖werkzurg协议

### 66.Flask蓝图的作用

# blueprint把实现不同功能的module分开.也就是把一个大的App分割成各自实现不同功能的module.

# 在一个blueprint中可以调用另一个blueprint的视图函数, 但要加相应的blueprint名.

### 67.列举使用的Flask第三方组件？

[复制代码](javascript:void(0);)

# Flask组件

flask-session session放在redis

flask-SQLAlchemy 如django里的ORM操作

flask-migrate 数据库迁移

flask-script 自定义命令

blinker 信号-触发信号

# 第三方组件

Wtforms 快速创建前端标签、文本校验

dbutile 创建数据库连接池

gevnet-websocket 实现websocket

# 自定义Flask组件

自定义auth认证

参考flask-login组件

[复制代码](javascript:void(0);)

### 68.简述Flask上下文管理流程?

[复制代码](javascript:void(0);)

# a、简单来说，falsk上下文管理可以分为三个阶段：

　　1、'请求进来时'：将请求相关的数据放入上下问管理中

　　2、'在视图函数中'：要去上下文管理中取值

　　3、'请求响应'：要将上下文管理中的数据清除

# b、详细点来说：

　　1、'请求刚进来'：

        将request，session封装在RequestContext类中

        app，g封装在AppContext类中

        并通过LocalStack将requestcontext和appcontext放入Local类中

　　2、'视图函数中'：

        通过localproxy--->偏函数--->localstack--->local取值

　　3、'请求响应时'：

        先执行save.session()再各自执行pop(),将local中的数据清除

[复制代码](javascript:void(0);)

### 69.Flask中的g的作用？

# g是贯穿于一次请求的全局变量，当请求进来将g和current\_app封装为一个APPContext类，

# 再通过LocalStack将Appcontext放入Local中，取值时通过偏函数在LocalStack、local中取值；

# 响应时将local中的g数据删除：

### Flask中上下文管理主要涉及到了那些相关的类？并描述类主要作用？

RequestContext #封装进来的请求（赋值给ctx）

AppContext #封装app\_ctx

LocalStack #将local对象中的数据维护成一个栈（先进后出）

Local    #保存请求上下文对象和app上下文对象

### 为什么要Flask把Local对象中的的值stack 维护成一个列表？

# 因为通过维护成列表，可以实现一个栈的数据结构，进栈出栈时只取一个数据，巧妙的简化了问题。

# 还有，在多app应用时，可以实现数据隔离；列表里不会加数据，而是会生成一个新的列表

# local是一个字典，字典里key（stack）是唯一标识，value是一个列表

### Flask中多app应用是怎么完成？

[复制代码](javascript:void(0);)

请求进来时，可以根据URL的不同，交给不同的APP处理。蓝图也可以实现。

    #app1 = Flask('app01')

    #app2 = Flask('app02')

    #@app1.route('/index')

    #@app2.route('/index2')

源码中在DispatcherMiddleware类里调用app2.\_\_call\_\_，

原理其实就是URL分割，然后将请求分发给指定的app。

之后app也按单app的流程走。就是从app.\_\_call\_\_走。

[复制代码](javascript:void(0);)

### 在Flask中实现WebSocket需要什么组件？

gevent-websocket

### wtforms组件的作用？

#快速创建前端标签、文本校验；如django的ModelForm

### Flask框架默认session处理机制？

[复制代码](javascript:void(0);)

# 前提:

不熟的话:记不太清了,应该是……分两个阶段吧

# 创建:

当请求刚进来的时候,会将request和session封装成一个RequestContext()对象,

接下来把这个对象通过LocalStack()放入内部的一个Local()对象中;

　　 因为刚开始 Local 的ctx中session是空的;

　　 所以,接着执行open\_session,将cookie 里面的值拿过来,重新赋值到ctx中

(Local实现对数据隔离,类似threading.local)

# 销毁:

最后返回时执行 save\_session() 将ctx 中的session读出来进行序列化,写到cookie

然后给用户,接着把 ctx pop掉

[复制代码](javascript:void(0);)

### 解释Flask框架中的Local对象和threading.local对象的区别？

[复制代码](javascript:void(0);)

# a.threading.local

作用：为每个线程开辟一块空间进行数据存储(数据隔离)。

问题：自己通过字典创建一个类似于threading.local的东西。

storage = {

4740: {val: 0},

4732: {val: 1},

4731: {val: 3},

}

# b.自定义Local对象

作用：为每个线程(协程)开辟一块空间进行数据存储(数据隔离)。

class Local(object):

def \_\_init\_\_(self):

object.\_\_setattr\_\_(self, 'storage', {})

def \_\_setattr\_\_(self, k, v):

ident = get\_ident()

if ident in self.storage:

self.storage[ident][k] = v

else:

self.storage[ident] = {k: v}

def \_\_getattr\_\_(self, k):

ident = get\_ident()

return self.storage[ident][k]

obj = Local()

def task(arg):

obj.val = arg

obj.xxx = arg

print(obj.val)

for i in range(10):

t = Thread(target=task, args=(i,))

t.start()

[复制代码](javascript:void(0);)

### Flask中 blinker 是什么？

[复制代码](javascript:void(0);)

# flask中的信号blinker

信号主要是让开发者可是在flask请求过程中定制一些行为。

或者说flask在列表里面预留了几个空列表，在里面存东西。

简言之，信号允许某个'发送者'通知'接收者'有事情发生了

@ before\_request有返回值，blinker没有返回值

# 10个信号

request\_started = \_signals.signal('request-started') #请求到来前执行

request\_finished = \_signals.signal('request-finished') #请求结束后执行

before\_render\_template = \_signals.signal('before-render-template')#模板渲染前执行

template\_rendered = \_signals.signal('template-rendered')#模板渲染后执行

got\_request\_exception = \_signals.signal('got-request-exception') #请求执行出现异常时执行

request\_tearing\_down = \_signals.signal('request-tearing-down')#请求执行完毕后自动执行（无论成功与否）

appcontext\_tearing\_down = \_signals.signal('appcontext-tearing-down')# 请求上下文执行完毕后自动执行（无论成功与否）

appcontext\_pushed = \_signals.signal('appcontext-pushed') #请求app上下文push时执行

appcontext\_popped = \_signals.signal('appcontext-popped') #请求上下文pop时执行

message\_flashed = \_signals.signal('message-flashed')#调用flask在其中添加数据时，自动触发

[复制代码](javascript:void(0);)

### SQLAlchemy中的 session和scoped\_session 的区别？

[复制代码](javascript:void(0);)

# Session：

由于无法提供线程共享功能，开发时要给每个线程都创建自己的session

打印sesion可知他是sqlalchemy.orm.session.Session的对象

# scoped\_session：

为每个线程都创建一个session，实现支持线程安全

在整个程序运行的过程当中，只存在唯一的一个session对象。

创建方式:

通过本地线程Threading.Local()

# session=scoped\_session(Session)

创建唯一标识的方法(参考flask请求源码)

[复制代码](javascript:void(0);)

### SQLAlchemy如何执行原生SQL？

# 使用execute方法直接操作SQL语句(导入create\_engin、sessionmaker)

engine=create\_engine('mysql://root:\*\*\*\*\*@127.0.0.1/database?charset=utf8')

DB\_Session = sessionmaker(bind=engine)

session = DB\_Session()

session.execute('alter table mytablename drop column mycolumn ;')

### ORM的实现原理？

# ORM的实现基于一下三点

映射类：描述数据库表结构，

映射文件：指定数据库表和映射类之间的关系

数据库配置文件：指定与数据库连接时需要的连接信息(数据库、登录用户名、密码or连接字符串)

### DBUtils模块的作用？

[复制代码](javascript:void(0);)

# 数据库连接池

使用模式：

1、为每个线程创建一个连接，连接不可控，需要控制线程数

2、创建指定数量的连接在连接池，当线程访问的时候去取，不够了线程排队，直到有人释放(推荐)

---------------------------------------------------------------------------

两种写法：

1、用静态方法装饰器，通过直接执行类的方法来连接使用数据库

2、通过实例化对象，通过对象来调用方法执行语句

https://www.cnblogs.com/ArmoredTitan/p/Flask.html

[复制代码](javascript:void(0);)

以下SQLAlchemy的字段是否正确？如果不正确请更正：

fromdatetime importdatetime

 fromsqlalchemy.ext.declarative

importdeclarative\_base

 fromsqlalchemy importColumn, Integer, String, DateTime

Base =declarative\_base()

classUserInfo(Base):

    \_\_tablename\_\_ ='userinfo'

    id=Column(Integer, primary\_key=True, autoincrement=True)

    name =Column(String(64), unique=True)

    ctime =Column(DateTime, default=datetime.now())

[复制代码](javascript:void(0);)

from datetime import datetime

from sqlalchemy.ext.declarative

import declarative\_base

from sqlalchemy import Column, Integer, String, DateTime

Base = declarative\_base()

class UserInfo(Base):

\_\_tablename\_\_ = 'userinfo'

id = Column(Integer, primary\_key=True, autoincrement=True)

name = Column(String(64), unique=True)

ctime = Column(DateTime, default=datetime.now())

-----------------------------------------------------------------------

不正确：

Ctime字段中参数应为’default=datetime.now’

now 后面不应该加括号，加了的话，字段不会实时更新。

[复制代码](javascript:void(0);)

### SQLAchemy中如何为表设置引擎和字符编码？

sqlalchemy设置编码字符集，一定要在数据库访问的URL上增加'charset=utf8'

否则数据库的连接就不是'utf8'的编码格式

eng=create\_engine('mysql://root:root@localhost:3306/test2?charset=utf8',echo=True)

[复制代码](javascript:void(0);)

1. 设置引擎编码方式为utf8。

engine = create\_engine("mysql+pymysql://root:123456@127.0.0.1:3306/sqldb01?charset=utf8")

2. 设置数据库表编码方式为utf8

class UserType(Base):

\_\_tablename\_\_ = 'usertype'

id = Column(Integer, primary\_key=True)

caption = Column(String(50), default='管理员')

# 添加配置设置编码

\_\_table\_args\_\_ = {

'mysql\_charset':'utf8'

}

这样生成的SQL语句就自动设置数据表编码为utf8了,\_\_table\_args\_\_还可设置存储引擎、外键约束等等信息。

[复制代码](javascript:void(0);)

### SQLAchemy中如何设置联合唯一索引？

通过'UniqueConstraint'字段来设置联合唯一索引

\_\_table\_args=(UniqueConstraint('h\_id','username',name='\_h\_username\_uc'))

#h\_id和username组成联合唯一约束

### 简述Tornado框架的特点。

异步非阻塞+websocket

### 简述Tornado框架中Future对象的作用？

[复制代码](javascript:void(0);)

# 实现异步非阻塞

视图函数yield一个futrue对象，futrue对象默认：

self.\_done = False ，请求未完成

self.\_result = None ，请求完成后返回值，用于传递给回调函数使用。

tornado就会一直去检测futrue对象的\_done是否已经变成True。

如果IO请求执行完毕，自动会调用future的set\_result方法：

self.\_result = result

self.\_done = True

参考：http://www.cnblogs.com/wupeiqi/p/6536518.html（自定义异步非阻塞web框架）

[复制代码](javascript:void(0);)

### Tornado框架中如何编写WebSocket程序？

[复制代码](javascript:void(0);)

Tornado在websocket模块中提供了一个WebSocketHandler类。

这个类提供了和已连接的客户端通信的WebSocket事件和方法的钩子。

当一个新的WebSocket连接打开时，open方法被调用，

而on\_message和on\_close方法，分别在连接、接收到新的消息和客户端关闭时被调用。

此外，WebSocketHandler类还提供了write\_message方法用于向客户端发送消息，close方法用于关闭连接。

[复制代码](javascript:void(0);)

### Tornado中静态文件是如何处理的？ 如： <link href="{{static\_url("commons.css")}}" rel="stylesheet" />

[复制代码](javascript:void(0);)

# settings.py

settings = {

"static\_path": os.path.join(os.path.dirname(\_\_file\_\_), "static"),

# 指定了静态文件的位置在当前目录中的"static"目录下

"cookie\_secret": "61oETzKXQAGaYdkL5gEmGeJJFuYh7EQnp2XdTP1o/Vo=",

"login\_url": "/login",

"xsrf\_cookies": True,

}

经上面配置后

static\_url()自动去配置的路径下找'commons.css'文件

[复制代码](javascript:void(0);)

### Tornado操作MySQL使用的模块？

torndb

torndb是基于mysqldb的再封装，所以使用时要先安装myqldb

### Tornado操作redis使用的模块？

tornado-redis

### 简述Tornado框架的适用场景？

web聊天室，在线投票

### git常见命令作用：

[复制代码](javascript:void(0);)

# git init

初始化，当前所在的文件夹可以被管理且以后版本相关的数据都会存储到.git文件中

# git status

查看当前文件夹以及子目录中文件是否发生变化：

内容修改/新增文件/删除，已经变化的文件会变成红色，已经add的文件会变成绿色

# git add .

给发生变化的文件（贴上一个标签）或 将发生变化的文件放到某个地方，

只写一个句点符就代表把git status中红色的文件全部打上标签

# git commit -m

新增用户登录认证功能以及xxx功能将“绿色”文件添加到版本中

# git log

查看所有版本提交记录，可以获取版本号

# git reset --hard 版本号

将最新的版本回退到更早的版本

# git reflog

回退到之前版本后悔了，再更新到最新或者最新之前的版本

# git reset --hard 版本 回退

[复制代码](javascript:void(0);)

### 简述以下git中stash命令作用以及相关其他命令。

[复制代码](javascript:void(0);)

'git stash'：将当前工作区所有修改过的内容存储到“某个地方”，将工作区还原到当前版本未修改过的状态

'git stash list'：查看“某个地方”存储的所有记录

'git stash clear'：清空“某个地方”

'git stash pop'：将第一个记录从“某个地方”重新拿到工作区（可能有冲突）

'git stash apply'：编号, 将指定编号记录从“某个地方”重新拿到工作区（可能有冲突）

'git stash drop'：编号，删除指定编号的记录

[复制代码](javascript:void(0);)

### git 中 merge 和 rebase命令 的区别。

[复制代码](javascript:void(0);)

merge：

会将不同分支的提交合并成一个新的节点，之前的提交分开显示，

注重历史信息、可以看出每个分支信息，基于时间点,遇到冲突,手动解决,再次提交

rebase：

将两个分支的提交结果融合成线性，不会产生新的节点;

注重开发过程，遇到冲突，手动解决，继续操作

[复制代码](javascript:void(0);)

### 公司如何基于git做的协同开发？

[复制代码](javascript:void(0);)

1、你们公司的代码review分支怎么做？谁来做？

答：组长创建review分支，我们小功能开发完之后，合并到review分支交给老大（小组长）来看，

1.1、你组长不开发代码吗？

他开发代码，但是它只开发核心的东西，任务比较少。

或者抽出时间，我们一起做这个事情

2、你们公司协同开发是怎么协同开发的？

每个人都有自己的分支，阶段性代码完成之后，合并到review，然后交给老大看

--------------------------------------------------------------------------

# 大致工作流程

公司：

下载代码

git clone https://gitee.com/wupeiqi/xianglong.git

或创建目录

cd 目录

git init

git remote add origin https://gitee.com/wupeiqi/xianglong.git

git pull origin maste

创建dev分支

git checkout dev

git pull origin dev

继续写代码

git add .

git commit -m '提交记录'

git push origin dev

回家：

拉代码：

git pull origin dev

继续写：

继续写代码

git add .

git commit -m '提交记录'

git push origin dev

[复制代码](javascript:void(0);)

### 如何基于git实现代码review？

https://blog.csdn.net/june\_y/article/details/50817993

### git如何实现v1.0 、v2.0 等版本的管理？

在命令行中，使用“git tag –a tagname –m “comment”可以快速创建一个标签。

需要注意，命令行创建的标签只存在本地Git库中，还需要使用Git push –tags指令发布到服务器的Git库中

### 什么是gitlab

gitlab是公司自己搭建的项目代码托管平台

### github和gitlab的区别？

1、gitHub是一个面向开源及私有软件项目的托管平台

(创建私有的话，需要购买，最低级的付费为每月7刀，支持5个私有项目)

2、gitlab是公司自己搭建的项目托管平台

### 如何为github上牛逼的开源项目贡献代码？

[复制代码](javascript:void(0);)

1、fork需要协作项目

2、克隆/关联fork的项目到本地

3、新建分支（branch）并检出（checkout）新分支

4、在新分支上完成代码开发

5、开发完成后将你的代码合并到master分支

6、添加原作者的仓库地址作为一个新的仓库地址

7、合并原作者的master分支到你自己的master分支,用于和作者仓库代码同步

8、push你的本地仓库到GitHub

9、在Github上提交 pull requests

10、等待管理员（你需要贡献的开源项目管理员）处理

[复制代码](javascript:void(0);)

### git中 .gitignore文件的作用

一般来说每个Git项目中都需要一个“.gitignore”文件，

这个文件的作用就是告诉Git哪些文件不需要添加到版本管理中。

实际项目中，很多文件都是不需要版本管理的，比如Python的.pyc文件和一些包含密码的配置文件等等。

### 什么是敏捷开发？

'敏捷开发'：是一种以人为核心、迭代、循序渐进的开发方式。

它并不是一门技术，而是一种开发方式，也就是一种软件开发的流程。

它会指导我们用规定的环节去一步一步完成项目的开发。

因为它采用的是迭代式开发，所以这种开发方式的主要驱动核心是人

### 简述 jenkins 工具的作用?

'Jenkins'是一个可扩展的持续集成引擎。

主要用于：

持续、自动地构建/测试软件项目。

监控一些定时执行的任务。

### 公司如何实现代码发布？

[nginx+uwsgi+django](https://www.cnblogs.com/wupeiqi/articles/8591782.html)

### 简述 RabbitMQ、Kafka、ZeroMQ的区别？

https://blog.csdn.net/zhailihua/article/details/7899006

### [RabbitMQ](http://www.cnblogs.com/yuanchenqi/articles/8507109.html" \t "_blank)如何在消费者获取任务后未处理完前就挂掉时，保证数据不丢失？

[复制代码](javascript:void(0);)

为了预防消息丢失，rabbitmq提供了ack

即工作进程在收到消息并处理后，发送ack给rabbitmq，告知rabbitmq这时候可以把该消息从队列中删除了。

如果工作进程挂掉 了，rabbitmq没有收到ack，那么会把该消息 重新分发给其他工作进程。

不需要设置timeout，即使该任务需要很长时间也可以处理。

ack默认是开启的，工作进程显示指定了no\_ack=True

[复制代码](javascript:void(0);)

### RabbitMQ如何对消息做持久化？

[复制代码](javascript:void(0);)

1、创建队列和发送消息时将设置durable=Ture，如果在接收到消息还没有存储时，消息也有可能丢失，就必须配置publisher confirm

channel.queue\_declare(queue='task\_queue', durable=True)

2、返回一个ack，进程收到消息并处理完任务后，发给rabbitmq一个ack表示任务已经完成，可以删除该任务

3、镜像队列：将queue镜像到cluster中其他的节点之上。

在该实现下，如果集群中的一个节点失效了，queue能自动地切换到镜像中的另一个节点以保证服务的可用性

[复制代码](javascript:void(0);)

### RabbitMQ如何控制消息被消费的顺序？

默认消息队列里的数据是按照顺序被消费者拿走，

例如：消费者1 去队列中获取奇数序列的任务，消费者2去队列中获取偶数序列的任务。

channel.basic\_qos(prefetch\_count=1)

表示谁来谁取，不再按照奇偶数排列（同时也保证了公平的消费分发）

### 以下RabbitMQ的exchange type分别代表什么意思？如：fanout、direct、topic。

[复制代码](javascript:void(0);)

amqp协议中的核心思想就是生产者和消费者隔离，生产者从不直接将消息发送给队列。

生产者通常不知道是否一个消息会被发送到队列中，只是将消息发送到一个交换机。

先由Exchange来接收，然后Exchange按照特定的策略转发到Queue进行存储。

同理，消费者也是如此。Exchange 就类似于一个交换机，转发各个消息分发到相应的队列中。

--------------------------------------------------

type=fanout 类似发布者订阅者模式，会为每一个订阅者创建一个队列，而发布者发布消息时，会将消息放置在所有相关队列中

type=direct 队列绑定关键字，发送者将数据根据关键字发送到消息exchange，exchange根据 关键字 判定应该将数据发送至指定队列。

type=topic 队列绑定几个模糊的关键字，之后发送者将数据发送到exchange，exchange将传入”路由值“和 ”关键字“进行匹配，匹配成功，  
则将数据发送到指定队列。

---------------------------------------------------

发送者路由值 队列中

old.boy.python old.\* -- 不匹配 \*表示匹配一个

old.boy.python old.# -- 匹配 #表示匹配0个或多个

[复制代码](javascript:void(0);)

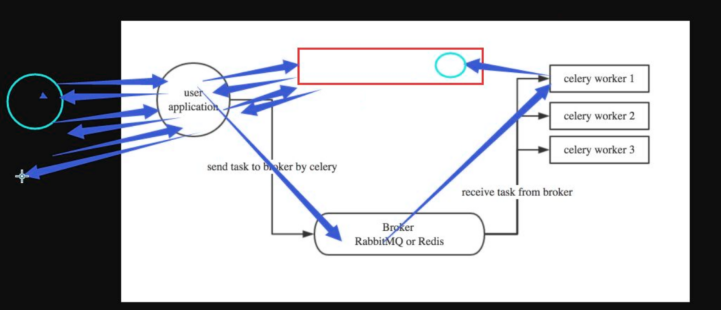
### 简述 celery 是什么以及应用场景？

# Celery是由Python开发的一个简单、灵活、可靠的处理大量任务的分发系统，

# 它不仅支持实时处理也支持任务调度。

# http://www.cnblogs.com/wupeiqi/articles/8796552.html

### 简述celery运行机制。



### celery如何实现定时任务？

[复制代码](javascript:void(0);)

# celery实现定时任务

启用Celery的定时任务需要设置CELERYBEAT\_SCHEDULE 。

CELERYBEAT\_SCHEDULE='djcelery.schedulers.DatabaseScheduler'#定时任务

'创建定时任务'

# 通过配置CELERYBEAT\_SCHEDULE：

#每30秒调用task.add

from datetime import timedelta

CELERYBEAT\_SCHEDULE = {

'add-every-30-seconds': {

'task': 'tasks.add',

'schedule': timedelta(seconds=30),

'args': (16, 16)

},

}

[复制代码](javascript:void(0);)

### 简述 celery多任务结构目录

[复制代码](javascript:void(0);)

pro\_cel

├── celery\_tasks # celery相关文件夹

│ ├── celery.py # celery连接和配置相关文件

│ └── tasks.py # 所有任务函数

├── check\_result.py # 检查结果

└── send\_task.py # 触发任务

[复制代码](javascript:void(0);)

### celery中装饰器 @app.task 和 @shared\_task的区别？

# 一般情况使用的是从celeryapp中引入的app作为的装饰器：@app.task

# django那种在app中定义的task则需要使用@shared\_task

### 简述 [requests](http://www.cnblogs.com/linhaifeng/articles/7785043.html)模块的作用及基本使用？

[复制代码](javascript:void(0);)

# 作用：

使用requests可以模拟浏览器的请求

# 常用参数：

url、headers、cookies、data

json、params、proxy

# 常用返回值：

content

iter\_content

text

encoding="utf-8"

cookie.get\_dict()

[复制代码](javascript:void(0);)

### 简述 beautifulsoup模块的作用及基本使用？

# BeautifulSoup

用于从HTML或XML文件中提取、过滤想要的数据形式

#常用方法

解析：html.parser 或者 lxml（需要下载安装）

find、find\_all、text、attrs、get

### 简述 seleninu模块的作用及基本使用?

Selenium是一个用于Web应用程序测试的工具，

他的测试直接运行在浏览器上，模拟真实用户，按照代码做出点击、输入、打开等操作

爬虫中使用他是为了解决requests无法解决javascript动态问题

### scrapy框架中各组件的工作流程？

[复制代码](javascript:void(0);)

#1、生成初始的Requests来爬取第一个URLS，并且标识一个回调函数

第一个请求定义在start\_requests()方法内默认从start\_urls列表中获得url地址来生成Request请求，

默认的回调函数是parse方法。回调函数在下载完成返回response时自动触发

#2、在回调函数中，解析response并且返回值

返回值可以4种：

a、包含解析数据的字典

b、Item对象

c、新的Request对象（新的Requests也需要指定一个回调函数）

d、或者是可迭代对象（包含Items或Request）

#3、在回调函数中解析页面内容

通常使用Scrapy自带的Selectors，但很明显你也可以使用Beutifulsoup，lxml或其他你爱用啥用啥。

#4、最后，针对返回的Items对象将会被持久化到数据库

通过Item Pipeline组件存到数据库

或者导出到不同的文件（通过Feed exports）

http://www.cnblogs.com/wupeiqi/articles/6229292.html

[复制代码](javascript:void(0);)

### 在scrapy框架中如何设置代理（两种方法）？

[复制代码](javascript:void(0);)

方式一：内置添加代理功能

# -\*- coding: utf-8 -\*-

import os

import scrapy

from scrapy.http import Request

class ChoutiSpider(scrapy.Spider):

name = 'chouti'

allowed\_domains = ['chouti.com']

start\_urls = ['https://dig.chouti.com/']

def start\_requests(self):

os.environ['HTTP\_PROXY'] = "http://192.168.11.11"

for url in self.start\_urls:

yield Request(url=url,callback=self.parse)

def parse(self, response):

print(response)

方式二：自定义下载中间件

import random

import base64

import six

def to\_bytes(text, encoding=None, errors='strict'):

"""Return the binary representation of `text`. If `text`

is already a bytes object, return it as-is."""

if isinstance(text, bytes):

return text

if not isinstance(text, six.string\_types):

raise TypeError('to\_bytes must receive a unicode, str or bytes '

'object, got %s' % type(text).\_\_name\_\_)

if encoding is None:

encoding = 'utf-8'

return text.encode(encoding, errors)

class MyProxyDownloaderMiddleware(object):

def process\_request(self, request, spider):

proxy\_list = [

{'ip\_port': '111.11.228.75:80', 'user\_pass': 'xxx:123'},

{'ip\_port': '120.198.243.22:80', 'user\_pass': ''},

{'ip\_port': '111.8.60.9:8123', 'user\_pass': ''},

{'ip\_port': '101.71.27.120:80', 'user\_pass': ''},

{'ip\_port': '122.96.59.104:80', 'user\_pass': ''},

{'ip\_port': '122.224.249.122:8088', 'user\_pass': ''},

]

proxy = random.choice(proxy\_list)

if proxy['user\_pass'] is not None:

request.meta['proxy'] = to\_bytes("http://%s" % proxy['ip\_port'])

encoded\_user\_pass = base64.encodestring(to\_bytes(proxy['user\_pass']))

request.headers['Proxy-Authorization'] = to\_bytes('Basic ' + encoded\_user\_pass)

else:

request.meta['proxy'] = to\_bytes("http://%s" % proxy['ip\_port'])

配置：

DOWNLOADER\_MIDDLEWARES = {

# 'xiaohan.middlewares.MyProxyDownloaderMiddleware': 543,

}

[复制代码](javascript:void(0);)

### scrapy框架中如何实现大文件的下载？

[复制代码](javascript:void(0);)

from twisted.web.client import Agent, getPage, ResponseDone, PotentialDataLoss

from twisted.internet import defer, reactor, protocol

from twisted.web.\_newclient import Response

from io import BytesIO

class \_ResponseReader(protocol.Protocol):

def \_\_init\_\_(self, finished, txresponse, file\_name):

self.\_finished = finished

self.\_txresponse = txresponse

self.\_bytes\_received = 0

self.f = open(file\_name, mode='wb')

def dataReceived(self, bodyBytes):

self.\_bytes\_received += len(bodyBytes)

# 一点一点的下载

self.f.write(bodyBytes)

self.f.flush()

def connectionLost(self, reason):

if self.\_finished.called:

return

if reason.check(ResponseDone):

# 下载完成

self.\_finished.callback((self.\_txresponse, 'success'))

elif reason.check(PotentialDataLoss):

# 下载部分

self.\_finished.callback((self.\_txresponse, 'partial'))

else:

# 下载异常

self.\_finished.errback(reason)

self.f.close()

[复制代码](javascript:void(0);)

### scrapy中如何实现限速？

http://scrapy-chs.readthedocs.io/zh\_CN/1.0/topics/autothrottle.html

### [scrapy中如何实现暂停爬虫？](https://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/jobs.html?highlight=item" \t "_blank)

[复制代码](javascript:void(0);)

# 有些情况下，例如爬取大的站点，我们希望能暂停爬取，之后再恢复运行。

# Scrapy通过如下工具支持这个功能:

一个把调度请求保存在磁盘的调度器

一个把访问请求保存在磁盘的副本过滤器[duplicates filter]

一个能持续保持爬虫状态(键/值对)的扩展

Job 路径

要启用持久化支持，你只需要通过 JOBDIR 设置 job directory 选项。

这个路径将会存储所有的请求数据来保持一个单独任务的状态(例如：一次spider爬取(a spider run))。

必须要注意的是，这个目录不允许被不同的spider 共享，甚至是同一个spider的不同jobs/runs也不行。

也就是说，这个目录就是存储一个 单独 job的状态信息。

[复制代码](javascript:void(0);)

### scrapy中如何进行[自定制命令](http://www.cnblogs.com/wupeiqi/articles/6229292.html)？

[复制代码](javascript:void(0);)

在spiders同级创建任意目录，如：commands

在其中创建'crawlall.py'文件（此处文件名就是自定义的命令）

from scrapy.commands import ScrapyCommand

from scrapy.utils.project import get\_project\_settings

class Command(ScrapyCommand):

requires\_project = True

def syntax(self):

return '[options]'

def short\_desc(self):

return 'Runs all of the spiders'

def run(self, args, opts):

spider\_list = self.crawler\_process.spiders.list()

for name in spider\_list:

self.crawler\_process.crawl(name, \*\*opts.\_\_dict\_\_)

self.crawler\_process.start()

在'settings.py'中添加配置'COMMANDS\_MODULE = '项目名称.目录名称''

在项目目录执行命令：'scrapy crawlall'

[复制代码](javascript:void(0);)

### scrapy中如何实现的记录爬虫的深度？

[复制代码](javascript:void(0);)

'DepthMiddleware'是一个用于追踪每个Request在被爬取的网站的深度的中间件。

其可以用来限制爬取深度的最大深度或类似的事情。

'DepthMiddleware'可以通过下列设置进行配置(更多内容请参考设置文档):

'DEPTH\_LIMIT':爬取所允许的最大深度，如果为0，则没有限制。

'DEPTH\_STATS':是否收集爬取状态。

'DEPTH\_PRIORITY':是否根据其深度对requet安排优先

[复制代码](javascript:void(0);)

### scrapy中的pipelines工作原理？

Scrapy 提供了 pipeline 模块来执行保存数据的操作。

在创建的 Scrapy 项目中自动创建了一个 pipeline.py 文件，同时创建了一个默认的 Pipeline 类。

我们可以根据需要自定义 Pipeline 类，然后在 settings.py 文件中进行配置即可

### scrapy的pipelines如何丢弃一个item对象？

通过raise DropItem**()**方法

### 简述scrapy中爬虫中间件和下载中间件的作用？1

http://www.cnblogs.com/wupeiqi/articles/6229292.html

### scrapy-redis组件的作用？

实现了分布式爬虫，url去重、调度器、数据持久化

'scheduler'调度器

'dupefilter'URL去重规则（被调度器使用）

'pipeline'数据持久化

### scrapy-redis组件中如何实现的任务的去重？

[复制代码](javascript:void(0);)

a. 内部进行配置，连接Redis  
b.去重规则通过redis的集合完成，集合的Key为：  
   key **=**defaults.DUPEFILTER\_KEY **% {'timestamp':**int**(**time.time**())}**默认配置：  
      DUPEFILTER\_KEY **= 'dupefilter:%(timestamp)s'**c.去重规则中将url转换成唯一标示，然后在redis中检查是否已经在集合中存在  
   **from**scrapy.utils **import**request  
   **from**scrapy.http **import**Request  
   req **=**Request**(**url**='http://www.cnblogs.com/wupeiqi.html')**result **=**request.request\_fingerprint**(**req**)**print**(**result**)**# 8ea4fd67887449313ccc12e5b6b92510cc53675c

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

scrapy和scrapy-redis的去重规则（源码）

1. scrapy中去重规则是如何实现？

class RFPDupeFilter(BaseDupeFilter):

"""Request Fingerprint duplicates filter"""

def \_\_init\_\_(self, path=None, debug=False):

self.fingerprints = set()

@classmethod

def from\_settings(cls, settings):

debug = settings.getbool('DUPEFILTER\_DEBUG')

return cls(job\_dir(settings), debug)

def request\_seen(self, request):

# 将request对象转换成唯一标识。

fp = self.request\_fingerprint(request)

# 判断在集合中是否存在，如果存在则返回True，表示已经访问过。

if fp in self.fingerprints:

return True

# 之前未访问过，将url添加到访问记录中。

self.fingerprints.add(fp)

def request\_fingerprint(self, request):

return request\_fingerprint(request)

2. scrapy-redis中去重规则是如何实现？

class RFPDupeFilter(BaseDupeFilter):

"""Redis-based request duplicates filter.

This class can also be used with default Scrapy's scheduler.

"""

logger = logger

def \_\_init\_\_(self, server, key, debug=False):

# self.server = redis连接

self.server = server

# self.key = dupefilter:123912873234

self.key = key

@classmethod

def from\_settings(cls, settings):

# 读取配置，连接redis

server = get\_redis\_from\_settings(settings)

# key = dupefilter:123912873234

key = defaults.DUPEFILTER\_KEY % {'timestamp': int(time.time())}

debug = settings.getbool('DUPEFILTER\_DEBUG')

return cls(server, key=key, debug=debug)

@classmethod

def from\_crawler(cls, crawler):

return cls.from\_settings(crawler.settings)

def request\_seen(self, request):

fp = self.request\_fingerprint(request)

# This returns the number of values added, zero if already exists.

# self.server=redis连接

# 添加到redis集合中：1，添加工程；0，已经存在

added = self.server.sadd(self.key, fp)

return added == 0

def request\_fingerprint(self, request):

return request\_fingerprint(request)

def close(self, reason=''):

self.clear()

def clear(self):

"""Clears fingerprints data."""

self.server.delete(self.key)

[复制代码](javascript:void(0);)

### scrapy-redis的调度器如何实现任务的深度优先和广度优先？....

### [简述 vitualenv](https://blog.csdn.net/u012734441/article/details/55044025/" \t "_blank)及应用场景?

'vitualenv'是一个独立的python虚拟环境

如：

当前项目依赖的是一个版本，但是另一个项目依赖的是另一个版本，这样就会造成依赖冲突，

而virtualenv就是解决这种情况的，virtualenv通过创建一个虚拟化的python运行环境，

将我们所需的依赖安装进去的，不同项目之间相互不干扰

### 简述 pipreqs 及应用场景？

可以通过对项目目录扫描，自动发现使用了那些类库，并且自动生成依赖清单。

pipreqs ./ 生成requirements.txt

### 在Python中使用过什么代码检查工具？

1）PyFlakes：静态检查Python代码逻辑错误的工具。

2）Pep8： 静态检查PEP8编码风格的工具。

3）NedBatchelder’s McCabe script：静态分析Python代码复杂度的工具。

Python代码分析工具：PyChecker、Pylint

### 简述 saltstack、ansible、fabric、puppet工具的作用？

### B Tree和B+ Tree的区别？

1.B树中同一键值不会出现多次，并且有可能出现在叶结点，也有可能出现在非叶结点中。

而B+树的键一定会出现在叶结点中，并有可能在非叶结点中重复出现，以维持B+树的平衡。

2.因为B树键位置不定，且在整个树结构中只出现一次，

### 请列举常见排序并通过代码实现任意三种。

冒泡/选择/插入/快排

https://www.cnblogs.com/Liqiongyu/p/5911613.html

http://www.cnblogs.com/feixuelove1009/p/6143539.html

### 请列举常见查找并通过代码实现任意三种。

[无序查找、二分查找、插值查找](http://www.cnblogs.com/feixuelove1009/p/6148357.html)

### 请列举你熟悉的[设计模式](http://www.cnblogs.com/haiyan123/p/8407453.html)？

工厂模式/单例模式等

### 有没有刷过leetcode？

leetcode是个题库，里面有多很编程题目，可以在线编译运行。

https://leetcode-cn.com/problemset/all/

### [列举熟悉的的Linux命令。](https://www.cnblogs.com/jassin-du/p/8973891.html" \t "_blank)

[复制代码](javascript:void(0);)

1创建目录

mkdir /data

cd /

mkdir data

2：查看目录

ls

ls -l 显示详细信息

[复制代码](javascript:void(0);)

### 公司线上服务器是什么系统？

Linux/Centos

### 解释 PV、UV 的含义？

PV访问量（Page View），即页面访问量，每打开一次页面PV计数+1，刷新页面也是。

UV访问数（Unique Visitor）指独立访客访问数，一台电脑终端为一个访客。

### 解释 QPS的含义？

'QPS(Query Per Second)'

每秒查询率，是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准

### uwsgi和wsgi的区别？

wsgi是一种通用的接口标准或者接口协议，实现了python web程序与服务器之间交互的通用性。

uwsgi:同WSGI一样是一种通信协议

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'是实现了uwsgi和WSGI两种协议的Web服务器，负责响应python的web请求。

### supervisor的作用？

[复制代码](javascript:void(0);)

# Supervisor：

是一款基于Python的进程管理工具，可以很方便的管理服务器上部署的应用程序。

是C/S模型的程序，其服务端是supervisord服务,客户端是supervisorctl命令

# 主要功能:

1 启动、重启、关闭包括但不限于python进程。

2 查看进程的运行状态。

3 批量维护多个进程。

[复制代码](javascript:void(0);)

### 什么是反向代理？

正向代理代理客户端(客户端找哟个代理去访问服务器，服务器不知道你的真实IP)

反向代理代理服务器(服务器找一个代理给你响应，你不知道服务器的真实IP)

### 简述SSH的整个过程。

SSH 为 'Secure Shell' 的缩写，是建立在应用层基础上的安全协议。

SSH 是目前较可靠，为远程登录会话和其他网络服务提供的安全性协议。

利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

### 有问题都去那些找解决方案？

起初是百度，发现搜到的答案不精准，净广告

转战谷歌，但墙了；捣鼓怎么FQ

还会去知乎、stackoverfloow、必应、思否(segmentfault)

### 是否有关注什么技术类的公众号？

python之禅(主要专注Python相关知识，作者：刘志军)

码农翻身(主要是Java的，但不光是java，涵盖面很广，作者：刘欣)

实验楼(在线练项目)

and so on

### 最近在研究什么新技术？

Numpy

pandas(金融量化分析、聚宽)

百度AI

图灵API

智能玩具

### 是否了解过领域[驱动模型](https://www.cnblogs.com/netfocus/archive/2011/10/10/2204949.html)？

Domain-Driven Design

## 二进制与十进制之间的转换

1、十进制 与 二进制之间的转换  
    (1)、十进制转换为二进制，分为整数部分和小数部分

### 整数部分

[复制代码](javascript:void(0);)

方法：除2取余法，即每次将整数部分除以2，余数为该位权上的数，而商继续除以2，余数又为上一个位权上的数。

这个步骤一直持续下去，直到商为0为止，最后读数时候，从最后一个余数读起，一直到最前面的一个余数。下面举例：

例：将十进制的168转换为二进制

得出结果 将十进制的168转换为二进制，（10101000）2

168 / 2 = 84 -- 0

84 / 2 = 42 -- 0

42 / 2 = 21 -- 0

21 / 2 = 10 -- 1

10 / 2 = 5 -- 0

5 / 2 = 2 -- 1

2 / 2 = 1 -- 0

1 / 2 = 0 -- 1

二进制（从后往前读）： 10101000

[复制代码](javascript:void(0);)

### ****小数部分****

方法：乘2取整法，即将小数部分乘以2，然后取整数部分，剩下的小数部分继续乘以2，然后取整数部分，  
剩下的小数部分又乘以2，一直取到小数部分为零为止。如果永远不能为零，就同十进制数的四舍五入一样，  
按照要求保留多少位小数时，就根据后面一位是0还是1，取舍，如果是零，舍掉，如果是1，向入一位。  
换句话说就是0舍1入。读数要从前面的整数读到后面的整数

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 小数部分

### 二进制转换为十进制 (不分整数和小数部分)

方法：按权相加法，即将二进制每位上的数乘以权，然后相加之和即是十进制数。

[复制代码](javascript:void(0);)

例：将二进制数101.101转换为十进制数。

得出结果：（101.101）2=(5.625)10

在做二进制转换成十进制需要注意的是

1）要知道二进制每位的权值

2）要能求出每位的值 101.101 转换为十进制

整数部分：2^2 + 2^0 = 5

小数部分：2^(-1) + 2^(-3) = 1/2 + 1/8 = 0.5 + 0.125 = 0.625

十进制： 2^2 + 2^0 + 2^(-1) + 2^(-3) = 5.625

### 各类型的转化

1、Int 与Str之间的转化，转化结果以及转化条件

　　-str(int型)可以转化为str 该过程没有条件

　　-字符串转化为数字必须全部由数字组成   才能通过int(str型)转换

2、Int 与 bool 之间如何转化，转化结果以及转化条件

　　- int型转化为bool没有条件，除了0转换为False以外其他数字都转换为True  bool(int型的数)

　　- int（False）= 0; int(True) = 1

3:、str 与 bool 之间如何转化，转化结果以及转化条件

　　- str 转化为bool没有条件，除了''转化为False其他的字符串都转为True    bool(srr型)

　　- str(True)='True'   str(False)='Flase'

4、str 与 list 能否转化？如何转化？

　　-可以 str>>list  1.用split切割  2.list(str字符串)

　　-可以 list>>str  1.用join+ 迭代   2. str(list列表)

### 能转化为False

0，“”，{}，[],（），set（）

### 字符串与列表互转

l = 'ab\_c'

print(list('ab\_c'),type(list('ab\_c')))

### 列表转字符串

使用 join

l1 = ['1','2','3']

str = ''.join(l1)

print(str,type(str))

li = ['alex','rain']

temp = ''.join(li)

print(temp,type(temp))

### 元组与列表互转

# 元组转为列表 加list

tu = (1,2,3,)

l = list(tu)

print(l,type(l))

# #

# # # 列表转为元组加tuple

li = [4,5,6]

print(tuple(li),type(tuple(li)))

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 函数

函数参数传递的是什么？ 引用、内存地址

[复制代码](javascript:void(0);)

#魔性的用法:默认参数尽量避免使用可变数据类型

def func(a1,a2=[]):

a2.append(a1)

return a2

l1 = func(1)

print(l1) # [1,]

l2 = func(3,[])

print(l2) # [3, ]

l3 = func(4)

print(l3) # [1,4]

默认参数只会被执行一次：第一次调用函数时，默认参数被初始化为[]，以后每次调用时都会使用已经初始化的 []。

def func(a1,a2=[]):

a2.append(a1)

print(a2)

func(1) # [1,]

func(3,[]) # [3,]

func(4) # [1,4]

func([]) #[1]

def func(a1,a2=[]):

a2.append(a1)

return a2

l1 = func(1) # l1=[1,4]

l2 = func(3,[]) # l2=[3,]

l3 = func(4) # l3=[1,4]

print(l2)

print(l1)

print(l3)

[复制代码](javascript:void(0);)

### 三元运算及应用场景

[复制代码](javascript:void(0);)

应用场景：简化if语句

# 关于三元运算

# 结果+　if + 条件 + else + 结果

result='gt' if 1>3 else 'lt'

print(result) # lt

val = 'aaa' if 1 == 1 else 'bbb'

# aaa

# 理解：如果条件为真，把if前面的值赋值给变量，否则把else后面的值赋值给变量。

lambda 表达式

应用场景：简化函数

temp = lambda x,y:x+y

print(temp(4,10)) # 14

可替代：

def foo(x,y):

return x+y

print(foo(4,10)) # 14

[复制代码](javascript:void(0);)

### 闭包

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

# 闭包:内部的函数引用了外部函数的变量

def func(arg):

def inner():

return arg + 1

return inner

v = func(111)

[复制代码](javascript:void(0);)

### 生成器表达式，列表表达式

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

# 生成式

val = [i + 100 for i in range(1,11)]

print(val)

# [101, 102, 103, 104, 105, 106, 107, 108, 109, 110]

# 列表表达式

val = [lambda x:x+i for i in range(10)]

# 问题1：val[4]是什么？

# print(val[4]) # <function <listcomp>.<lambda> at 0x0000000002208BF8> 函数

# 2：v = val[4](1) 是什么？

# 10

# 生成器

val = (lambda x:x+i for i in range(10))

# print(val,type(val)) 生成器

( i % 2 for i in range(10) )

print(( i % 2 for i in range(10) ))

# <generator object <genexpr> at 0x00000000020CEEB8> 生成器

# 在Python中，有一种自定义迭代器的方式，称为生成器（Generator）。

# 定义生成器的两种方式：

# 1.创建一个generator，只要把一个列表生成式的[]改成()，就创建了一个generator：

# generator保存的是算法，每次调用next()，就计算出下一个元素的值，直到计算到最后一个元素，

没有更多的元素时，抛出StopIteration的错误。

# 2.定义generator的另一种方法。如果一个函数定义中包含yield关键字，那么这个函数就不再是一个普通函数，

而是一个generator

[复制代码](javascript:void(0);)

### 常见内置函数：

**- map - filter**

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

**isinstance - type**

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

isinstance() 函数来判断一个对象是否是一个已知的类型，类似 type()。

isinstance() 与 type() 区别：

type() 不会认为子类是一种父类类型，不考虑继承关系。

isinstance() 会认为子类是一种父类类型，考虑继承关系。

如果要判断两个类型是否相同推荐使用 isinstance()。

[复制代码](javascript:void(0);)

**zip**

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。

>>>a = [1,2,3]

>>> b = [4,5,6]

>>> c = [4,5,6,7,8]

>>> zipped = zip(a,b) # 打包为元组的列表

[(1, 4), (2, 5), (3, 6)]

>>> zip(a,c) # 元素个数与最短的列表一致

[(1, 4), (2, 5), (3, 6)]

>>> zip(\*zipped) # 与 zip 相反，可理解为解压，返回二维矩阵式

[(1, 2, 3), (4, 5, 6)]

[复制代码](javascript:void(0);)

**reduce**

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

reduce() 函数会对参数序列中元素进行累积。

用传给reduce中的函数 function（有两个参数）先对集合中的第1、2个元素进行操作，得到的结果再与第三个数据用function函数运算，最后得到一个结果。

>>>def add(x, y) : # 两数相加

... return x + y

...

>>> reduce(add, [1,2,3,4,5]) # 计算列表和：1+2+3+4+5

>>> reduce(lambda x, y: x+y, [1,2,3,4,5]) # 使用 lambda 匿名函数

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 生成器、迭代器、装饰器、可迭代对象

### 生成器

包括含有yield这个关键字，生成器也是迭代器，调动next把函数变成迭代器。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

应用场景：

range/xrange

- py2： range(1000000) ,会立即创建，xrange(1000000)生成器

- py3：range（10000000）生成器

- redis获取值

- redis获取值

conn = Redis(...)

　　　　def hscan\_iter(self, name, match=None, count=None):

　　　　　　"""

　　　　　　Make an iterator using the HSCAN command so that the client doesn't

　　　　　　need to remember the cursor position.

　　　　　　``match`` allows for filtering the keys by pattern

　　　　　　``count`` allows for hint the minimum number of returns

　　　　　　"""

　　　　　　cursor = '0'

　　　　　　while cursor != 0:

　　　　　　　　# 去redis中获取数据：12

　　　　　　　　# cursor，下一次取的位置

　　　　　　　　# data：本地获取的12条数数据

　　　　　　　　cursor, data = self.hscan(name, cursor=cursor,match=match, count=count)

　　　　　　　　for item in data.items():

　　　　　　　　　　yield item

stark组件

def index(request):

　　　　data = [

　　　　　　{'k1':1,'name':'alex'},

　　　　　　{'k1':2,'name':'老男孩'},

　　　　　　{'k1':3,'name':'小男孩'},

　　　　]

　　　　new\_data = []

　　　　for item in data:

　　　　　　item['email'] = "xxx@qq.com"

　　　　　　new\_data.append(item)

　　　　return render(request,'xx.html',{'data':new\_data})

　　　　　　new\_data.append(item)

　　　　return render(request,'xx.html',{'data':new\_data})

[复制代码](javascript:void(0);)

### 迭代器

含有\_\_iter\_\_和\_\_next\_\_方法 (包含\_\_next\_\_方法的可迭代对象就是迭代器)

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

### 可迭代对象

内部实现\_\_iter\_\_方法且返回一个迭代器。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 应用场景

### 装饰器

调用装饰器其实是一个闭包函数，为其他函数添加附加功能，不修改被修改的源代码和不修改被修饰的方式，装饰器的返回值也是一个函数对象。  
比如：插入日志、性能测试、事物处理、缓存、权限验证等，有了装饰器，就可以抽离出大量与函数功能本身无关的雷同代码并继续重用。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

问题：什么是装饰器？

　　在对原函数不进行修改时，在函数执行前和执行后添加功能

问题：手写装饰器

import functools

def warpper(func):

@functools.wraps(func) #不改变原函数属性

def inner(\*args,\*\*kwargs):

#执行函数前

return func(\*args,\*\*kwargs)

#执行函数后

return inner

# 1. 执行wapper函数，并将被装饰的函数当做参数。 wapper(index)

# 2. 将第一步的返回值，重新赋值给 新index = wapper(老index)

@warpper #index=warpper(index)

def index(x):

return x+100

问题：应用场景

　　　　django: csrf 内置认证、缓存

　　　　flask： 路由、before\_request

带参数装饰器：flask：路由

　　　　　　　CBV as\_view()

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 偏函数

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 谈谈面向对象认识

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

### 深度优先及广度优先

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

### 给一个路径，找到该路径下所有文件

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 2中方法

### 创建、删除文件

1 # 创建一个文件

2 open("chao.txt","w",encoding="utf-8")

3 import os

#删除文件

4 os.remove("chao.txt")

### 第三方软件安装

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 2种

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 网络编程

### C/S架构

C/S:Client（客户端）与Server（服务端）即：客户端与服务端架构，这种架构也是从用户层面（也可以是物理层面）来划分的。

这里的客户端一般泛指应用程序的.exe，程序需要先安装后，才能运行在用户的电脑上，对用户的电脑操作系统环境依赖较大。

### B/S架构

指的是Brosver端（浏览器端）与Sever端，浏览器端与服务器端架构，这种架构是从用户层面来划分的。

隶属与C/S架构的

Browser浏览器，其实也是一种Client客户端，只需在浏览器上通过HTTP请求服务器端相关的资源（网页资源），客户端Browser浏览器就能进行增删改查。

优点：统一了所有应用程序的入口、方便、轻量级

### MAC地址

mac地址：在网卡上的mac地址是网卡的生产厂商给定的，每生产一个网卡，就要附加一个mac地址（全球唯一）

### IP地址

一个4位点分十进制组成的

### 如何通过ip地址确定Mac地址

arp协议：把ip地址转换为mac地址

### 两个电脑是如何传递信息

信息过来 ----> 路由器-----> 路由表 ---> 交换机 ---> 交换机通过广播找所属ip的电脑 ----> 所属ip的电脑通过单播的形式回答

路由表：里面有网关ip的网段信息

### 交换机与路由器区别

1：交换机：是负责内网里面的数据传递（arp协议）根据MAC地址寻址

路由器：在网络层，路由器根据路由表，寻找该ip的网段

2：路由器可以处理TCP/IP协议

3：路由器可以把一个IP分配给很多个主机使用，这些主机对外只表现出一个IP。

交换机可以把很多主机连起来，这些主机对外各有各的IP。

4：交换机是做端口扩展的，也就是让局域网可以连进来更多的电脑。

路由器是用来做网络连接，也就是；连接不同的网络

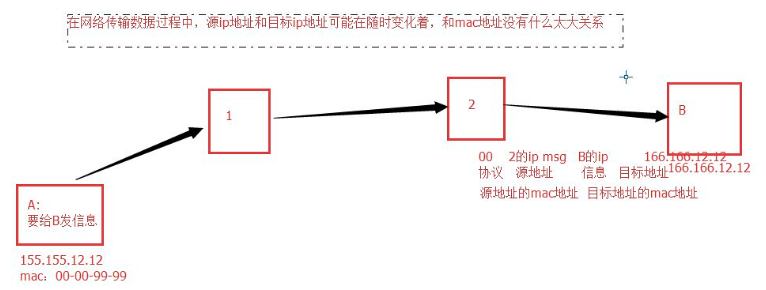
### 如何确定某个主机上某个程序

ip+端口  确定某个主机上的某个程序

端口：每个电脑上所运行的程序都有唯一（本机）的一个端口

&&---> 按位与

ip地址  && 子网掩码   ==== >  可以确定一个网关ip



目标： 155.155.12.12 msg 166.166.12.12

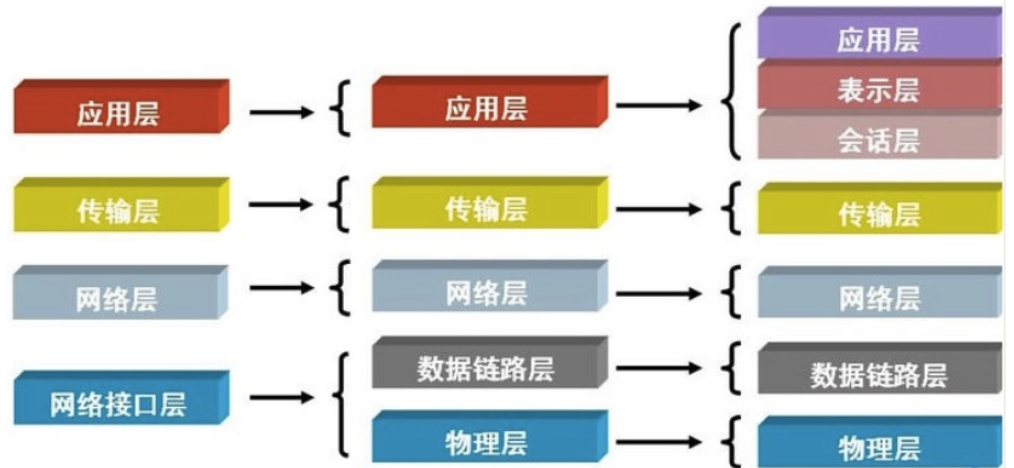
过程：（ip地址是随时变化的）

155.155.12.12 msg 1的ip

1的ip msg 2的ip

2的ip msg 166.166.12.12

### -OSI 7层协议



首先，用户感知到的只是最上面一层应用层，自上而下每层都依赖于下一层，所以我们从最下一层开始切入，比较好理解

每层都运行特定的协议，越往上越靠近用户，越往下越靠近硬件

物理层：主要定义物理设备标准，基于电器特性发送高低电压(电信号)，高电压对应数字1，低电压对应数字0

数据链路层：单纯的电信号0和1没有意义，需要定义了电信号的分组方式及每组的意思。

网络层：给位于不同地理位置的网络中的两个主机系统之间提供连接和路径选择，就是提供一些网络地址来区分不同的广播域/或子网

传输层：建立端口到端口的通信（区分每一个电脑的应用程序的每个端口）

应用层：用户使用的都是应用程序，应用程序大都不一样，所以应用层是规定应用程序的数据格式

### - [三次握手、四次挥手](http://www.cnblogs.com/jassin-du/p/7904497.html)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

　　三次握手：

　　　　SYC=1（建立连接） ACK（确认请求）

　　　　1、客户端（Client）向服务端（Server）发一次请求（SYN=1，随机产生一个值seq=J）

　　　　2、服务端确认并回复客户端（ACK=1, SYC=1，并在seq基础上产生一个随机数发给客户端）

　　　　3、客户端检验确认请求(ACK=1) 此时客户端与服务端就建立了连接

　　四次挥手：

　　　　FAN=1（断连接） ACK=1（确认请求）

　　　　1、客户端向服务端发一次请求（FAN=1）

　　　　2、服务端回复客户端 （ACK=1） （断开客户端—>服务端）

　　　　3、服务端再向客户端发请求（FAN=1） （因为有数据传输，所以2、3不能合并）

　　　　4、客户端确认请求（ACK=1） （断开服务端--->客户端）

[复制代码](javascript:void(0);)

### - TCP和UDP

[复制代码](javascript:void(0);)

TCP与UDP基本区别

1.基于连接与无连接

2.TCP要求系统资源较多，UDP较少；

3.UDP程序结构较简单

4.流模式（TCP）与数据报模式(UDP);

5.TCP保证数据正确性，UDP可能丢包

6.TCP保证数据顺序，UDP不保证

UDP应用场景：

1.面向数据报方式

2.网络数据大多为短消息

3.拥有大量Client

4.对数据安全性无特殊要求

5.网络负担非常重，但对响应速度要求高

具体编程时的区别

1.socket()的参数不同

　　 2.UDP Server不需要调用listen和accept

　　 3.UDP收发数据用sendto/recvfrom函数

　　 4.TCP：地址信息在connect/accept时确定

　　 5.UDP：在sendto/recvfrom函数中每次均 需指定地址信息

　　 6.UDP：shutdown函数无效

编程区别

通常我们在说到网络编程时默认是指TCP编程，即用前面提到的socket函数创建一个socket用于TCP通讯，函数参数我们通常填为SOCK\_STREAM。即socket(PF\_INET, SOCK\_STREAM, 0)，这表示建立一个socket用于流式网络通讯。

　 SOCK\_STREAM这种的特点是面向连接的，即每次收发数据之前必须通过connect建立连接，也是双向的，即任何一方都可以收发数据，协议本身提供了一些保障机制保证它是可靠的、有序的，即每个包按照发送的顺序到达接收方。

　　而SOCK\_DGRAM这种是User Datagram Protocol协议的网络通讯，它是无连接的，不可靠的，因为通讯双方发送数据后不知道对方是否已经收到数据，是否正常收到数据。任何一方建立一个socket以后就可以用sendto发送数据，也可以用recvfrom接收数据。根本不关心对方是否存在，是否发送了数据。它的特点是通讯速度比较快。大家都知道TCP是要经过三次握手的，而UDP没有。

基于上述不同，UDP和TCP编程步骤也有些不同，如下：

TCP:

TCP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt(); \* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、开启监听，用函数listen()；

　　5、接收客户端上来的连接，用函数accept()；

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

　　8、关闭监听；

TCP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置要连接的对方的IP地址和端口等属性；

　　5、连接服务器，用函数connect()；

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

UDP:

与之对应的UDP编程步骤要简单许多，分别如下：

　　UDP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、循环接收数据，用函数recvfrom();

　　5、关闭网络连接；

UDP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置对方的IP地址和端口等属性;

　　5、发送数据，用函数sendto();

　　6、关闭网络连接；

TCP和UDP是OSI模型中的运输层中的协议。TCP提供可靠的通信传输，而UDP则常被用于让广播和细节控制交给应用的通信传输。

UDP补充：

UDP不提供复杂的控制机制，利用IP提供面向无连接的通信服务。并且它是将应用程序发来的数据在收到的那一刻，立刻按照原样发送到网络上的一种机制。即使是出现网络拥堵的情况下，UDP也无法进行流量控制等避免网络拥塞的行为。此外，传输途中如果出现了丢包，UDO也不负责重发。甚至当出现包的到达顺序乱掉时也没有纠正的功能。如果需要这些细节控制，那么不得不交给由采用UDO的应用程序去处理。换句话说，UDP将部分控制转移到应用程序去处理，自己却只提供作为传输层协议的最基本功能。UDP有点类似于用户说什么听什么的机制，但是需要用户充分考虑好上层协议类型并制作相应的应用程序。

TCP补充：

TCP充分实现了数据传输时各种控制功能，可以进行丢包的重发控制，还可以对次序乱掉的分包进行顺序控制。而这些在UDP中都没有。此外，TCP作为一种面向有连接的协议，只有在确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费。TCP通过检验和、序列号、确认应答、重发控制、连接管理以及窗口控制等机制实现可靠性传输。

TCP与UDP区别总结：

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

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

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

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

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

5、TCP首部开销20字节;UDP的首部开销小，只有8个字节

6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

区别

[复制代码](javascript:void(0);)

TCP/UDP区别

TCP协议是面向连接，保证高可靠性（数据无丢失，数据无失序，数据无错误，数据无重复达到）传输层协议

UDP：数据丢失，无秩序的传输层协议（qq基于udp协议）

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 并发编程

### - 进程、线程、协程区别

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

### [- GIL锁](http://www.cnblogs.com/jassin-du/p/7986238.html)

[复制代码](javascript:void(0);)

GIL:全局解释器锁，是锁在cpython解释器上，导致同一时刻，同一进程只能有一个线程被执行

多进程：多进程模块multiprocessing来实现，cpu密集型，IO计算型可以用多进程

多线程：多线程模块threading来实现，IO密集型，多线程可以提高效率

协程：依赖于geenlet，对于多线程应用。cpu通过切片的方式来切换线程间的执行，遇到IO操作自动切换，线程切换时需要耗时，  
而协成好处没有切换的消耗，没有锁定概念。

进程：是资源管理单位，进行是相互独立的，实现并发和并发

线程：是最小的执行单位，线程的出现为了降低上下文切换的消耗，提供系统的并发性

[复制代码](javascript:void(0);)

### - 进程池和线程池

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 数据库

http://www.cnblogs.com/wupeiqi/articles/5748496.html

### 引擎

inondb、myisam

加锁语句关键点：for update

|  |  |
| --- | --- |
| 名称 | 特点 |
| inondb | 支持事务操作 |
|  | 加锁语句：  　　表锁：select \* from tb for update  　　行锁：select id,name from tb where id=2 for update |
|  |  |
| myisam | 查询速度快 |
|  | 全文索引 |
|  | 加锁语句  　　表锁：select \* from tb for update |

### 设计表

            - FK  
            - M2M

什么时候设计FK？

下拉框里面的数据就需要用FK关联另一张表

M2M？

多选的下拉框，或者checkbox  
            PS:   
                - 数据库部分：课程、老师、学生、分数  
                - 博客  
                - 权限表

### 查询

http://www.cnblogs.com/wupeiqi/articles/5729934.html

　　group by 分组对聚合的条件进行筛选需要通过havhing

SQL的left join 、right join、inner join之间的区别

left join (左连接) 返回包括左表中的所有记录和右表中联结字段相等的记录

right join(右连接) 返回包括右表中的所有记录1和左表中联结字段相等的记录

inner join（内连接）： 只返回两个表中联结字段相等的行

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 概念

### 触发器

[复制代码](javascript:void(0);)

对数据库某个表的增加，删除，改前后定义一些操作  
  
触发器是一种与表操作有关的数据库对象，当触发器所在表上发生指定事件时，将调用该对象，

即表的操作事件触发表上的触发器的执行。

可以这样来理解：当某个表上的数据发生变化的时候，就会执行一个函数，这个函数可能会去

执行一些对其他表的操作。这个函数就是触发器，它就相当于编程里的监听器一样，一旦监听到这个表

发生了某些变化，就会执行已经写好的一套逻辑。按照面向对象的思想，这个触发器是该表的一个属性

[复制代码](javascript:void(0);)

### 函数

触发函数是通过select

在SQL语句中使用函数

-select sleep（5）

- select date\_format(ctime,"%m") from tb;

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 关于mysql的函数

### 存储过程

[复制代码](javascript:void(0);)

含义:将SQL语句保存到数据库中，并命名；以后在代码中调用时，直接发送名称  
参数类型：  
　　in 只将参数传进去  
　　out 只拿结果  
　　inout 既可以传，可以取  
通过代码调用：  
import pymysql conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='123', db='t1')   
cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)   
# 执行存储过程   
cursor.callproc('p1', args=(1, 22, 3, 4))   
# 获取执行完存储的参数   
cursor.execute("select @\_p1\_0,@\_p1\_1,@\_p1\_2,@\_p1\_3")   
result = cursor.fetchall()   
conn.commit()   
cursor.close()   
conn.close()   
print(result)  
  
  
用途：保险、金融、

存储过程包含了一系列可执行的sql语句，存储过程存放于数据库(MySQL)中，通过调用它的名字可以执行其内部的一堆sql。

使用存储过程的优点：

　　（1）用于替代程序写的SQL语句，实现程序与sql解耦；

　　（2）基于网络传输，传别名的数据量小，而直接传sql数据量大。缺点是程序员扩展功能不方便。

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 问题

存储过程和函数的区别

|  |  |
| --- | --- |
| 存储过程 | 函数 |
| 返回值：return | 返回值：out/inout |
| 参数：in/out/in out | 参数：in/out/in out |
|  |  |

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 视图

对某些表进行sql查询，将结果显示实时显示出来（只能查）

视图是一个虚拟表（非真实存在），其本质是【根据SQL语句获取动态的数据集，并为其命名】，用户使用时只需使用【名称】即可获取结果集，并可以将其当作表来使用

注意

使用视图时，将其当作表进行操作即可，由于视图是虚拟表，所以无法使用其对真实表进行创建、更新和删除操作，仅能做查询用。

v = select \* from tb where id < 1000

这里的v相当于视图

select \* from v

select \* from (select \* from tb where id < 1000) as v

**以上内容都是保存在数据库中。**

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 索引

[复制代码](javascript:void(0);)

特殊的存储数据结构：B+树/哈希索引（类似字典的key，当数据都不重复可以做哈希索引） ==> 查找速度快；

缺点：建索引需要额外的文件存在索引，所以更新速度慢（修改）

b+树性质

　　1、索引字段要尽量的小

　　2、索引的最左匹配原则

单列

功能

普通索引：加速查找

唯一索引：加速查找 + 约束：不能重复（只能有一个空，不然就重复了）

主键（primay key）：加速查找 + 约束：不能重复 + 不能为空

多列

　　联合索引（多个列创建索引）-----> 相当于单列的普通索引

　　联合唯一索引 -----> 相当于单列的唯一索引

　　ps：联合索引的特点：遵循最左前缀的规则

其他词语：

·· - 索引合并，利用多个单例索引查询；（例如在数据库查用户名和密码，分别给用户名和密码建立索引）

- 覆盖索引，在索引表中就能将想要的数据查询到；

[复制代码](javascript:void(0);)

### 问题：创建了索引，应该如何命中索引或什么情况无法命中索引\*\*

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 无法命中索引

### 如何开启慢日志查询？

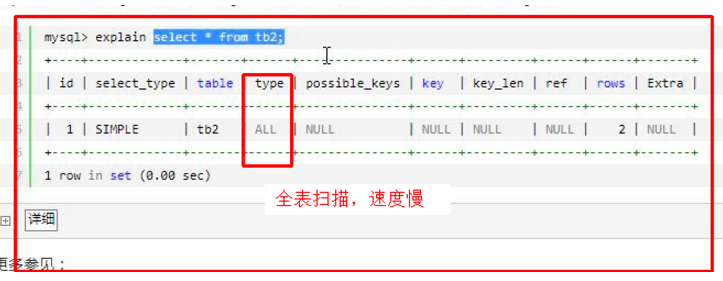
https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 配置文件

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 相关

### 执行计划

查看有没有命中索引，让数据库帮看看运行速度快不快

explain select \* from table;



当type为all时，是为全表索引

### 导入、导出数据库

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 命令

### 数据库分页

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 分页问题

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 优化方案

[复制代码](javascript:void(0);)

1、创建数据表时把固定长度的放在前面（）

2、将固定数据放入内存： 例如：choice字段 （django中有用到，数字1、2、3…… 对应相应内容）

3、char 和 varchar 的区别(char可变, varchar不可变 )

4、联合索引遵循最左前缀(从最左侧开始检索)

5、避免使用 select \*

6、读写分离

　　　　- 实现：两台服务器同步数据

　　　　- 利用数据库的主从分离：主，用于删除、修改、更新；从，用于查；  
读写分离:利用数据库的主从进行分离：主，用于删除、修改更新；从，用于查

7、分库  
　　　　- 当数据库中的表太多，将某些表分到不同的数据库，例如：1W张表时  
　　　　- 代价：连表查询

8、分表  
　　　　- 水平分表：将某些列拆分到另外一张表，例如：博客+博客详情  
　　　　- 垂直分表：讲些历史信息分到另外一张表中，例如：支付宝账单  
  
9、加缓存  
　　　　- 利用redis、memcache （常用数据放到缓存里，提高取数据速度）

如果只想获取一条数据

- select \* from tb where name=‘alex’ limit 1

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 爬虫

并发的方案

　　多进程---> 多线程 ----> 利用“异步非阻塞”模块实现单线程并发请求

[复制代码](javascript:void(0);)

本质：

sk = socket()

# 阻塞

sk.connect(('www.cnblogs.com',80))

sk.sendall(b"GET /wupeiqi http1.1\r\n.....\r\n\r\n")

sk.sendall(b"POST /wupeiqi http1.1\r\n.....\r\n\r\nuser=alex&pwd=123")

# 阻塞

data = sk.recv(8096)

sk.close()

[复制代码](javascript:void(0);)

什么是异步非阻塞

　　非阻塞：不等待

代码：

sk = socket.socket()

sk.setblocking(False)

　　异步：回调，当达到某个指定的状态之后，自动调用特定函数

　　实例：

　　nb\_async.py   实现异步非阻塞的模块

IO多路复用

　　监听多个socket是否发生变化

如何自定义异步非阻塞模块？

基于socket设置setblocking和 IO多路复用来实现。  
爬虫发送Http请求本质创建socket对象；  
IO多路复用"循环"监听socket是否发生变化，一旦发生变化，我们可以自定义操作（触发某个函数的执行）

### Http协议本质

[复制代码](javascript:void(0);)

　　浏览器本质,socket客户端遵循Http协议

　　HTTP协议本质：通过\r\n分割的规范+ 请求响应之后断开链接 == > 无状态、 短连接

具体：

　　Http协议是建立在tcp之上的，是一种规范，它规范定了发送的数据的数据格式，  
然而这个数据格式是通过\r\n 进行分割的，请求头与请求体也是通过2个\r\n分割的，响应的时候，  
响应头与响应体也是通过\r\n分割，并且还规定已请求已响应就会断开链接

即---> 短连接、无状态

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## Django rest framework 框架

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## restful

### ****1：用django写接口时，有没有用什么框架？****

- 使用rest framework框架

- 原生CBV

### ****2：rest framework框架优点？****

rest framework帮助开发者提供了很多组件，可以提高开发效率。

### 3：有哪些组件

[复制代码](javascript:void(0);)

- **路由**  
　　　　自动帮助开发者快速为一个视图创建4个url

　　　　www.oldboyedu.com/api/v1/student/www.oldboyedu.com/api/v1/student(?P<format>\w+)

　　　　www.oldboyedu.com/api/v1/student/(?P<pk>\d+)/www.oldboyedu.com/api/v1/student/(?P<pk>\d+)(?P<format>\w+)

- **版本处理**

- 问题：版本都可以放在那里？

　　- url

　　- GET

　　- 请求头

- **认证**

　　- 问题：认证流程？

- **权限**

　　- 权限是否可以放在中间件中？以及为什么？

- **节流**

　　- 匿名用户可以真正的防止？无法做到真正的访问频率控制，只能把小白拒之门外。

　　如果要封IP，使用防火墙来做。

　　- 登录用户可以通过用户名作为唯一标示进行控制，如果有人注册很多账号，也无法防止。

- **视图**

- **解析器** ，根据Content-Type请求头对请求体中的数据格式进行处理。request.data

- **分页**

- **序列化**

　　- 序列化

　　　　- source

　　　　- 定义方法

　　- 请求数据格式校验

- **渲染器**

[复制代码](javascript:void(0);)

### 4：谈谈对restful规范的理解

[复制代码](javascript:void(0);)

1：restful其实是一套编写接口的规范、协议。规定了如何编写以及如何设置返回值，状态码等信息。

2：最显著的特点

restful：（用restful）

给用户一个url，根据method不同在后端做不同的处理，比如：post创建数据、get获取数据  
、put和patch修改数据、delete删除数据。

no rest：（不用restful）

给调用者很对url，每个url代表一个功能，比如;add\_usser/delete\_user/edit\_user

3: 当然，协议里面还有其他的，比如：、

- 版本 ，来控制让程序有多个版本共存的情况，版本可以放在url、请求头（accept/自定义）、GET参数

- 状态码 ，200（成功，服务器提供了请求的网页）/300（ 重定向 - 要完成请求必须进行更进一步的处理）  
　　　　　　　　/400（客户端错误）/500（服务端错误）

- url中尽量使用名词，restful也可以称为“面向资源编程”

- api标示：

api.baidu.com

www.baidu.com/api/

[复制代码](javascript:void(0);)

### 5：restful怎么学的

- 因为之前公司要写这样项目

- 接口

- 公司要做前后端分离的项目

- 公司要做微信小程序的开发

- 所以就开始学习restful规范，看的技术文章 阮一峰的博客学到的规范。

### 6：状态码

[复制代码](javascript:void(0);)

1XX 提示信息 - 表示请求已被成功接收，继续处理

2XX 成功 - 表示请求已被成功接收，理解，接受

3XX 重定向 - 要完成请求必须进行更进一步的处理

4XX 客户端错误 - 请求有语法错误或请求无法实现

5XX 服务器端错误 - 服务器未能实现合法的请求

[复制代码](javascript:void(0);)

### 7:method

post 创建数据、get获取数据、put和patch修改数据、delete删除数据

### 8:常见请求头

- user-agent  
- host  
- referer  
- cookie   
- content-type

### 9:你是用什么开发的restful接口？

　　使用django rest framework框架

### 10：为什么使用django rest framework框架

[复制代码](javascript:void(0);)

在编写接口时，可以不使用django rest framework框架

如果不使用：也是可以做，那么就可以django的CBV来实现，开发者编写的代码会更多一些。

使用：内部帮助我们提供了很多方便的组件，我们通过配置就可以完成相应操作，如：

序列化，可以做用户请求数据校验+queryset对象的序列化称为json

解析器，获取用户请求数据request.data,会自动根据content-type请求头的能对数据进行解析

分页：将从数据库获取到的数据在页面进行分页显示。

还有其他：

认证

权限

访问频率控制

[复制代码](javascript:void(0);)

### 11：rest framwork视图你都用过哪些基类？（继承哪些类）

[复制代码](javascript:void(0);)

a. 继承APIView（最原始）但定制性比较强

这个类属于rest framework中的顶层类，内部帮助我们实现了只是基本功能：认证、权限、频率控制，  
但凡是数据库、分页等操作都需要手动去完成，比较原始。

class GenericAPIView(APIView)

def post(...):

pass

b.继承GenericViewSet（ViewSetMixin，generics.GenericAPIView）  
　　首先他的路由就发生变化

如果继承它之后，路由中的as\_view需要填写对应关系  
　　在内部也帮助我们提供了一些方便的方法：  
　　get\_queryset  
　　get\_object  
　　get\_serializer  
　　get\_serializer\_class  
　　get\_serializer\_context  
　　filter\_queryset  
注意：要设置queryset字段，否则会抛出断言的异常。  
  
代码  
只提供增加功能 只继承GenericViewSet  
  
class TestView(GenericViewSet):  
　　serialazer\_class = xxx  
　　def creat(self,\*args,\*\*kwargs):

　　　　pass # 获取数据并对数据  
  
c. 继承 modelviewset --> 快速快发  
　　　　-ModelViewSet(增删改查全有+数据库操作)  
　　　　-mixins.CreateModelMixin（只有增）,GenericViewSet  
　　　　-mixins.CreateModelMixin,DestroyModelMixin,GenericViewSet  
　　对数据库和分页等操作不用我们在编写，只需要继承相关类即可。

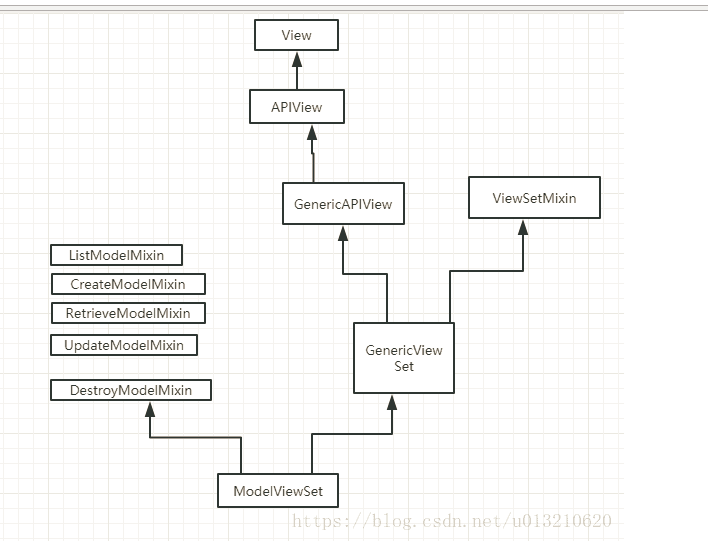
示例：只提供增加功能  
class TestView(mixins.CreateModelMixin,GenericViewSet):  
　　　　serializer\_class = XXXXXXX

\*\*\*

　　modelviewset --> 快速开发，复杂点的genericview、apiview

[复制代码](javascript:void(0);)

 类的继承关系，总共继承11个类



### 12、认证流程

[复制代码](javascript:void(0);)

- 如何编写？写类并实现authenticators

　　请求进来认证需要编写一个类，类里面有一个authenticators方法，我们可以自定义这个方法，可以定制3类返回值。  
　　成功返回元组，返回none为匿名用户，抛出异常为认证失败。  
  
源码流程：请求进来先走dispatch方法，然后封装的request对象会执行user方法，由user触发authenticators认证流程

- 方法中可以定义三种返回值：

- （user,auth），认证成功

- None , 匿名用户

- 异常 ，认证失败

- 流程：

- dispatch

- 再去request中进行认证处理

[复制代码](javascript:void(0);)

### 13、访问频率控制（redis应用场景）

[复制代码](javascript:void(0);)

- 匿名用户，根据用户IP或代理IP作为标识进行记录，为每一个用户在redis中创建一个列表

{

throttle\_1.1.1.1:[1526868876.497521,152686885.497521...]

throttle\_1.1.1.2:[1526868876.497521,152686885.497521...]

throttle\_1.1.1.3:[1526868876.497521,152686885.497521...]

throttle\_1.1.1.4:[1526868876.497521,152686885.497521...]

throttle\_1.1.1.5:[1526868876.497521,152686885.497521...]

}

每个用户再来访问时，需要先去记录中剔除以及过期时间，再根据列表的长度判断是否可以继续访问。

如何封IP：在防火墙中进行设置

- 注册用户，根据手机号或邮箱进行判断。

{

throttle\_xxxx1:[1526868876.497521,152686885.497521...]

throttle\_xxxx2:[1526868876.497521,152686885.497521...]

throttle\_xxxx3:[1526868876.497521,152686885.497521...]

throttle\_xxxx4:[1526868876.497521,152686885.497521...]

}

每个用户再来访问时，需要先去记录中剔除以及过期时间，再根据列表的长度判断是否可以继续访问。

1分钟：40次

[复制代码](javascript:void(0);)

### 14：接口的幂等性？（是否会造成2次伤害）

[复制代码](javascript:void(0);)

一个接口通过1次相同的访问，再对该接口进行N次相同的访问时候，对资源不造影响，那么就认为接口具有幂等性。

比如：

GET， 第一次获取结果、第二次也是获取结果对资源都不会造成影响，幂等。

POST，第一次新增数据，第二次也会再次新增，非幂等。

PUT， 第一次更新数据，第二次不会再次更新，幂等。

PATCH，第一次更新数据，第二次不会再次更新，非幂等。

DELTE，第一次删除数据，第二次不在再删除，幂等。

[复制代码](javascript:void(0);)

### restful规范

GET、POST、PUT、UPDATE/PATCH、DELETE

客户端通过以上4种http动词，对服务器资源进行操作，实现“表现层状态转化”；  
**【注意】**  
只是一种架构而不是标准，提供设计原则和约束条件。  
适用于客户端与服务器交互类的软件，这是因为它更加简洁、有层次、更利于实现缓存等机制；

### API设计

* API与用户的通信协议，总是使用[HTTPs协议](http://www.ruanyifeng.com/blog/2014/02/ssl_tls.html)。
* 域名
  + https://api.example.com                         尽量将API部署在专用域名（会存在跨域问题）
  + https://example.org/api/                        API很简单
* 版本
  + URL，如：https://api.example.com/v1/
  + 请求头                                                  跨域时，引发发送多次请求
* 路径，视网络上任何东西都是资源，均使用名词表示（可复数）
  + https://api.example.com/v1/zoos
  + https://api.example.com/v1/animals
  + https://api.example.com/v1/employees
* method
  + GET      ：从服务器取出资源（一项或多项）
  + POST    ：在服务器新建一个资源
  + PUT      ：在服务器更新资源（客户端提供改变后的完整资源）
  + PATCH  ：在服务器更新资源（客户端提供改变的属性）
  + DELETE ：从服务器删除资源
* 过滤，通过在url上传参的形式传递搜索条件
  + https://api.example.com/v1/zoos?limit=10：指定返回记录的数量
  + https://api.example.com/v1/zoos?offset=10：指定返回记录的开始位置
  + https://api.example.com/v1/zoos?page=2&per\_page=100：指定第几页，以及每页的记录数
  + https://api.example.com/v1/zoos?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序
  + https://api.example.com/v1/zoos?animal\_type\_id=1：指定筛选条件
* 错误处理，状态码是4xx时，应返回错误信息，error当做key。
* {
* error: "Invalid API key"

}

* 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。
* [复制代码](javascript:void(0);)

复制代码

GET /collection：返回资源对象的列表（数组）

GET /collection/resource：返回单个资源对象

POST /collection：返回新生成的资源对象

PUT /collection/resource：返回完整的资源对象

PATCH /collection/resource：返回完整的资源对象

DELETE /collection/resource：返回一个空文档

复制代码

[复制代码](javascript:void(0);)

* 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。

[复制代码](javascript:void(0);)

复制代码

{"link": {

"rel": "collection https://www.example.com/zoos",

"href": "https://api.example.com/zoos",

"title": "List of zoos",

"type": "application/vnd.yourformat+json"

}}

复制代码

[复制代码](javascript:void(0);)

12. Https和Http区别？

[复制代码](javascript:void(0);)

端口：

http：80

https: 443

流程：

- 自定义证书

- 认证机构

[复制代码](javascript:void(0);)

### 源码后的心得

1：封装的思想：

　　例如：将认证组件封装request里面，接着就只要requesst。user就能

2：开放，封闭思想（只要改配置即生效）

3：配置：中间件配置

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 问题

### 请求头

### 请求方法（8种）

[复制代码](javascript:void(0);)

请求方法：

get： 获得URI指定的资源，主要目的是获取

post：将主体传输到URI，主要目的是传输，而不是获取

put：传输文件，在请求主体中包含文件内容，然后保存到URI指定的位置。

head：只获取GET结果的头部，除去主体。

delete：删除文件，是PUT的逆操作

options：查询针对请求URI指定资源支持的方法。用法如：OPTIONS \* HTTP/1.1

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

trace：回显服务器收到的请求，主要用于测试或诊断

注意：由于PUT和DELETE方法涉及服务器文件的直接操作，较为敏感，所以一般Web网站并不支持

除非配合使用了Web的验证机制。

[复制代码](javascript:void(0);)

### ****渲染器有坑****

[复制代码](javascript:void(0);)

渲染器有坑

- 指定渲染器只用JSON

- 视图：

　　class UserView(...):

　　queryset = []

　　...

assert 是的作用？断言

条件成立则继续往下，否则跑出异常，一般用于：满足某个条件之后，才能执行，否则应该跑出异常。

[复制代码](javascript:void(0);)

### 场景

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 源码

### 

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 关联问题

### TCP和UDP区别？

TCP协议是面向连接，保证高可靠性（数据无丢失，数据无失序，数据无错误，数据无重复达到）传输层协议

UDP：数据丢失，无秩序的传输层协议（qq基于udp协议）

### 三次握手、四次挥手

### OSI 七层协议

### scrapy框架

[复制代码](javascript:void(0);)

基本命令：

scrapy startproject sp1

cd sp1

scrapy genspider chouti chouti.com

scrapy crawl chouti --nolog

编写：

a. 起始URL

b. parse函数

c. pipelines

d. 解析器

[复制代码](javascript:void(0);)

### 协程

[复制代码](javascript:void(0);)

协程本身不是并发

协程是"微线程"（在实际中是不存在的），是由程序员人为创造出来并控制程序：  
　　先执行某段代码、再跳到某处执行某段代码

如果遇到非IO请求来回切换：性能更低。

如果遇到IO（耗时）请求来回切换：性能高、实现并发（本质上利用IO等待的过程，再去干一些其他的事）

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 前端

HTML、CSS、JS

### 框架和类库（模块）

- jQuery

- BootStrap

- Vue.js(与vue齐名的前端框架React和Angular)

### 响应式布局

[复制代码](javascript:void(0);)

响应式布局是通过@media实现的

@media （min-width：768px）{

.pg-header{

background-color:green;

}

}

@media (min-width:992px){

.pg-header{

background-color:pink;

}

}

[复制代码](javascript:void(0);)

### jQuery选择器

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

### Ajax

jQuery Ajax 和 原生ajax

jQuery ajax ：$.ajax(..)

原生：ajax：XMLHttpRequest

### 跨域（jsonp、cors）

线上不会遇到跨域

#### JSONP

JSONP

jsonp是json用来跨域的一个东西。原理是通过script标签的跨域特性来绕过同源策略。

JSONP的简单实现模式，或者说是JSONP的原型：创建一个回调函数，然后在远程服务上调用这个函数并且将JSON 数据形式作为参数传递，  
完成回调。

#### CORS

[复制代码](javascript:void(0);)

CORS

浏览器将CORS请求分成两类：简单请求（simple request）和复杂请求（not-so-simple request）。

简单请求

同时满足以下两大条件

（1) 请求方法是以下三种方法之一：

HEAD

GET

POST

（2）HTTP的头信息不超出以下几种字段：

Accept

Accept-Language

Content-Language

Last-Event-ID

Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain

凡是不同时满足上面两个条件，就属于非简单请求。

[复制代码](javascript:void(0);)

#### 简单请求与复杂请求的区别

[复制代码](javascript:void(0);)

\* 简单请求和非简单请求的区别？

简单请求：一次请求

非简单请求：两次请求，在发送数据之前会先发一次请求用于做“预检”，只有“预检”通过后才再发送一次请求用于数据传输。

\* 关于“预检”

- 请求方式：OPTIONS

- “预检”其实做检查，检查如果通过则允许传输数据，检查不通过则不再发送真正想要发送的消息

- 如何“预检”

=> 如果复杂请求是PUT等请求，则服务端需要设置允许某请求，否则“预检”不通过

Access-Control-Request-Method

=> 如果复杂请求设置了请求头，则服务端需要设置允许某请求头，否则“预检”不通过

Access-Control-Request-Headers

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

支持跨域，简单请求

服务器设置响应头：Access-Control-Allow-Origin = '域名' 或 '\*'

支持跨域，复杂请求

由于复杂请求时，首先会发送“预检”请求，如果“预检”成功，则发送真实数据。

“预检”请求时，允许请求方式则需服务器设置响应头：Access-Control-Request-Method

“预检”请求时，允许请求头则需服务器设置响应头：Access-Control-Request-Headers

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## web框架

### 你了解哪些Web框架和区别？（django、flask）

[复制代码](javascript:void(0);)

django和flask区别

对于django来说吧，内部组件就特别多。（比如from组件、modelfrom、admin、orm），功能也都挺完善的，（有那种大而成的感觉）

对于flask，内部组件就比较少了，但是有很多第三方组件组件来扩展它，（比如说wtform、Flask-SQLAlchemy，  
这些组件其实也能堆建出来框架，）所以呢，他的可扩展行就比较强了，(所以对他本身来说有那种短小精悍的感觉)

这就是他们大体上的区别。

对于django它内部已经实现这些组件的功能，而flask不过只要通过第三方下载安装这些组件来实现这些功能

对于他们2个共同点：他们2个框架都没写socket，所以他们都是第三方模块wsgi，  
django本身运行起来使用wsgiref 而flask是用Werkzeug WSGI

还有一个区别就是他们的请求管理不太一样:djagno是通过参数传递的，而flask是通过上下文管理机制

最重要的的是flask他有那个上下文管理机制，挺牛逼的

Django的Form主要具有以下功能？

生成HTMl标签，验证用户数据 is\_vaild，HTML Form提交保留上次提交数据，初始化页面显示内容

[复制代码](javascript:void(0);)

### django的form组件作用？

[复制代码](javascript:void(0);)

- 对用户请求的数据进行校验

- 生成HTML标签

PS：

- form对象是一个可迭代对象。

- 问题：choice的数据如果从数据库获取可能会造成数据无法实时更新

- 重写构造方法，在构造方法中重新去数据库获取值。

- ModelChoiceField字段

from django.forms import Form

from django.forms import fields

from django.forms.models import ModelChoiceField

class UserForm(Form):

name = fields.CharField(label='用户名',max\_length=32)

email = fields.EmailField(label='邮箱')

ut\_id = ModelChoiceField(queryset=models.UserType.objects.all())

依赖：

class UserType(models.Model):

title = models.CharField(max\_length=32)

def \_\_str\_\_(self):

return self.title

[复制代码](javascript:void(0);)

### ****django的Form组件存在的问题与modelform区别？应用场景？****

form对象是一个可迭代对象，字段需要自己手写。

　　- 问题：choice的数据如果从数据库获取可能会造成数据无法实时更新

　　- 重写构造方法，在构造方法中重新去数据库获取值。

　　- ModelChoiceField字段

### 区别

[复制代码](javascript:void(0);)

- 区别：

- Form，字段需要自己手写。

class Form(Form):

xx = fields.CharField(.)

xx = fields.CharField(.)

xx = fields.CharField(.)

xx = fields.CharField(.)

- ModelForm，可以通过Meta进行定义

class MForm(ModelForm):

class Meta:

fields = "\_\_all\_\_"

model = UserInfo

- 应用：只要是客户端向服务端发送表单数据时，都可以进行使用，如：用户登录注册

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

**ContentType的作用？以及应用场景？**

[复制代码](javascript:void(0);)

contenttype是django的一个组件（app），为我们找到django程序中所有app中的所有表并添加到记录中。

可以使用他再加上表中的两个字段实现：一张表和N张表创建FK关系。

- 字段：表名称

- 字段：数据行ID

应用：路飞表结构优惠券和专题课和学位课关联。

[复制代码](javascript:void(0);)

### Django信号作用

[复制代码](javascript:void(0);)

django的信号其实就是django内部为开发者预留的一些自定制功能的钩子。

只要在某个信号中注册了函数，那么django内部执行的过程中就会自动触发注册在信号中的函数。

如：

pre\_init # django的modal执行其构造方法前，自动触发

post\_init # django的modal执行其构造方法后，自动触发

pre\_save # django的modal对象保存前，自动触发

post\_save # django的modal对象保存后，自动触发

[复制代码](javascript:void(0);)

### ****用信号做过什么？****

在数据库某些表中添加数据时，可以进行日志记录。

**csrf原理**

目标：防止用户直接向服务端发起POST请求。

方案：先发送GET请求时，将token保存到：cookie、Form表单中（隐藏的input标签），以后再发送请求时只要携带过来即可。

问题：如果想后台发送POST请求？

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 三种

### Django[请求生命周期](http://www.cnblogs.com/jassin-du/p/8379173.html)

用户请求进来 先走到 wsgi 然后请求就交给 jango的中间件 （穿过django中间件的方法是 process\_request）   
接着是 路由匹配 路由匹配成功之后 就执行相应的 视图函数

在视图函数中可以调用orm做数据库操作 再从模板路径 将模板拿到 然后在后台经过渲染

渲染完成之后变成一个字符串 再把这个字符串经过所有中间件 和 wsgi 返回给用户

### 浏览器上输入地址，回车然后发生了什么？

关联到=> Http请求生命周期

- 域名解析--->浏览器会先到本地host文件（本地dns缓存）中找，找不到会取DNS服务器，DNS服务器请求解析该URL中的域名所对应的IP地址

- 连接成功--->，DNS服务器会将域名解析出IP地址后，根据IP地址和默认端口80，和服务器建立TCP连接

- 浏览器发送数据---->浏览器发出读取文件的http请求，该请求报文作为TCP，三次握手的第三个报文的数据发送给服务器

- 服务端接收数据并处理再响应---->服务器对浏览器请求做出响应，并把对应的html文件发送给浏览器

- 服务端做的事比较复杂---->释放TCP连接浏览器将该HMTL渲染并显示内容

，如果是：Django、Flask

浏览器会先到本地host文件（本地dns缓存）中找，找不到会取DNS服务器，DNS服务器请求解析该URL中的域名所对应的IP地址，  
根据ip找到对应的web服务器，这样你和服务器连接就建立了，然后就等着收发消息

### 什么是wsgi

是web服务网关接口，是一套协议。以下模块实现了wsgi协议：

- wsgiref

- werkzurg

- uwsgi 关于部署

以上模块本质：编写socket服务端，用于监听请求，当有请求到来，则将请求数据进行封装，然后交给web框架处理。

[复制代码](javascript:void(0);)

from wsgiref.simple\_server import make\_server

def run\_server(environ, start\_response):

"""

environ: 封装了请求相关的数据

start\_response：用于设置响应头相关数据

"""

start\_response('200 OK', [('Content-Type', 'text/html')])

return [bytes('<h1>Hello, web!</h1>', encoding='utf-8'), ]

if \_\_name\_\_ == '\_\_main\_\_':

httpd = make\_server('', 8000, run\_server)

httpd.serve\_forever()

[复制代码](javascript:void(0);)

Django源码：wsgi

[复制代码](javascript:void(0);)

class WSGIHandler(base.BaseHandler):

request\_class = WSGIRequest

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(WSGIHandler, self).\_\_init\_\_(\*args, \*\*kwargs)

self.load\_middleware()

def \_\_call\_\_(self, environ, start\_response):

# 请求刚进来之后 #

set\_script\_prefix(get\_script\_name(environ))

signals.request\_started.send(sender=self.\_\_class\_\_, environ=environ)

request = self.request\_class(environ)

response = self.get\_response(request)

response.\_handler\_class = self.\_\_class\_\_

status = '%d %s' % (response.status\_code, response.reason\_phrase)

response\_headers = [(str(k), str(v)) for k, v in response.items()]

for c in response.cookies.values():

response\_headers.append((str('Set-Cookie'), str(c.output(header=''))))

start\_response(force\_str(status), response\_headers)

if getattr(response, 'file\_to\_stream', None) is not None and environ.get('wsgi.file\_wrapper'):

response = environ['wsgi.file\_wrapper'](response.file\_to\_stream)

return response

[复制代码](javascript:void(0);)

### 中间件

#### 作用

对所有的请求进行批量处理，在视图函数执行前后进行自定义操作。

#### 应用用户登录校验/， 为什么：如果不使用就需要为每个函数添加装饰器，太繁琐

[复制代码](javascript:void(0);)

权限处理/ why? 用户登录后，将权限放到session中，然后再每次请求进来在中间件里，根据当前的url去session中匹配，  
 判断当前用户是否有权限访问当前url,有权限就继续访问，没有就返回，  
 （检查的东西就可以放到中间件中进行统一处理）在process\_request方法里面做的，  
　　　　　　　　我们的中间件是放在session后面，因为中间件需要到session里面取数据

还有一些内置：  
　　CSRF：对所有的post请求做验证/ 将jango生成的一串字符串发送给我们，1种是从请求体发过来，一种是放在隐藏的标签里面  
　　　　　　　用的是process\_view　  
　　session/  
　　缓存/ 常用的数据放在缓存里面，就不用走视图函数，请求进来通过所有的process\_request,会到缓存里面查数据，有就直接拿，  
　　　　　　　　没有就走视图函数  
　　　　　　关键点：1：执行完所有的process\_request才去缓存取数据  
　　　　　　　　　　2：执行完所有的process\_response才将数据放到缓存  
　　日志记录

另外，还有一个就是处理：cors跨域（场景：前后端分离时，本地测试开发时使用）  
　　　　　　　　如果网站之间存在跨域，域名不同，端口不同会导致出现跨域，但凡出现跨域，浏览器就会出现同源策略的限制  
　　　　　　　　解决：在我们的服务端给我们响应数据，加上响应头---> 在中间件加的

[复制代码](javascript:void(0);)

#### 关于缓存问题

1:为什么放在最后一个process\_request才去缓存

因为需要验证完用户的请求，才能返回数据

2:什么时候将数据放到缓存中

第一次走中间件，缓存没有数据，会走视图函数，取数据库里面取数据，

当走完process\_response,才将数据放到缓存里，因为，走process\_response的时候可能给我们的响应加处理

### ****为什么要用缓存？****

将常用且不太频繁修改的数据放入缓存。

以后用户再来访问，先去缓存查看是否存在，如果有就返回

否则，去数据库中获取并返回给用户（再加入到缓存，以便下次访问）

### ****django内部支持哪些缓存？****

[复制代码](javascript:void(0);)

Django中提供了6种缓存方式：

　　开发调试（不加缓存）

　　内存

　　文件

　　数据库

　　Memcache缓存（python-memcached模块）

　　Memcache缓存（pylibmc模块）

安装第三方组件支持redis：

　　django-redis组件

[复制代码](javascript:void(0);)

### ****设置缓存****

- 全站缓存

- 视图函数缓存

- 局部模板缓存

#### 中间件中方法（五个）

process\_request(self,request) 先走request 通过路由匹配返回

process\_view(self, request, callback, callback\_args, callback\_kwargs) 再返回执行view

process\_template\_response(self,request,response) 当视图函数的返回值

process\_exception(self, request, exception) 当视图函数的返回值对象中有render方法时，该方法才会被调用

process\_response(self, request, response)

### 执行流程https://images2018.cnblogs.com/blog/1258691/201805/1258691-20180516093120159-1804198263.png

第一次缓存里面没有数据，

### 注意：

　　对于所有请求的批量做处理的时候用中间件  
　　单独对某几个函数做处理的时候用装饰器

### csrf本质

目标：防止用户直接向服务端发起POST请求  
如何防止：  
 - 用户先发送GET获取csrf token: Form表单中一个隐藏的标签 + cookie

- 发起POST请求时，需要携带之前发送给用户的csrf token；

- 在中间件的process\_view方法中进行校验。

### 问题：如何想后台发送POST请求

[复制代码](javascript:void(0);)

爬虫：  
     reqeusts.post()

ajax

$.ajax({  
    url:'/index',  
    type:'POST',  
    data:{csrfmiddlewaretoken:'{{ csrf\_token }}',name:'alex'}  
})

from表单  
<form method="POST">  
    {% csrf\_token %}  
    <input type='text' name='user' />  
    <input type='submit' />  
</form>

[复制代码](javascript:void(0);)

### [关联ajax问题](http://www.cnblogs.com/jassin-du/p/9046116.html" \t "_blank)

### 路由

url+关系

反向解析---> 写个名称 reverse

路由分发，本质返回一个元组，里面三个元素，第一是列表

### [视图](http://www.cnblogs.com/jassin-du/p/8313970.html" \l "_label2_0" \t "_blank)

基于函数  fbv

[复制代码](javascript:void(0);)

# FBV 写法

# urls.py

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

# views.py

def login(request):

if request.method == "POST":

print(request.POST)

return render(request,"login.html")

# HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>登录页面</title>

</head>

<body>

<form action="{% url 'login' %}" method="post" enctype="multipart/form-data">

<input type="text" name="user2">

<input type="file" name="file">

<input type="submit" value="提交">

</form>

</body>

</html>

[复制代码](javascript:void(0);)

CBV 基于类

[复制代码](javascript:void(0);)

# urls.py

url(r'^login/$',views.Login.as\_view(), name="login"),

# views.py

from django.views import View

class Login(View): # 类首字母大写

def get(self,request):

return render(request,"login.html")

def post(self,request):

print(request.POST)

return HttpResponse("OK")  
  
加装饰器  
  
=================================  
class IndexView(View):  
      
    # 如果是crsf相关，必须放在此处  
    def dispach(self,request):  
        # 通过反射执行post/get   
      
    @method\_decoretor(装饰器函数)  
    def get(self,request):  
        pass  
          
    def post(self,request):  
        pass   
路由：IndexView.as\_view()

[复制代码](javascript:void(0);)

### FBV和CBV的区别

- 没什么区别，因为他们的本质都是函数。CBV的.as\_view()返回的view函数，view函数中调用类的dispatch方法，  
在dispatch方法中通过反射执行get/post/delete/put等方法。

非要说区别的话：

- CBV比较简洁，GET/POST等业务功能分别放在不同get/post函数中。FBV自己做判断进行区分。

### CBV的注意事项

[复制代码](javascript:void(0);)

添加装饰器

方式一：

from django.views import View

from django.utils.decorators import method\_decorator ---> 需要引入memethod\_decorator

def auth(func):

def inner(\*args,\*\*kwargs):

return func(\*args,\*\*kwargs)

return inner

class UserView(View):

@method\_decorator(auth)

def get(self,request,\*args,\*\*kwargs):

return HttpResponse('...')

方式二：

- csrf的装饰器要加到dispath前面

from django.views import View

from django.utils.decorators import method\_decorator

from django.views.decorators.csrf import csrf\_exempt,csrf\_protect ---> 需要引入 csrf\_exempt

class UserView(View):

@method\_decorator(csrf\_exempt)

def dispatch(self, request, \*args, \*\*kwargs):

return HttpResponse('...')

或者：

from django.views import View

from django.utils.decorators import method\_decorator

from django.views.decorators.csrf import csrf\_exempt,csrf\_protect

@method\_decorator(csrf\_exempt,name='dispatch') ---> 指定名字

class UserView(View):

def dispatch(self, request, \*args, \*\*kwargs):

return HttpResponse('...')

[复制代码](javascript:void(0);)

### 注意

能从request里能取到什么：（request里面有什么）--->请求发过来什么，全部都能拿到

GET、POST、method、cookie、session、files、body（这7个为基础）

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## ORM

增删改查

- values/values\_list

- F/Q

- 性能相关

- 原生SQL

### 增

[复制代码](javascript:void(0);)

增：两种(以字典的形式增的话，需打散)

　　# 增加一条数据，可以接受字典类型数据 \*\*kwargs

1. models.User.objects.create(\*\*{}) 直接创建并且提交到数据库

2. user = models.User(\*\*{}) user.save()

models.UserInfo.objects.create()

obj = models.UserInfo(name='xx')  
obj.save()

[复制代码](javascript:void(0);)

### 删

找到对象删除

models.User.objects.filter().delete()

### 改

更新 两种

1. models.User.objects.filter().update(username="一枝花") 支持\*\*kwargs

2. user = models.User.objects.get(id=1)

　　user.username="梁志华"

　　　user.save()

### 查

[复制代码](javascript:void(0);)

models.User.objects.all() # 获取全部

models.User.objects.get(条件) 不满足条件就报错|返回的是一个对象

models.User.objects.filter(条件) 不满足条件不报错，但是返回的结果是一个对象的列表

models.User.objects.filter().first()

models.User.objects.filter().last()

models.User.objects.exclude(条件） # 不包含指定条件

[复制代码](javascript:void(0);)

### 常用操作

### order\_by

# order by

# models.Tb1.objects.filter(name='seven').order\_by('id') # asc

# models.Tb1.objects.filter(name='seven').order\_by('-id') # desc

### group\_by

group by # from django.db.models import Count, Min, Max, Sum

# models.Tb1.objects.filter(c1=1).values('id').annotate(c=Count('num'))

# SELECT "app01\_tb1"."id", COUNT("app01\_tb1"."num") AS "c" FROM "app01\_tb1" WHERE "app01\_tb1"."c1" = 1 GROUP BY "app01\_tb1"."id"

### 靠近原生SQL-->extra\raw

[复制代码](javascript:void(0);)

- extra

def extra(self, select=None, where=None, params=None, tables=None, order\_by=None,   
select\_params=None)

# 构造额外的查询条件或者映射，如：子查询

Entry.objects.extra(select={'new\_id': "select col from sometable where othercol > %s"},  
 select\_params=(1,))

Entry.objects.extra(where=['headline=%s'], params=['Lennon'])

Entry.objects.extra(where=["foo='a' OR bar = 'a'", "baz = 'a'"])

Entry.objects.extra(select={'new\_id': "select id from tb where id > %s"}, s  
elect\_params=(1,), order\_by=['-nid'])

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

- raw

def raw(self, raw\_query, params=None, translations=None, using=None):

# 执行原生SQL

models.UserInfo.objects.raw('select \* from userinfo')

# 如果SQL是其他表时，必须将名字设置为当前UserInfo对象的主键列名

models.UserInfo.objects.raw('select id as nid,name as title from 其他表')

# 为原生SQL设置参数

models.UserInfo.objects.raw('select id as nid from userinfo where nid>%s', params=[12,])

# 将获取的到列名转换为指定列名

name\_map = {'first': 'first\_name', 'last': 'last\_name', 'bd': 'birth\_date', 'pk': 'id'}

Person.objects.raw('SELECT \* FROM some\_other\_table', translations=name\_map)

# 指定数据库

models.UserInfo.objects.raw('select \* from userinfo', using="default")

[复制代码](javascript:void(0);)

### 原生sql-->connection

from django.db import connection, connections

cursor = connection.cursor() # cursor = connections['default'].cursor()

cursor.execute("""SELECT \* from auth\_user where id = %s""", [1])

row = cursor.fetchone() # fetchall()/fetchmany(..)

### 高级一点

### F

# F

#

# from django.db.models import F

# models.Tb1.objects.update(num=F('num')+1)

### Q

[复制代码](javascript:void(0);)

# 方式一：

# Q(nid\_\_gt=10)

# Q(nid=8) | Q(nid\_\_gt=10)

# Q(Q(nid=8) | Q(nid\_\_gt=10)) & Q(caption='root')

# 方式二：

# con = Q()

# q1 = Q()

# q1.connector = 'OR'

# q1.children.append(('id', 1))

# q1.children.append(('id', 10))

# q1.children.append(('id', 9))

# q2 = Q()

# q2.connector = 'OR'

# q2.children.append(('c1', 1))

# q2.children.append(('c1', 10))

# q2.children.append(('c1', 9))

# con.add(q1, 'AND')

# con.add(q2, 'AND')

#

# models.Tb1.objects.filter(con)

[复制代码](javascript:void(0);)

### select\_related/ prefech\_related

[复制代码](javascript:void(0);)

def select\_related(self, \*fields)

性能相关：表之间进行join连表操作，一次性获取关联的数据。

model.tb.objects.all().select\_related()

model.tb.objects.all().select\_related('外键字段')

model.tb.objects.all().select\_related('外键字段\_\_外键字段')

def prefetch\_related(self, \*lookups)

性能相关：多表连表操作时速度会慢，使用其执行多次SQL查询在Python代码中实现连表操作。

# 获取所有用户表

# 获取用户类型表where id in (用户表中的查到的所有用户ID)

models.UserInfo.objects.prefetch\_related('外键字段')

from django.db.models import Count, Case, When, IntegerField

Article.objects.annotate(

numviews=Count(Case(

When(readership\_\_what\_time\_\_lt=treshold, then=1),

output\_field=CharField(),

))

)

students = Student.objects.all().annotate(num\_excused\_absences=models.Sum(

models.Case(

models.When(absence\_\_type='Excused', then=1),

default=0,

output\_field=models.IntegerField()

)))

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

补充

# 1次SQL

# select \* from userinfo

objs = UserInfo.obejcts.all()

for item in objs:

print(item.name)

# n+1次SQL

# select \* from userinfo

objs = UserInfo.obejcts.all()

for item in objs:

# select \* from usertype where id = item.id

print(item.name,item.ut.title)

# 1次SQL

# select \* from userinfo inner join usertype on userinfo.ut\_id = usertype.id

objs = UserInfo.obejcts.all().select\_related('ut') 连表查询

for item in objs:

print(item.name,item.ut.title)

.prefetch\_related()

# select \* from userinfo where id <= 8

# 计算：[1,2]

# select \* from usertype where id in [1,2]

objs = UserInfo.obejcts.filter(id\_\_lte=8).prefetch\_related('ut')

for obj in objs:

print(obj.name,obj.ut.title)

[复制代码](javascript:void(0);)

### 其他

[复制代码](javascript:void(0);)

def annotate(self, \*args, \*\*kwargs)

# 用于实现聚合group by查询

from django.db.models import Count, Avg, Max, Min, Sum

v = models.UserInfo.objects.values('u\_id').annotate(uid=Count('u\_id'))

# SELECT u\_id, COUNT(ui) AS `uid` FROM UserInfo GROUP BY u\_id

v = models.UserInfo.objects.values('u\_id').annotate(uid=Count('u\_id')).filter(uid\_\_gt=1)

# SELECT u\_id, COUNT(ui\_id) AS `uid` FROM UserInfo GROUP BY u\_id having count(u\_id) > 1

v = models.UserInfo.objects.values('u\_id').annotate(uid=Count('u\_id',distinct=True)).filter(uid\_\_gt=1)

# SELECT u\_id, COUNT( DISTINCT ui\_id) AS `uid` FROM UserInfo GROUP BY u\_id having count(u\_id) > 1

def distinct(self, \*field\_names)

# 用于distinct去重

models.UserInfo.objects.values('nid').distinct()

# select distinct nid from userinfo

注：只有在PostgreSQL中才能使用distinct进行去重

def order\_by(self, \*field\_names)

# 用于排序

models.UserInfo.objects.all().order\_by('-id','age')

def extra(self, select=None, where=None, params=None, tables=None, order\_by=None, select\_params=None)

# 构造额外的查询条件或者映射，如：子查询

Entry.objects.extra(select={'new\_id': "select col from sometable where othercol > %s"}, select\_params=(1,))

Entry.objects.extra(where=['headline=%s'], params=['Lennon'])

Entry.objects.extra(where=["foo='a' OR bar = 'a'", "baz = 'a'"])

Entry.objects.extra(select={'new\_id': "select id from tb where id > %s"}, select\_params=(1,), order\_by=['-nid'])

def reverse(self):

# 倒序

models.UserInfo.objects.all().order\_by('-nid').reverse()

# 注：如果存在order\_by，reverse则是倒序，如果多个排序则一一倒序

def defer(self, \*fields):

models.UserInfo.objects.defer('username','id')

或

models.UserInfo.objects.filter(...).defer('username','id')

#映射中排除某列数据

def only(self, \*fields):

#仅取某个表中的数据

models.UserInfo.objects.only('username','id')

或

models.UserInfo.objects.filter(...).only('username','id')

def using(self, alias):

指定使用的数据库，参数为别名（setting中的设置）

##################################################

# PUBLIC METHODS THAT RETURN A QUERYSET SUBCLASS #

##################################################

def raw(self, raw\_query, params=None, translations=None, using=None):

# 执行原生SQL

models.UserInfo.objects.raw('select \* from userinfo')

# 如果SQL是其他表时，必须将名字设置为当前UserInfo对象的主键列名

models.UserInfo.objects.raw('select id as nid from 其他表')

# 为原生SQL设置参数

models.UserInfo.objects.raw('select id as nid from userinfo where nid>%s', params=[12,])

# 将获取的到列名转换为指定列名

name\_map = {'first': 'first\_name', 'last': 'last\_name', 'bd': 'birth\_date', 'pk': 'id'}

Person.objects.raw('SELECT \* FROM some\_other\_table', translations=name\_map)

# 指定数据库

models.UserInfo.objects.raw('select \* from userinfo', using="default")

################### 原生SQL ###################

from django.db import connection, connections

cursor = connection.cursor() # cursor = connections['default'].cursor()

cursor.execute("""SELECT \* from auth\_user where id = %s""", [1])

row = cursor.fetchone() # fetchall()/fetchmany(..)

def values(self, \*fields):

# 获取每行数据为字典格式

def values\_list(self, \*fields, \*\*kwargs):

# 获取每行数据为元祖

def dates(self, field\_name, kind, order='ASC'):

# 根据时间进行某一部分进行去重查找并截取指定内容

# kind只能是："year"（年）, "month"（年-月）, "day"（年-月-日）

# order只能是："ASC" "DESC"

# 并获取转换后的时间

- year : 年-01-01

- month: 年-月-01

- day : 年-月-日

models.DatePlus.objects.dates('ctime','day','DESC')

def datetimes(self, field\_name, kind, order='ASC', tzinfo=None):

# 根据时间进行某一部分进行去重查找并截取指定内容，将时间转换为指定时区时间

# kind只能是 "year", "month", "day", "hour", "minute", "second"

# order只能是："ASC" "DESC"

# tzinfo时区对象

models.DDD.objects.datetimes('ctime','hour',tzinfo=pytz.UTC)

models.DDD.objects.datetimes('ctime','hour',tzinfo=pytz.timezone('Asia/Shanghai'))

"""

pip3 install pytz

import pytz

pytz.all\_timezones

pytz.timezone(‘Asia/Shanghai’)

"""

def none(self):

# 空QuerySet对象

####################################

# METHODS THAT DO DATABASE QUERIES #

####################################

def aggregate(self, \*args, \*\*kwargs):

# 聚合函数，获取字典类型聚合结果

from django.db.models import Count, Avg, Max, Min, Sum

result = models.UserInfo.objects.aggregate(k=Count('u\_id', distinct=True), n=Count('nid'))

===> {'k': 3, 'n': 4}

def count(self):

# 获取个数

def get(self, \*args, \*\*kwargs):

# 获取单个对象

def create(self, \*\*kwargs):

# 创建对象

def bulk\_create(self, objs, batch\_size=None):

# 批量插入

# batch\_size表示一次插入的个数

objs = [

models.DDD(name='r11'),

models.DDD(name='r22')

]

models.DDD.objects.bulk\_create(objs, 10)

def get\_or\_create(self, defaults=None, \*\*kwargs):

# 如果存在，则获取，否则，创建

# defaults 指定创建时，其他字段的值

obj, created = models.UserInfo.objects.get\_or\_create(username='root1', defaults={'email': '1111111','u\_id': 2, 't\_id': 2})

def update\_or\_create(self, defaults=None, \*\*kwargs):

# 如果存在，则更新，否则，创建

# defaults 指定创建时或更新时的其他字段

obj, created = models.UserInfo.objects.update\_or\_create(username='root1', defaults={'email': '1111111','u\_id': 2, 't\_id': 1})

def first(self):

# 获取第一个

def last(self):

# 获取最后一个

def in\_bulk(self, id\_list=None):

# 根据主键ID进行查找

id\_list = [11,21,31]

models.DDD.objects.in\_bulk(id\_list)

def delete(self):

# 删除

def update(self, \*\*kwargs):

# 更新

def exists(self):

# 是否有结果

[复制代码](javascript:void(0);)

### F的使用：数字自增和字符串更新

[复制代码](javascript:void(0);)

# 数字自增 只能用数字

from django.db.models import F

models.UserInfo.objects.update(num=F('num') + 1)

# 字符串更新

from django.db.models.functions import Concat

from django.db.models import Value

models.UserInfo.objects.update(name=Concat('name', 'pwd'))

models.UserInfo.objects.update(name=Concat('name', Value('666')))

[复制代码](javascript:void(0);)

### ****MTV和MVC****

MVC： model view controller

MTV： model tempalte view

### 遇到的难题：

　　- CBV是添加csrf装饰器  
　　- 多数据库配置 allow\_relation方法

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## [多数据库相关操作](http://www.cnblogs.com/jassin-du/p/9052841.html)

[复制代码](javascript:void(0);)

相关执行命令

python manage.py makemigraions

python manage.py migrate app名称 --databse=配置文件数据名称的别名

手动操作：

models.UserType.objects.using('db1').create(title='普通用户')

result = models.UserType.objects.all().using('default')

自动操作：

class Router1:

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth models go to auth\_db.

"""

return 'db1'

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth models go to auth\_db.

"""

return 'default'

配置：

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

},

'db1': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db1.sqlite3'),

},

}

DATABASE\_ROUTERS = ['db\_router.Router1',]

使用：

models.UserType.objects.create(title='VVIP')

result = models.UserType.objects.all()

print(result)

[复制代码](javascript:void(0);)

### 补充-->精细粒度

[复制代码](javascript:void(0);)

粒度更细

class Router1:

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth models go to auth\_db.

"""

if model.\_meta.model\_name == 'usertype':

return 'db1'

else:

return 'default'

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth models go to auth\_db.

"""

return 'default'

[复制代码](javascript:void(0);)

### 问题：app01中的表在default数据库创建app02中的表在db1数据库创建

[复制代码](javascript:void(0);)

问题：

app01中的表在default数据库创建

app02中的表在db1数据库创建

# 第一步：

python manage.py makemigraions

# 第二步：

app01中的表在default数据库创建

python manage.py migrate app01 --database=default

# 第三步：

app02中的表在db1数据库创建

python manage.py migrate app02 --database=db1

# 手动操作：

m1.UserType.objects.using('default').create(title='VVIP')

m2.Users.objects.using('db1').create(name='VVIP',email='xxx')

# 自动操作：

配置：

class Router1:

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth models go to auth\_db.

"""

if model.\_meta.app\_label == 'app01':

return 'default'

else:

return 'db1'

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth models go to auth\_db.

"""

if model.\_meta.app\_label == 'app01':

return 'default'

else:

return 'db1'

DATABASE\_ROUTERS = ['db\_router.Router1',]

使用：

m1.UserType.objects.using('default').create(title='VVIP')

m2.Users.objects.using('db1').create(name='VVIP',email='xxx')

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

其他：

数据库迁移时进行约束：

class Router1:

def allow\_migrate(self, db, app\_label, model\_name=None, \*\*hints):

"""

All non-auth models end up in this pool.

"""

if db=='db1' and app\_label == 'app02':

return True

elif db == 'default' and app\_label == 'app01':

return True

else:

return False

# 如果返回None，那么表示交给后续的router，如果后续没有router，则相当于返回True

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth models go to auth\_db.

"""

if model.\_meta.app\_label == 'app01':

return 'default'

else:

return 'db1'

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth models go to auth\_db.

"""

if model.\_meta.app\_label == 'app01':

return 'default'

else:

return 'db1'

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## websocket

### 什么是轮询

通过定时器让程序每隔n秒执行一次操作。

### 什么是长轮询

客户端向服务器发送请求，服务器接到请求后hang住连接，等待30秒，30s过后再重新发起请求，  
直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。

### 轮询和长轮询的目的

由于Http请求是无状态、短连接所以服务端无法向客户端实时推送消息，  
所以，我们就是可以使用：轮训和长轮训去服务端获取实时数据。

### 什么是websocket

[复制代码](javascript:void(0);)

websocket是给浏览器新建的一套（类似与http）协议，协议规定：（\r\n分割）浏览器和服务器连接之后不断开，  
　　　　以此完成：服务端向客户端主动推送消息。

websocket协议额外做的一些操作

握手 ----> 连接钱进行校验

加密 ----> payload\_len=127/126/<=125 --> mask key

[复制代码](javascript:void(0);)

### http协议

[复制代码](javascript:void(0);)

　　HTTP协议本质：通过\r\n分割的规范+ 请求响应之后断开链接 == > 无状态、 短连接

具体：

　　Http协议是建立在tcp之上的，是一种规范，它规范定了发送的数据的数据格式，

然而这个数据格式是通过\r\n 进行分割的，请求头与请求体也是通过2个\r\n分割的，响应的时候，

响应头与响应体也是通过\r\n分割，并且还规定已请求已响应就会断开链接

即---> 短连接、无状态

[复制代码](javascript:void(0);)

### websocket本质

[复制代码](javascript:void(0);)

创建一个连接后不断开的socket

当连接成功之后：

客户端（浏览器）会自动向服务端发送消息，包含： Sec-WebSocket-Key: iyRe1KMHi4S4QXzcoboMmw==

服务端接收之后，会对于该数据进行加密：base64(sha1(swk + magic\_string))

构造响应头：

HTTP/1.1 101 Switching Protocols\r\n

Upgrade:websocket\r\n

Connection: Upgrade\r\n

Sec-WebSocket-Accept: 加密后的值\r\n

WebSocket-Location: ws://127.0.0.1:8002\r\n\r\n

发给客户端（浏览器）

建立：双工通道，接下来就可以进行收发数据

发送数据是加密，解密，根据payload\_len的值进行处理

payload\_len <= 125

payload\_len == 126

payload\_len == 127

获取内容：

mask\_key

数据

根据mask\_key和数据进行位运算，就可以把值解析出来。

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## Flask

[复制代码](javascript:void(0);)

问题一：flask和django的区别：

django：大而全的全的框架，重武器；内置很多组件：ORM、admin、Form、ModelForm、中间件、信号、缓存、csrf等

flask: 微型框架、可扩展强，如果开发简单程序使用flask比较快速，如果实现负责功能就需要引入一些组件：

　　　　　　flask-session/flask-SQLAlchemy/wtforms/flask-migrate/flask-script/blinker

这两个框架都是基于wsgi协议实现的，默认使用的wsgi模块不一样。

还有一个显著的特点，他们处理请求的方式不同：

django： 通过将请求封装成Request对象，再通过参数进行传递。

flask：通过上下文管理实现。

问题1.1: flask上下文管理：

　　简单来说，falsk上下文管理可以分为三个阶段：

　　　　　　　　1、请求进来时，将请求相关的数据放入上下问管理中

　　　　　　　　2、在视图函数中，要去上下文管理中取值

　　　　　　　　3、请求响应，要将上下文管理中的数据清除

　　详细点来说：

　　　　　　　　1、请求刚进来，将request，session封装在RequestContext类中，app，g封装在AppContext类中，并通过LocalStack将requestcontext和appcontext放入Local类中

　　　　　　　　2、视图函数中，通过localproxy--->偏函数--->localstack--->local取值

　　　　　　　　3、请求相应时，先执行save.session()再各自执行pop(),将local中的数据清除

问题1.2 flask第三方组件

　　flask：

　　　　　　-flask-session 默认放入cookie,可以放入redis

　　　　　　-flask-redis

　　　　　　-flask-migrate

　　　　　　-flask-script

　　　　　　-blinker 信号

　公共： DBUtils 数据库连接池

　　　　　　wtforms 表单验证+生成HTML标签

　　　　　　sqlalchemy

　　自定义：Auth 参考falsk-login

问题二：Flask中的session是什么时候创建，什么时候销毁的？

　　当请求进来时，会将requset和session封装为一个RequestContext对象，通过LocalStack将RequestContext放入到Local对象中，因为

请求第一次来session是空值，所以执行open\_session,给session（uuid4()）赋值，再通过视图函数处理，请求响应时执行save.session,将签名session写入cookie中，

　　再将Local中的数值pop掉。

问题三：flask中一共有几个LocalStack和Local对象

　　两个LocalStack，两个Local

　　request、session共同用一个LocalStack和Local

　　g、app共同用一个Localstack和Local

问题四： 为什么把请求放到RequestContext中：

　　　因为request和session都是在视图中操作频繁的数据，也是用户请求需要用的数据，将request和session封装在RequestContext中top，

　　pop一次就可以完成，而单独不封装在一起就会多次操作，

　　　　ctx = RequestContext(request,session)

问题五：local作用

　　　　-保存 请求上下文对象和app上下文对象

　　　　　-localstack的源码与threading.local（线程处理）作用相似，不同之处是Local是通过greenlet（协程）获取唯一标识，粒度更细

问题六：Localstack作用

　　　　2、将local对象中的数据维护成一个栈【ctx,ctx】（先进后出）

{

“协程或线程的唯一标识”: { stack:[ctx,ctx,ctx,] }

}

为什么维护成一个栈？

　　 当是web应用时：不管是单线程还是多线程，栈中只有一个数据

　　 - 服务端单线程：

　　　　{

　　　　111:{stack: [ctx, ]}

　　　　}

　　- 服务端多线程：

　　　　{

　　　　111:{stack: [ctx, ]}

　　　　112:{stack: [ctx, ]}

　　　　}

离线脚本：可以在栈中放入多个数据,在任何情况下都可以获取到当前app的请求和响应

with app01.app\_context():

　　print(current\_app)

　　with app02.app\_context():

　　　　print(current\_app)

　　print(current\_app）

问题七：什么是g?

　　　　g 相当于一次请求的全局变量，当请求进来时将g和current\_app封装为一个APPContext类，在通过LocalStack将Appcontext放入Local中，

　　取值时通过偏函数，LocalStack、loca　l中取值，响应时将local中的g数据删除：

问题八：怎么获取Session/g/current\_app/request

　　　　通过 、偏函数（lookup\_req\_object）、Localstack、Local取值

1、反射

　　-CBV

　　-django配置文件

　　-wtforms中的Form()示例化中 将"\_fields中的数据封装到From类中"

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## 栈

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 栈

[回到顶部](https://www.cnblogs.com/jassin-du/articles/8993722.html#_labelTop)

## redis

### 基本操作

[复制代码](javascript:void(0);)

- 连接

- 直接连接：

import redis

r = redis.Redis(host='10.211.55.4', port=6379)

r.set('foo', 'Bar')

print r.get('foo')

- 连接池：

import redis

pool = redis.ConnectionPool(host='10.211.55.4', port=6379)

r = redis.Redis(connection\_pool=pool)

r.set('foo', 'Bar')

print r.get('foo')

[复制代码](javascript:void(0);)

### 5大数据类型

[复制代码](javascript:void(0);)

- 5大数据类型

- 字符串 "sadf" (只有字符串有默认的超时时间)

- set('k1','123',ex=10)

- get

- mset

- mget

- incr

- 超时时间：

import redis

r = redis.Redis(host='10.211.55.4', port=6379)

r.set('foo', 'Bar',ex=10)

- 字典 {'k1':'v1'}

- hset(name, key, value)

- hmset(name, mapping)

- hget(name,key)

- 超时时间（字典，列表、集合、有序结合相同)：自定义

import redis

conn = redis.Redis(host='10.211.55.4', port=6379)

conn.hset('n1', 'k1','123123')

conn.expire('n1',10)

- 如果一个字典在redis中保存了10w个值，我需要将所有值全部循环并显示，请问如何实现？

for item in r.hscan\_iter('k2',count=100):

print item

- 列表 [11,11,22,33] 没有默认的超时时间

- lpush

- rpush

- lpop

- blpop

- rpop

- brpop

- llen

- lrange

- 如果一个列表在redis中保存了10w个值，我需要将所有值全部循环并显示，请问如何实现？

一个一个取值，列表没有iter方法，但能自定义  
　　　　 def list\_scan\_iter(name,count=3):

start = 0

while True:

result = conn.lrange(name, start, start+count-1)

start += count

if not result:

break

for item in result:

yield item

for val in list\_scan\_iter('num\_list'):

print(val)  
　　场景：投票系统，script-redis

- 集合 {'alex','oldboy','日天'}

- 有序集合 {('alex',59),('oldboy',100),('日天',1)}

[复制代码](javascript:void(0);)

- 公共操作：

- delete(\*names)

- keys(pattern='\*')

expire(name ,time)

...

### 内置有16个小数据库（分离）

### 事务

[复制代码](javascript:void(0);)

import redis

pool = redis.ConnectionPool(host='10.211.55.4', port=6379)

conn = redis.Redis(connection\_pool=pool)

# pipe = r.pipeline(transaction=False)

pipe = conn.pipeline(transaction=True)

# 开始事务

pipe.multi()

pipe.set('name', 'alex')

pipe.set('role', 'sb')

pipe.lpush('roless', 'sb')

# 提交

pipe.execute()

注意：咨询是否当前分布式redis是否支持事务

[复制代码](javascript:void(0);)

### 检测，watch

### 假如开发商城，剩余多少件商品，如何控制剩余数量不会出问题?

[复制代码](javascript:void(0);)

**方案一：通过redis的watch实现**  
import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

# conn.set('count',1000)

val = conn.get('count')

print(val)

with conn.pipeline(transaction=True) as pipe:

# 先监视，自己的值没有被修改过

conn.watch('count')

# 事务开始

pipe.multi()

old\_count = conn.get('count')

count = int(old\_count)

print('现在剩余的商品有:%s',count)

input("问媳妇让不让买？")

pipe.set('count', count - 1)

# 执行，把所有命令一次性推送过去

pipe.execute()

[复制代码](javascript:void(0);)

**方案二：通过数据库，加锁for update**

### 发布和订阅

只要有任务就所有订阅者每人一份

[复制代码](javascript:void(0);)

发布者：

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

conn.publish('104.9MH', "hahahahahaha")

订阅者：

import redis

conn = redis.Redis(host='127.0.0.1',port=6379)

pub = conn.pubsub()

pub.subscribe('104.9MH')

while True:

msg= pub.parse\_response()

print(msg)

[复制代码](javascript:void(0);)

### 主从复制

[复制代码](javascript:void(0);)

优势：

- 高可用

- 分担主压力

注意：

- slave设置只读

从的配置文件添加以下记录，即可：

slaveof 1.1.1.1 3306

[复制代码](javascript:void(0);)

### sentinel，哨兵

[复制代码](javascript:void(0);)

启动主redis:

redis-server /etc/redis-6379.conf 启动主redis

redis-server /etc/redis-6380.conf 启动从redis

在linux中：

找到 /etc/redis-sentinel-8001.conf 配置文件，在内部：

- 哨兵的端口 port = 8001

- 主redis的IP，哨兵个数的一半/1

找到 /etc/redis-sentinel-8002.conf 配置文件，在内部：

- 哨兵的端口 port = 8002

- 主redis的IP, 1

启动两个哨兵

[复制代码](javascript:void(0);)

### redis-cluster集群方案

redis集群、分片、分布式redis

redis-py-cluster

### 问题：

目前你们公司项目1000用户，QPS=1000 ，如果用户猛增10000w?  
项目如何提高的并发？

1. 数据库读写分离  
2. 设置缓存  
3. 负载均衡

### 用redis做过什么

[复制代码](javascript:void(0);)

　　-配合django做过缓存，将常用且不易修改的数据放进来（博客）

　　-购物车信息

　　-Session

　　　　-缓存配置文件

　　　　-session配置文件中指定使用缓存

　　-rest api中访问频率控制

　　-基于flask、websocket实现的投票系统（redis做消息队列）

　　-scrapy中

　　　　- 去重规则

　　　　- 调度器：先进先出、后进先出、优先级队列

　　　　-pipeline

　　　　-起始url

　　- 商品热点信息

　　- 计数器

　　- 排行（有序集合）

[复制代码](javascript:void(0);)

### 为什么redis要做主从复制

[复制代码](javascript:void(0);)

目的是对redis做高可用，为每一个redis实例创建一个备份称为slave，让主和备之间进行数据同步，save/bsave。

主：写

从：读

优点：

- 性能提高，从分担读的压力。

- 高可用，一旦主redis挂了，从可以直接代替。

存在问题：当主挂了之后，需要人为手工将从变成主。

[复制代码](javascript:void(0);)

### redis的sentinel是什么？

帮助我们自动在主从之间进行切换

检测主从中 主是否挂掉，且超过一半的sentinel检测到挂了之后才进行进行切换。

如果主修复好了，再次启动时候，会变成从。

### redis的cluster是什么?

[复制代码](javascript:void(0);)

集群方案：

- redis cluster 官方提供的集群方案。

- codis，豌豆荚技术团队。

- tweproxy，Twiter技术团队。

redis cluster的原理？

- 基于分片来完成。

- redis将所有能放置数据的地方创建了 16384 个哈希槽。

- 如果设置集群的话，就可以为每个实例分配哈希槽：

- 192.168.1.20【0-5000】

- 192.168.1.21【5001-10000】

- 192.168.1.22【10001-16384】

- 以后想要在redis中写值时，

set k1 123

将k1通过crc16的算法，将k1转换成一个数字。然后再将该数字和16384求余，如果得到的余数 3000，那么就将该值写入到 192.168.1.20 实例中。

[复制代码](javascript:void(0);)

### redis是否可以做持久化？

[复制代码](javascript:void(0);)

RDB：每隔一段时间对redis进行一次持久化。

- 缺点：数据不完整

- 优点：速度快

AOF：把所有命令保存起来，如果想到重新生成到redis，那么就要把命令重新执行一次。

- 缺点：速度慢，文件比较大

- 优点：数据完整

[复制代码](javascript:void(0);)

### redis的过期策略

[复制代码](javascript:void(0);)

voltile-lru： 从已设置过期时间的数据集（server.db[i].expires）中挑选最近频率最少数据淘汰

volatile-ttl： 从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru： 从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random： 从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

[复制代码](javascript:void(0);)

### redis的分布式锁实现。

[复制代码](javascript:void(0);)

- 写值并设置超时时间

- 超过一半的redis实例设置成功，就表示加锁完成。

- 使用：安装redlock-py

from redlock import Redlock

dlm = Redlock(

[

{"host": "localhost", "port": 6379, "db": 0},

{"host": "localhost", "port": 6379, "db": 0},

{"host": "localhost", "port": 6379, "db": 0},

]

)

# 加锁，acquire

my\_lock = dlm.lock("my\_resource\_name",10000)

if my\_lock:

# J进行操作

# 解锁,release

dlm.unlock(my\_lock)

else:

print('获取锁失败')