目录

欢迎前往GitHub贡献自己的问题和答案
前往有道云浏览（推荐）
前往CSDN浏览

[TOC]

Python模块

01.python中有哪些可变类型与不可变类型

初级 Python

- 可变类型：会在原来的内存地址上修改元素 比如： 列表，字典
- 不可变类型：不会在原来的内存地址上修改元素，而是指向了新的内存引用 比如：整型，字符串，元组

返回目录

02.栈和堆的区别是什么

初级 Python

1. 申请方式的不同。栈由系统自动分配，而堆是人为申请开辟；
2. 申请大小的不同。栈获得的空间较小，而堆获得的空间较大；
3. 申请效率的不同。栈速度较快，堆速度比较慢;
4. 底层不同。栈是连续的空间，堆是不连续的空间，是一棵完全二叉树。
5. 存储内容的不同。==栈在函数调用时，第一个进栈的是主函数中的下一条指令的地址，然后是函数的各个参数，== 在大多数C编译器中，参数是由右向左入栈的，==然后是函数中的局部变量，注意静态变量是不入栈的，== 静态变量存储在静态存储区。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行；==堆一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容由程序员安排。==

详情
返回目录

03.简述数组，链表，队列，堆栈的区别

初级 Python

数组和链表是存储方式的概念，数组在连续的空间中存储数据，链表在非连续的空间中存储数据；
队列和堆栈是描述数据存取方法的概念，队列是先进先出，而堆栈是后进后出，队列和堆栈可以用链表来实现，也可以用数组来实现

返回目录

04.深拷贝和浅拷贝的区别是什么

高级 Python

首先深拷贝和浅拷贝都是对象的拷贝，都会生成一个看起来相同的对象，他们本质的区别是拷贝出来的对象的地址是否和原对象一样，也就是地址的复制还是值的复制的区别。

- **浅拷贝是对一个对象父级（外层）的拷贝，并不会拷贝子级（内部）**

> 使用浅拷贝的时候，分为两种情况：
>
> > 1. 如果最外层的数据类型是可变的，比如说列表，字典等，浅拷贝会开启新的地址空间去存放。
> > 2. 如果最外层的数据类型是不可变的，比如元组，字符串等，浅拷贝对象的时候，还是引用对象的地址空间。

- **深拷贝对一个对象是所有层次的拷贝（递归），内部和外部都会被拷贝过来。**

> 深拷贝也分两种情况：
>
> > 1. 最外层数据类型可变。这个时候，内部和外部的都会拷贝过来。
> > 2. 外层数据类型不可变，如果里面是可变数据类型，会新开辟地址空间存放。如果内部数据类型不可变，才会如同浅拷贝一样，是对地址的引用。

详情

返回目录

05.面向对象的3个特性是什么

初级 Python

1. 封装：根据职责将属性和方法封装到一个抽象的类中定义类的准则
2. 继承：实现代码的重用，相同的代码不需要重复的编写
3. 多态：不同的子类调用相同的父类，产生不同的结果

返回目录

06.什么是闭包

高级 Python

闭包是由两个函数嵌套定义，内部函数里面用到了外部函数里面的变量值，那么这个变量（变量值）加上内部还是里面的代码组成的代码块，组成了一个新的内存空间，我们把这个空间叫做闭包。 闭包 = 函数+环境变量（在函数定义的时候定义，它不是全局变量,这个环境变量一定要被内部函数调用才算是闭包）

答案来源
返回目录

07.匿名函数/函数/闭包/对象在做实参时有什么区别

高级 Python

1. **匿名函数**：能够完成基本的简单功能，传递是这个函数的引用，只有功能
2. **普通函数**：能够完成比较普通的功能，传递是这个函数的引用，只有功能
3. **闭包**：能够完成比较复杂的功能，传递是这个闭包中的函数和数据，因此传递是功能+数据
4. **对象**：能够完成最为复杂的功能，传递是很多数据和很对功能，因此传递是功能+数据

返回目录

08.简述什么是进程、线程、协程

高级 Python

1. 进程是系统进行资源分配和调度的一个独立单位，一个程序至少有一个进程,一个进程至少有一个线程。
2. 线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但它可与同属一个进程的其他的线程共享进程所拥有的全部资源。线程不能够独立执行，必须依存在进程中
3. 协程是一种比线程更加轻量级的存在，一个线程也可以拥有多个协程。

09.简述进程、线程、协程三者的区别

高级 Python

1. 进程切换需要的资源很大，效率相对低
2. 线程切换需要的资源一般，效率比进程高
3. 协程切换任务资源很小，三者中效率最高
4. 多进程、多线程根据CPU核数不一样可能是并行，但是协程是在一个线程中所以是并发。

10.简述进程、线程、协程适用于那种应用场景类型

高级 Python

- 计算密集型：用进程
- IO密集型：用线程或协程

返回目录

11.协程为何比线程还快

高级 Python

高并发+高扩展性+低成本：一个CPU支持上万的协程都不是问题。所以很适合用于高并发处理协程能保留

上一次调用时的状态，不管是进程还是线程，每次阻塞、切换都需要陷入系统调用，使用线程时需要非常小心地处理同步问题，而协程完全不存在这个问题。

返回目录

12.什么是迭代器，为什么要使用它

高级 Python

可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator

```
from collections import Iterator

def Iteror2():
    '''迭代器原理'''
    list = [1, 2, 3, 4]
    it = iter(list)  # 创建迭代器
    while True:
        try:
            print(next(it))
        except StopIteration:
            sys.exit()

# 判断是否为迭代器
isinstance((x for x in range(10)), Iterator)
#>>> True
```

迭代是访问集合元素的一种方式。 ==迭代器保存的是获取数据的方式而不是结果，所以想用的时候就可以生成，节省大量内存空间，它是一个可以记住遍历的位置的对象。== 迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。

迭代器有两个基本的方法：iter() 和 next()。字符串，列表或元组对象都可用于创建迭代器。

返回目录

13.什么是生成器，为什么要使用它

高级 Python

生成器定义在Python中,一边循环一边计算的机制，使用了 yield 的函数被称为生成器（generator）。跟普通函数不同的是，生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个特殊的迭代器。 在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息， 返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。 调用一个生成器函数，返回的是一个迭代器对象。

```
list5 = [x for x in range(5)]
print(list5)   #output：[0, 1, 2, 3, 4]
```

列表所有数据都在内存中，如果有海量数据的话将会非常耗内存。如仅需要访问前面几个元素，那后面绝大多数元素占用的空间都白白浪费了。如果列表元素按照某种算法推算出来，那我们就可以在循环的过程中不断推算出后续的元素，这样就不必创建完整的list，从而节省大量的内存空间。

==简单一句话：我又想要得到庞大的数据，又想让它占用内存空间少，那就用生成器！== 生成器仅仅保存了一套生成数值的算法，并且没有让这个算法现在就开始执行，而是我什么时候调它，它什么时候开始计算一个新的值，并给你返回。

返回目录

14.简述迭代器、生成器的区别

高级 Python

能使用for遍历的就叫可迭代对象，能使用next方法的就是迭代器，生成器是特殊的迭代器。生成器能做到迭代器能做的所有事,而且因为自动创建了 iter()和 next()方法,生成器显得特别简洁和高效，使用生成器表达式取代列表解析可以同时节省内存空间。

返回目录

15.什么是装饰器，为什么要使用它

高级 Python

- 装饰器本质上是一个函数，这个函数的主要作用是包装另一个函数或类
  包装的- 目的是在不改变原函数名的情况下改变被包装对象的行为。
- 接收一个函数，内部对其包装，然后返回一个新函数，这样子动态的增强函数功能
- 通过高阶函数传递函数参数，新函数添加旧函数的需求，然后执行旧函数。

在django中middkeware中间件 其实就是高级的装饰器用法。

返回目录

16.什么是线程安全

高级 Python

线程安全：就是对于多线程同时操作是是安全的而不会发生写冲突,比如python的Queue
非线程安全：就是多线成同时操作时会发生写冲突,比如python的list,set,dict
返回目录

17.标准库有哪些线程安全的队列

高级 Python

Python Queue模块有三种队列:

1. FIFO队列先进先出.(线程安全)
2. LifoQueue类似于堆,即先进后出(线程安全)
3. PriorityQueue优先级队列,级别越低,越先出来(线程安全)

对应的构造函数:

1. class Queue.Queue(maxsize) FIFO
2. class Queue.LifoQueue(maxsize) LIFO
3. class Queue.PriorityQueue(maxsize) 优先级队列

返回目录

18.什么是GIL

高级 Python

GIL是python中的全局解释器锁，是不可控的，同一个进程中，假如有多个线程在运行，那么其中一个线程在运行的时候就会霸占GIL锁，就使得其他线程无法运行，等该线程运行结束以后，其他线程才能运行。如果线程中遇到耗时操作(I/O密集型任务)，则解释器锁会解开，使得其他线程运行，所以说在多线程中，线程的运行仍是有先后顺序的，并不是同时进行。

返回目录

19.什么时候释放GIL锁

高级 Python

1. 时间片耗尽(cpu时间)
2. 任务遇到I/O等待时
3. 执行任务结束
4. 执行到字节码阈值时

返回目录

20.互斥锁与GIL的区别

高级 Python

互斥锁是在多线程的情况下，确保当前线程执行完成后，再执行下一个任务，当前任务没有结束，下个任务会阻塞。

GIL是保证同一时间只有1个线程在执行，但是该线程让出GIL的时，有可能并没完成该线程的任务，该线程的任务分多少次执行完成这个会安装GIL的默认策略。

返回目录

21.python高并发解决方案有哪些

高级 Python

- gevent 代码看起来好看一些,但是维护比较差,patch没有规律,而且里面封装了C,对python3的支持最差.
- twisted 稳定性是最好的,但是需要较长时间的学习.对python3的支持较差.
- tornado 兼容性最好.但是过于简单了,功能不强,另外没有python数据库适配器能和tornado无缝对接,因此调用数据库很麻烦,而且只支持web.

返回目录

22.谈谈对不定长参数的理解

初级 Python

一般分为两种：

- 一种是*args 位置参数 在定义函数时，在形参前面加一个*，代表可以接收任意多个实参，用元组类型保存所有数据。一般写成def function(*args)
- 一种是**kwargs 命名参数 代表可以接收任意多个的命名参数，用字典类型保存。

返回目录

23.谈谈对缺省参数的理解

初级 Python

如果调用函数的时候，传递了对应位置的实参，那就使用这个传递的值，如果没有传递对应的值，那就使用缺省参数的值。

返回目录

24.break和continue的区别

初级 Python

- break和continue都是用于while嵌套循环中
- continue是结束内层的while循环，但并没有终止整个循环
- break是结束整个while循环

返回目录

25.is 和 ==的区别

高级 Python

- “==”仅判断A和B的值是否相等
- is 不仅是会判断A和B的值是否相等，还会判断A和B的id是否一致

返回目录

26.__new__() 和 __init__()的区别

高级 Python

_new_作用于_init_之前。前者可以决定是否调用后者，可以决定调用哪个类的_init_方法。

返回目录

27.range和xrange的区别

初级 Python

xrange和range 的用法完全相同，但是xrange返回的是一个生成器。

返回目录

28. yield和return的相同点和区别

高级 Python

- 相同点：功能都是返回程序执行结果
- 区别：yield返回执行结果并不中断程序执行，return在返回执行结果的同时中断程序执行。

返回目录

29.列举5个python常用标准库并说明其作用

初级 Python

- os：提供不少于操作系统相关联的函数
- sys：通常用于命令行参数
- datetime：日期时间
- re：正则匹配
- math：数学运算

返回目录

30.谈谈你知道的几种设计模式

高级 Python

- 单例模式：保证一个类仅有一个实例，并提供一个访问他的全局访问点，例如框架中的数据库连接
- 装饰器模式：不修改元类代码和继承的情况下动态扩展类的功能，例如框架中的每个controller文件会提供before和after方法。
- 迭代器模式： 提供一个方法顺序访问一个聚合对象中各个元素，在PHP中将继承 Iterator 类
- 命令模式: 将"请求"封闭成对象, 以便使用不同的请求,队列或者日志来参数化其他对象. 命令模式也支持可撤销的操作.

返回目录

31.Python2和Python3的区别

1. python解释器默认编码

python2 解释器默认编码：ascii
python3 解释器默认编码：utf-8

2. 输入

python2：name=raw_input('请输入姓名')
python3：name=input('请输入你的姓名')

3. 输出

python2：print "你好"
python3：print("你好")

4. 数字表示

python2 64位机器，范围-2^63~2^63-1 超出上述范围，python自动转化为long(长整型) 注：long(长整型)数字末尾有一个L
python3 所有整型都是int，没有long(长整型)

5.整型除法

python2：只能保留整数位
python3：可以保留所有内容

6. range / xrange

python2：
xrange：不会在内存中立即创建，而是在循环时，边循环边创建
range：在内存立即把所有的值创建
python3：
只有range，相当于python2中的xrange
range：不会在内存中立即创建，而是在循环时，边循环边创建

7. 包的定义

python2：文件夹中必须有_ _ init _ *.py文件*
python3：不需要有 _ init _ _.py文件

8. 字典的keys / values / items方法

python2：返回列表，通过索引可以取值
python3：返回迭代器，只能通过循环取值，不能通过索引取值

9. map / filter

python2：返回列表，直接创建值，可以通过索引取值
python3：返回迭代器，不直接创建值，通过循环，边循环边创建

10. str(字符串类型)的区别(最大区别，优先写这个)

python2：
str类型，相当于python3中的字节类型，utf-8/gbk等其他编码
unicode类型，相当于python3中的字符串类型，unicode编码
python2中没有字节类型
python3：
str类型，字符串类型，unicode编码
python3中没有unicode类型

11. 继承object

```
class Foo:
    pass
class Foo(object):
    pass
# 在python3中这俩的写法是一样，因为所有的类默认都会继承object类，全部都是新式类。
# 如果在python2中这样定义，则称其为：经典类
class Foo:
    pass
# 如果在python2中这样定义，则称其为：新式类
class Foo(object):
    pass
# 新式类
# 继承object
    # 支持super
    # 多继承 广度优先C3算法
    # mro方法
# 经典类
    # py2中不继承object
    # 没有super语法
    # 多继承 深度优先
    # 没有mro方法
```

返回目录

Linux模块

01.列举10个常见的Linux命令

初级 Linux

```
cd pwd touch ls mkdir rm help sudo ssh date clear vim ps cat more scp cp find mv
grep echo
```

返回目录

HTTP模块

01.HTTP是什么

初级 HTTP

1. HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。
2. HTTP协议是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。
3. HTTP协议是一个属于应用层的面向对象的协议。
4. HTTP协议工作于客户端和服务端架构上。浏览器作为HTTP客户端通过URL向HTTP服务器发送请求，服务器根据接收到的请求后，向客户端发送响应信息，

- 基于TCP/IP

  双方建立通信的顺序,以及Web页面显示需要 处理的步骤,等等。像这样把与互联网相关联的协议集合起来总称为 TCP/IP。而HTTP协议是基于TCP/IP协议之上的应用层协议。

- 基于请求－响应模式

  HTTP协议规定,请求从客户端发出,最后服务器响应该请求并返回

- 无状态保存

  HTTP是一种无状态协议。HTTP协议不对请求和响应之间的通信状态进行保存，不做持久化处理。这是为了更快地处理大量事务,确保协议的可伸缩性,而特意把HTTP协议设计成 如此简单的。

  可是,随着Web的不断发展,因无状态而导致业务处理变得棘手的情况增多了。比如,用户登录到一家购物网站,即使他跳转到该站的 其他页面后,也需要能继续保持登录状态。

  针对这个实例,网站为了能够掌握是谁送出的请求,需要保存用户的状态。HTTP/1.1虽然是无状态协议,但为了实现期望的保持状态功能, 于是引入了Cookie技术。有了Cookie再用HTTP协议通信,就可以管 理状态了。

返回目录

02.HTTP请求报文与响应报文格式

初级 HTTP

- **请求报文包含三部分：**

1. 请求行(包含请求方法、URL、HTTP版本)
2. 请求首部字段
3. 请求内容实体

- **响应报文包含三部分：**

1. 状态行(包含状态码、状态码的原因短语、HTTP版本)
2. 响应首部字段
3. 响应内容实体

返回目录

03.HTTP常见首部字段有哪些

初级 HTTP

1. 通用首部字段（请求报文与响应报文都会使用的首部字段）

字段	字段描述
Date	创建报文时间
Connection	连接的管理
Cache-Control	缓存的控制
Transfer-Encoding	报文主体的传输编码方式

2. 请求首部字段（请求报文会使用的首部字段）

字段	字段描述
Host	请求资源所在服务器
Accept	可处理的媒体类型
Accept-Charset	可接收的字符集
Accept-Encoding	可接受的内容编码
Accept-Language	可接受的自然语言

3. 响应首部字段（响应报文会使用的首部字段）

字段	字段描述
Accept-Ranges	可接受的字节范围
Location	令客户端重新定向到的URI
Server	HTTP服务器的安装信息

4. 实体首部字段（请求报文与响应报文的实体部分使用的首部字段）

字段	字段描述
Allow	资源可支持的HTTP方法
Content-Type	实体主类的类型
Content-Encoding	实体主体适用的编码方式
Content-Language	实体主体的自然语言
Content-Length	实体主体的的字节数
Content-Range	实体主体的位置范围，一般用于发出部分请求时使用

返回目录

04.HTTP与HTTPS的区别

初级 HTTP HTTPS

- HTTP协议传输的数据都是明文、未加密的。因此使用HTTP协议传输隐私信息非常不安全。
- HTTPS协议是由HTTP+SSL构建的网络协议，可进行加密传输、身份认证，要比HTTP协议安全。

不同点	HTTP	HTTPS
安全证书	不需要	需要到ca申请证书（免费/付费可选）
是否加密	不加密，信息是明文传输	具有安全性的ssl加密传输协议
验证数据	不验证数据，可能遭到伪装 无法验证报文完整性，可能被篡改	验证数据且对数据完整性保护
连接方式	无状态连接	HTTPS协议是由HTTP+SSL协议构建的 可进行加密传输、身份认证的网络协议，比HTTP协议安全。
默认端口	80	443
安全性	不安全	安全

返回目录

05.列举HTTP请求中常见的请求方式

初级 HTTP

方法名称	定义
GET	向特定的路径资源发出请求
POST	向指定路径资源提交数据进行处理请求（一般用于提交表单或者上传文件）
PUT	从客户端向服务器传送数据更新指定的资源
PATCH	从客户端向服务器传送数据更新部分指定的资源
DELETE	请求服务器删除指定的资源
HEAD	向服务器索要与GET一样的请求，但是不返回返回体。 这个方法可以在不必传输整个响应内容的情况下，获取包含在响应消息头中的元信息
OPTIONS	查询相应URL支持的HTTP方法
TRACE	返回服务器收到的请求，主要用于测试或诊断
CONNECT	HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务

返回目录

06.GET方法与POST方法的区别

初级 HTTP

不同点	GET	POST
本质	产生1个TCP数据包，只跑1次 浏览器会把HTTP header和data一并发送出去，服务器响应200（返回数据）	产生2个TCP数据包，来回各1次（共2次） 浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）
请求形式	参数通过URL传递	把数据放在Request的body中
安全性	因为URL是可见的，可能会泄露私密信息， 如账号密码等，所以不安全	数据在request中，用户不可见，较get安全性较高
传输数据量	传输的数据量小，因为受URL长度最多是1024字节，但效率较高；	可以传输大量数据，所以上传文件时只能用Post方式；

不同点	GET	POST
支持编码	只能进行url编码	支持多种编码
字符格式	只能支持ASCII字符，向服务器传的中文字符可能会乱码。	支持标准字符集，可以正确传递中文字符。
浏览器处理	请求会被浏览器主动cache，请求参数会被完整保留在浏览器历史记录里	浏览器不会主动cache，参数不会被保留，除非手动设置
业务应用	常用在从服务器上获取数据	常用在向服务器发送数据

==**GET只需要跑一趟就把信息送到了，而POST得跑两趟。第一趟，先去和服务器打个招呼“嗨，我等下要送一消息来，你们打开门迎接我”，然后再回头把数据送过去。**==

因为POST需要两步，时间上消耗的要多一点。所以看起来GET比POST更有效,但事实上不是，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。而且并非所有浏览器都会在POST中发送两次包，Firefox浏览器就只发送一次。

返回目录

07.常见的HTTP相应状态码

初级 HTTP

状态码	描述
1xx	指示信息--表示请求已接收，继续处理
2xx	成功--表示请求已被成功接收、理解、接受
3xx	重定向--要完成请求必须进行更进一步的操作
4xx	客户端错误--请求有语法错误或请求无法实现
5xx	服务器端错误--服务器未能实现合法的请求

状态码	描述
200	请求被正常处理
204	请求被受理但没有资源可以返回
206	客户端只是请求资源的一部分，服务器只对请求的部分资源执行GET方法，相应报文中通过Content-Range指定范围的资源。
301	永久性重定向

状态码	描述
302	临时重定向
303	与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上
304	发送附带条件的请求时，条件不满足时返回，与重定向无关
307	临时重定向，与302类似，只是强制要求使用POST方法
400	请求报文语法有误，服务器无法识别
401	请求需要认证
403	请求的对应资源禁止被访问
404	服务器无法找到对应资源
500	服务器内部错误
503	服务器正忙

返回目录

08.如何对HTTP进行优化

中级 HTTP

1. TCP复用

TCP连接复用是将多个客户端的HTTP请求复用到一个服务器端的TCP连接上，HTTP复用是指一个客户端的多个HTTP请求通过一个TCP连接进行处理

2. 内容缓存

将经常用到的内容进行缓存到浏览器中，那么客户端就可以直接在内存中获取响应的数据

3. 压缩

将文本数据进行压缩，减少带宽

4. SSL加速

使用SSL协议对HTTP协议进行加密，在通道内加密并加速

返回目录

09.TCP与UDP的区别

初级 TCP UDP

UDP协议和TCP协议都是传输层协议。

TCP（Transmission ControlProtocol，传输控制协议）提供的是面向连接，可靠的字节流服务。客户和服务器交换数据前，必须现在双方之间建立一个TCP连接，之后才能传输数据。并提供超时重发、丢弃重复数据，检验数据，流量控制等功能，保证数据能完整地从一端传到另一端。

==简单说就是必须要建立连接后才能传输数据，确保传输完整性，类比现实当中的打电话。==

UDP（User Data Protocol，用户数据报协议）是一个简单的面向数据报的运输层协议。它不提供可靠性，只是把应用程序传给IP层的数据报发送出去，但是不能保证它们能到达目的地。由于UDP在传输数据报前不用再客户和服务器之间建立一个连接，且没有超时重发等机制，所以传输速度很快。

==简单说就是单向把程序中的信息发送了，但也不知道对方收到没有，类比现实当中的寄信。==

类型	不同点描述
TCP	面向连接，可靠的，速度慢，效率低。
UDP	无连接、不可靠、速度快、效率高。

返回目录

10.请介绍TCP的3次握手和4次挥手流程

初级 TCP

SYN:请求字段
ACK:应答字段

==建立双工通信，确保双方都能收到对方的信息，所以需要3次握手==
第1次握手：建立连接时，客户端发送syn包（syn=x）到服务器，并进入同步已发送状态，等待服务器确认；SYN：同步序列编号。

> 例如：客户端SYN=1; 客户端向服务器发送建立通信请求把客户端SYN=1的包发到服务器
> 客户端：我发了！

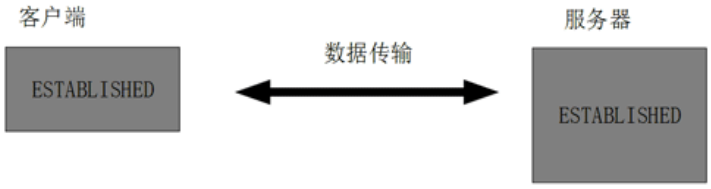
第2次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，此时服务器进入同步已接受状态；

> 例如：服务端ACK=客户端SYN+1=2,服务端SYN=10；意思是服务端已经收到客户端请求，并在客户端SYN请求值上加1作为ACK值返回给客户端，告知客户端自己已收到，同时发送服务端自己的SYN值给客户端。（注意此处会同时返回SYN和ACK包给客户端）
> 服务器：你发来吧，我收到你的SYN了！

第3次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=y+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

> 例如：客户端AC包=服务端SYN+1=11；客户端在收到服务器的SYN包，并在服务器SYN请求值上加1作为ACK返回给服务端，告知客服端已收到服务端的SYN包，完成第三次握手。
> 客户端：我收到你的SYN了！

数据传输.....



==全双工关闭需要客户端和服务器发送和接受都关闭，但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，只能先回复一个ACK报文，所以需要4次挥手==
第1次挥手：客户端进程发出连接释放报文，并且停止发送数据。此时，客户端进入FIN-WAIT-1（终止等待1）状态。

> 客户端：我不发了！停止发送，等待你的确认

第2次挥手：
服务器收到连接释放报文，发出确认报文。服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。

> 服务器：我知道了！进入等待程序关闭状态

客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）。

> 客户端：好，我知道你已经收到了！继续等待你的关闭确认

第3次挥手：服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最

后确认）状态，等待客户端的确认。

> 服务器：我不发了！关闭发送，等待你的确认

第4次挥手：
客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。

> 客户端：我已经收到你的确认了，关闭接收

服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

> 服务器：我已经收到你的确认了，关闭发送

返回目录

11.为什么TCP连接的时候是3次握手，关闭的时候却是4次握手？

初级 TCP

因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。==但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。== 只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

返回目录

12.TCP为什么会分包和粘包

高级 TCP

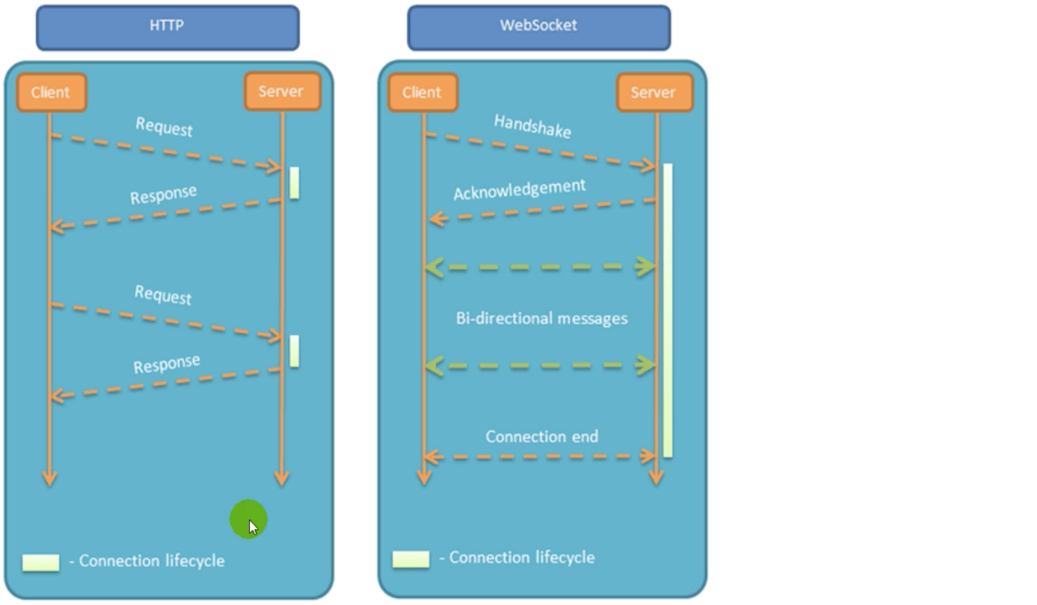
- TCP是以段（Segment）为单位发送数据的,建立TCP链接后,有一个最大消息长度（MSS）.如果应用层数据包超过MSS,就会把应用层数据包拆分,分成两个段来发送.这个时候接收端的应用层就要拼接这两个TCP包,才能正确处理数据.
- 有时候,TCP为了提高网络的利用率,会使用一个叫做Nagle的算法.该算法是指,发送端即使有要发送的数据,如果很少的话,会延迟发送.如果应用层给TCP传送数据很快的话,就会把两个应用层数据包"粘"在一起,TCP最后只发一个TCP数据包给接收端.

返回目录

13.HTTP和websocket的区别

中级 HTTP WEBSOCKET

WebSocket允许服务端主动向客户端推送数据。在WebSocket协议中，客户端浏览器和服务器只需要完成一次握手就可以创建持久性的连接，并在浏览器和服务器之间进行双向的数据传输。



- ==**最大的区别就是HTTP只能由客户端推送信息给被动的服务端，而websocket既可以让客户端发送消息给服务端，也可以让服务端主动推送消息到客户端，实现双工通信**==
- **http协议是用在应用层的协议，他是基于tcp协议的，http协议建立链接也必须要有三次握手才能发送信息。**

> http链接分为短链接，长链接，短链接是每次请求都要三次握手才能发送自己的信息。即每一个request对应一个response。长链接是在一定的期限内保持链接。保持TCP连接不断开。客户端与服务器通信，必须要由客户端发起然后服务器返回结果。客户端是主动的，服务器是被动的。

- **WebSocket是为解决客户端与服务端实时通信。浏览器和服务器只需要做1个握手的动作，在建立连接之后，双方可以在任意时刻，相互推送信息。同时，服务器与客户端之间交换的头信息很小。**

> 建立了WebSocket之后服务器不必在浏览器发送request请求之后才能发送信息到浏览器。这时的服务器已有主动权想什么时候发就可以发送信息到服务器。而且信息当中不必在带有head的部分信息了与http的长链接通信来说，这种方式，不仅能降低服务器的压力。而且信息当中也减少了部分多余的信息。

返回目录

14.HTTP的长连接与websocket的持久连接的区别

高级 HTTP WEBSOCKET

- **HTTP1.1的连接默认使用长连接**（persistent connection）

  HTTP 1.1默认进行持久连接。在1次 TCP 连接中可以完成多个 HTTP 请求，但是对每个请求仍然要单独发 header，Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Nginx）中设定这个时间。这种长连接是一种“伪链接”

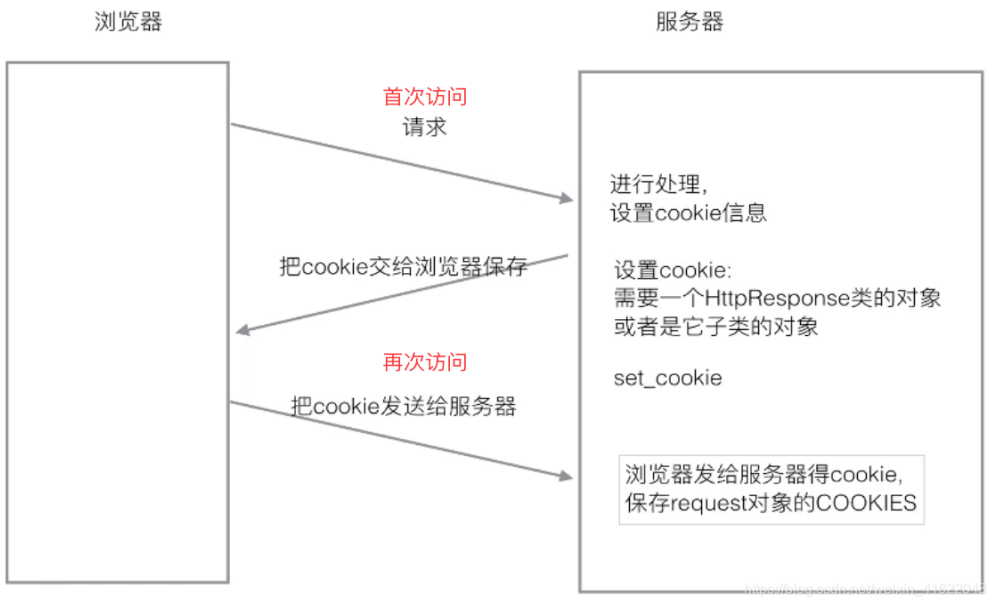
- **WebSocket的持久连接**

  WebSocket 是一个持久化的协议，只需建立1次Request/Response消息对，之后都是TCP连接，==避免了需要多次建立Request/Response消息对而产生的冗余头部信息。==

返回目录

15.什么是cookie？其特点和应用场景有哪些？其如何获取、设置？

中级 COOKIE PYTHON



- **cookie是一种会话跟踪技术，由服务器生成，然后通过响应发送给客户端的一个键值对。**
- 特点：

> 1. cookie是一个标准的python字典，以键值对方式进行存储，键值都是字符串
> 2. 通过浏览器访问一个网站时，浏览器会将存储该网站相关的所有cookie信息发送给该网站的服务器，request.COOKIES
> 3. cookie是基于域名安全的
> 4. cookie是有过期时间的，如果不指定，默认关闭浏览器后cookie就会过期
> 5. 单个Cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个Cookie。

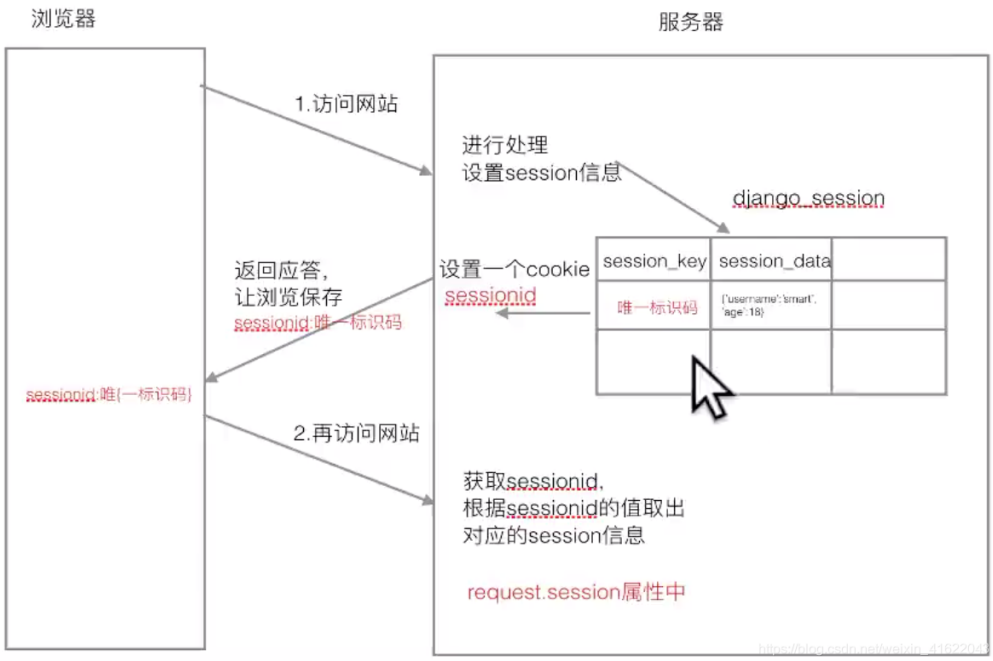
- **应用场景：记住用户名和密码，安全性要求不高**

```
response.set_cookie("is_login",True)  # 获取
request.COOKIES.get("is_login")  # 设置
```

返回目录

16.什么是session？其特点和应用场景有哪些？其如何设置、获取、清空？

中级 SESSION PYTHON



- **Session是服务器端技术，它保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。这就是 Session。客户端浏览器再次访问时只需要从该 Session 中查找该客户的状态就可以了。**
- 特点：

> 1. 它是以已键值对进行存储
> 2. 它依赖cookie，唯一的标识码session ID保存在cookie中
> 3. 它也是有过期时间，如果不指定，默认两周就会过期
> 4. 它是用base64加密

- 应用场景：涉及到安全性要求比较高的数据，银行卡账户、密码

```
request.session["is_login"] = True  # 设置
is_login = request.session.get("is_login")  # 获取
request.session.flush()  # 清空
```

返回目录

17.session和cookie的相同点和区别

中级 SESSION COOKIE

- **相同点**

	cookie/session
跟踪技术	都是踪浏览器用户身份的绘画方式
生成地点	都在服务器

	cookie/session
存储形式	都是键值对
过期时间	都可自定义

- **不同点**

	cookie	session
存放地点	浏览器	服务器
安全性	不安全，他人可通过分析存放在本地的cookie并进行Cookie欺骗	比较安全，因数据在服务器中，且用base64加密
依赖性	无	依赖cookie，唯一的标识码session ID保存在cookie中
过期时间	默认关闭浏览器后就会过期	默认两周就会过期

返回目录

18.什么是JWT，有什么特点

初级 JWT cookie session

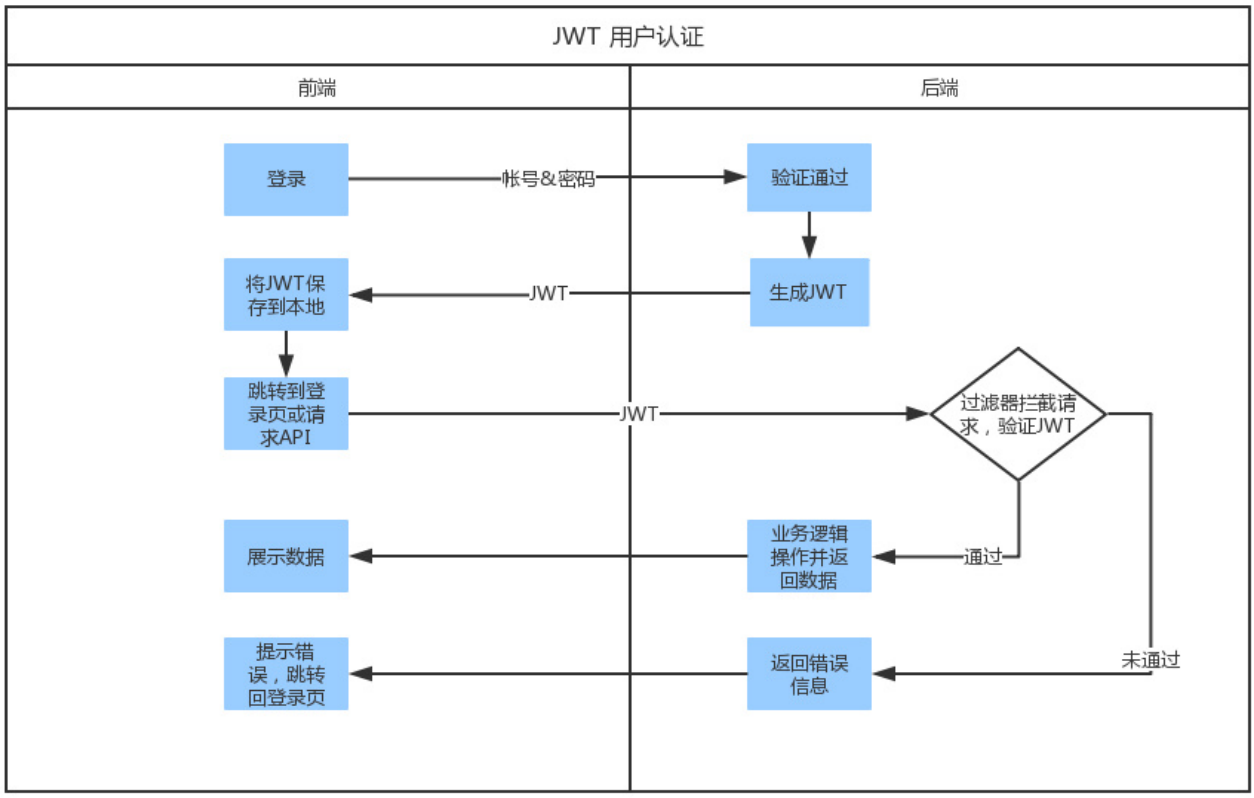
JWT 全称 JSON Web Tokens，它定义了一种以JSON 对象形式的安全通信方法。它由头部、负载和签名3部分组成。它具有2个特点：

- 简洁：可以通过URL, POST 参数或者在 HTTP header 发送，因为数据量小，传输速度快
- 自包含：负载中包含了所有用户所需要的信息，避免了多次查询数据库

返回目录

19.简述JWT的工作原理

初级 JWT cookie session



1. 客户端通过Web表单将正确的用户名和密码发送到服务端的接口。这一过程一般是POST请求。建议的方式是通过SSL加密的传输（https协议），从而避免敏感信息被嗅探。
2. 服务端核对用户名和密码成功后，将用户的id等其他信息作为JWT Payload（负载），将其与头部分别进行Base64编码拼接后签名，形成一个JWT。形成的JWT就是一个形同lll.zzz.xxx的字符串，并设置有效时间。
3. 服务端将JWT字符串作为登录成功的返回结果返回给客户端。
4. 客户端将返回的JWT以cookie的形式保存在浏览器上，并设置cookie的有效时间（建议客户端cookie和服务端JWT的有效时间设置为一致），用户登出时客户端需删除cookie。
5. 客户端在每次请求时将JWT放入HTTP Header中的Authorization位。(解决XSS和XSRF问题)
6. 服务端对收到的JWT进行解密和校验，如检查签名是否正确、Token是否过期、Token的接收方是否是自己等。
7. 验证通过后服务端使用JWT中包含的用户信息进行其他逻辑操作，返回相应结果，否则返回401。

返回目录

20.传统Session方式存储ID和JWT的区别

初级 JWT cookie session

Session是保存在服务端的，而JWT是保存在客户端的。

Session方式存储用户id的最大弊病在于Session是存储在服务器端的，所以==需要占用大量服务器内存，对于较大型应用而言可能还要保存许多的状态。== 一般而言，大型应用还需要借助一些KV数据库和缓存机制来实现Session的存储。

==JWT方式将用户状态分散到了客户端中，可以减少服务器查询数据库的次数和减轻服务端的内存压力。== 除了用户id之外，还可以存储其他的和用户相关的信息，例如该用户是否是管理员、用户所在的分组等。虽

说JWT方式让服务器有一些计算压力（例如加密、编码和解码），但是这些压力相比磁盘存储而言可能就不算什么了。具体是否采用，需要在不同场景下用数据说话。

返回目录

Django模块

框架层

01.什么是Django框架？

初级 Django

Django是一个开放源代码的Web应用框架，由Python写成。采用了MTV的框架模式。使用这种架构，程序员可以方便、快捷地创建高品质、易维护、数据库驱动的应用程序。它还包含许多功能强大的第三方插件，使得Django具有较强的可扩展性。

返回目录

02.Django对web开发有哪些优势

初级 Django

- **功能完善、要素齐全**：该有的、可以没有的都有，自带大量常用工具和框架，无须你自定义、组合、增删及修改。
- **完善的文档**：经过十多年的发展和完善，Django有广泛的实践案例和完善的在线文档。开发者遇到问题时可以搜索在线文档寻求解决方案。
- **强大的数据库访问组件**：Django的Model层自带数据库ORM组件，使得开发者无须学习其他数据库访问技术（SQL、pymysql、SQLALchemy等）。
- **灵活的URL映射**：Django使用正则表达式管理URL映射，灵活性高。新版的2.0，进一步提高了URL编写的优雅性。
- **丰富的Template模板语言**：类似jinjia模板语言，不但原生功能丰富，还可以自定义模板标签，并且与其ORM的用法非常相似。
- **自带后台管理系统admin**：只需要通过简单的几行配置和代码就可以实现一个完整的后台数据管理控制平台。
- **完整的错误信息提示**：在开发调试过程中如果出现运行错误或者异常，Django可以提供非常完整的错误信息帮助定位问题。

返回目录

03.简述Django项目的组成模块

初级 Django

1. **Project**
2. **Apps**

3. **Model**
4. **URL Route**
5. **View**
6. **DTL**
7. **Admin**
8. **Cache System**

以下详细参考：

- **工程**

> 工程是承载了Django实例的所有设置的Python程序包。大部分情况下，一个Web站点就是一个工程。工程内可以新建及存放该工程固有的应用，或者保存Web站点的设置(数据库设置、Django的选项设置、各应用的设置等)

- **应用**

> 对于Django而言，应用之的是表示单一工程的Web应用的Python程序包。由于其本质就是Python程序包，因此方法PYTHONPATH有效地任何位置都没有问题。这里最好尽量减少应用与工程、应用于应用之间的依赖关系，做到功能独立，以便在其他工程中重复利用。

- **模型**

> Django提供了O/R映射工具，因此可以用Python代码来描述数据库布局。 每个模型都是继承了django.db.models.Model类的Python的类，分别对应数据库中的一个表格。通过建数据库的字段、关系、行为定义为模型类的属性或方法，我们可以使用丰富且灵活的数据库方位API。

- **URL分配器**

> URL分配器机制使得URL信息不再受框架及扩展名的制约，从而让Web应用的URL设计保持简介。
> URI在URIconf模块中进行描述，URLconf模块中包含使用正则表达式书写的URL和Python函数的映像。URIconf能够以应用为单位进行分割，因此提高了应用的可重复利用性。另外，我们可以利用给URL设置名称并定义的方式让代码和目标直接通过该名称调用URL，从而将URL设计与代码分离。

- **视图**

> Django的视图时一类函数，它能够生成指定页面的HttpResponse对象或像Http 404这样的异常情况，返回HTTP请求。典型的视图函数的处理流程通常是从请求参数中获取数据，读取模型，热按后根据获取的数据渲染模板。

- **模板系统**

> 在Django的概念中，模板系统只负责显示，并不是编写逻辑代码的环境。因此Django的模板系统将设计与内容、代码分离开来，是一共功能强、扩展性高、对设计者很友好的模板语言。
> 模板基于文本而不是XML，因此它不但能生成XML和HTML，还能生成E-mail、JavaScript、CSV等任意文本格式。
> 另外，如果使用模板继承功能，子模板只需要将父模板中预留的空位填满即可。我们在编写模板时只需要描述各个模板独有的部分，因此可以省去重复冗余的编码过程。

- **管理界面**

> 大多Web应用在运行过程中，都需要一个专供拥有管理员权限的用户添加、编辑、删除数据的界面，但是实际制作这个界面并不容易。
>
> Django只需将已经完工的模型添加到管理站点，就能根据模型定义，动态地生成页面。为我们提供一个功能齐全的管理界面。

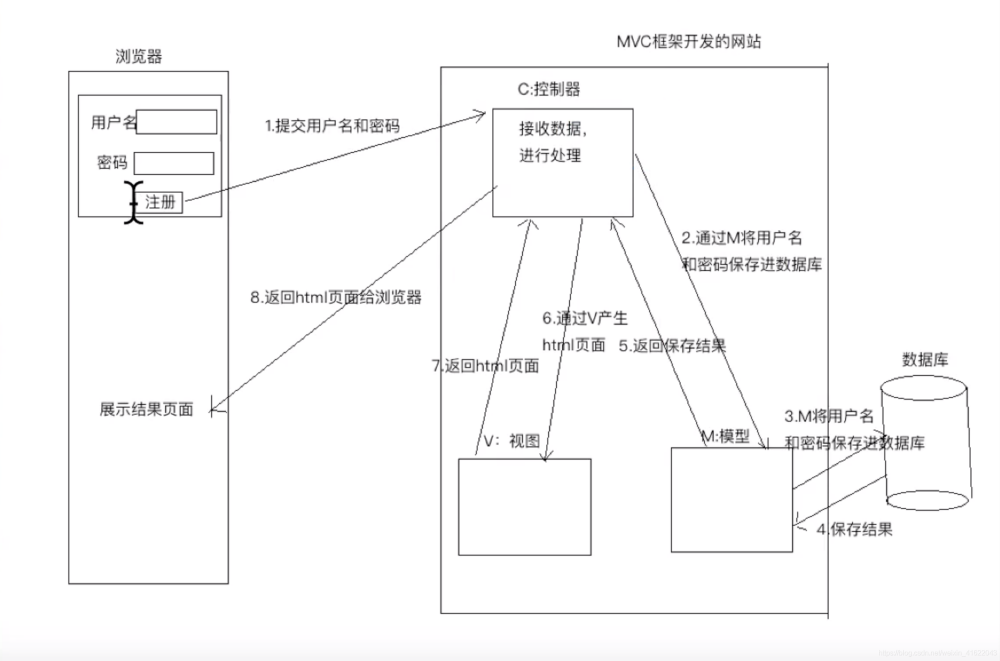
- **缓存系统**

> Django可以使用memcached等缓存后端轻松地缓存数据。比如可以将动态页面的渲染结果缓存下来，等到下次需要时直接读取缓存，从而不必每次都对动态页面进行处理。
>
> 缓存的后端可以从memcached、数据库、文件系统、本地内存等位置进行选择。缓存对象也支持整个网站、特定的整个视图、部分模板、特定数据等。

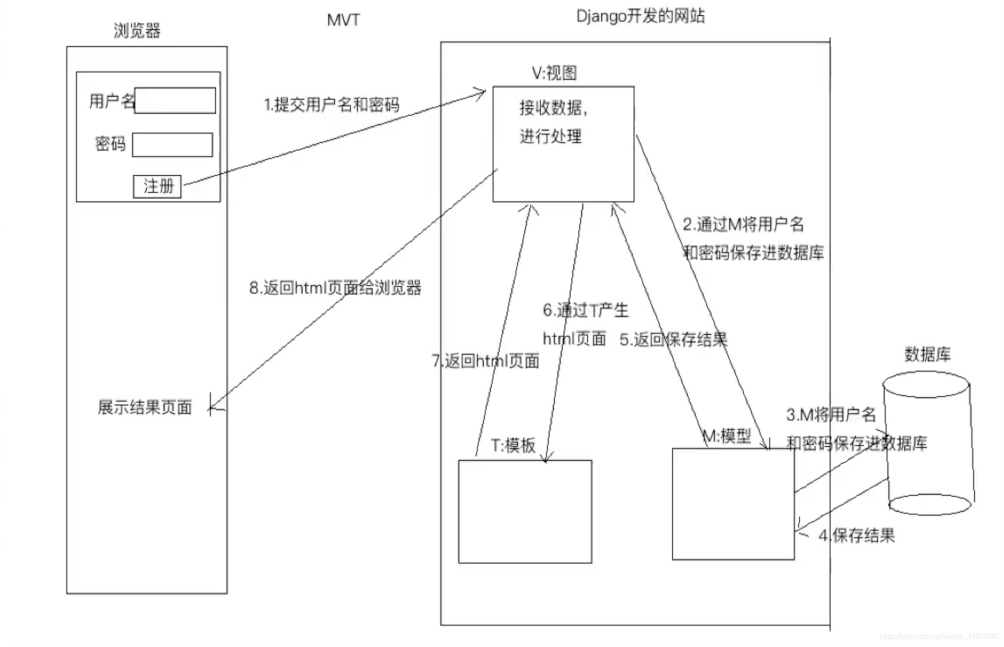
返回目录

04.简述MVC模式和MVT模式

初级 MVC MVT



MVC就是把Web应用分为模型(M)，控制器(C)和视图(V)三层,他们之间以一种插件式的、松耦合的方式连接在一起，模型负责业务对象与数据库的映射(ORM)，视图负责与用户的交互(页面)，控制器接受用户的输入调用模型和视图完成用户的请求。



MTV

Django的MTV模式本质上和MVC是一样的，也是为了各组件间保持松耦合关系，只是定义上有些许不同，Django的MTV分别是：

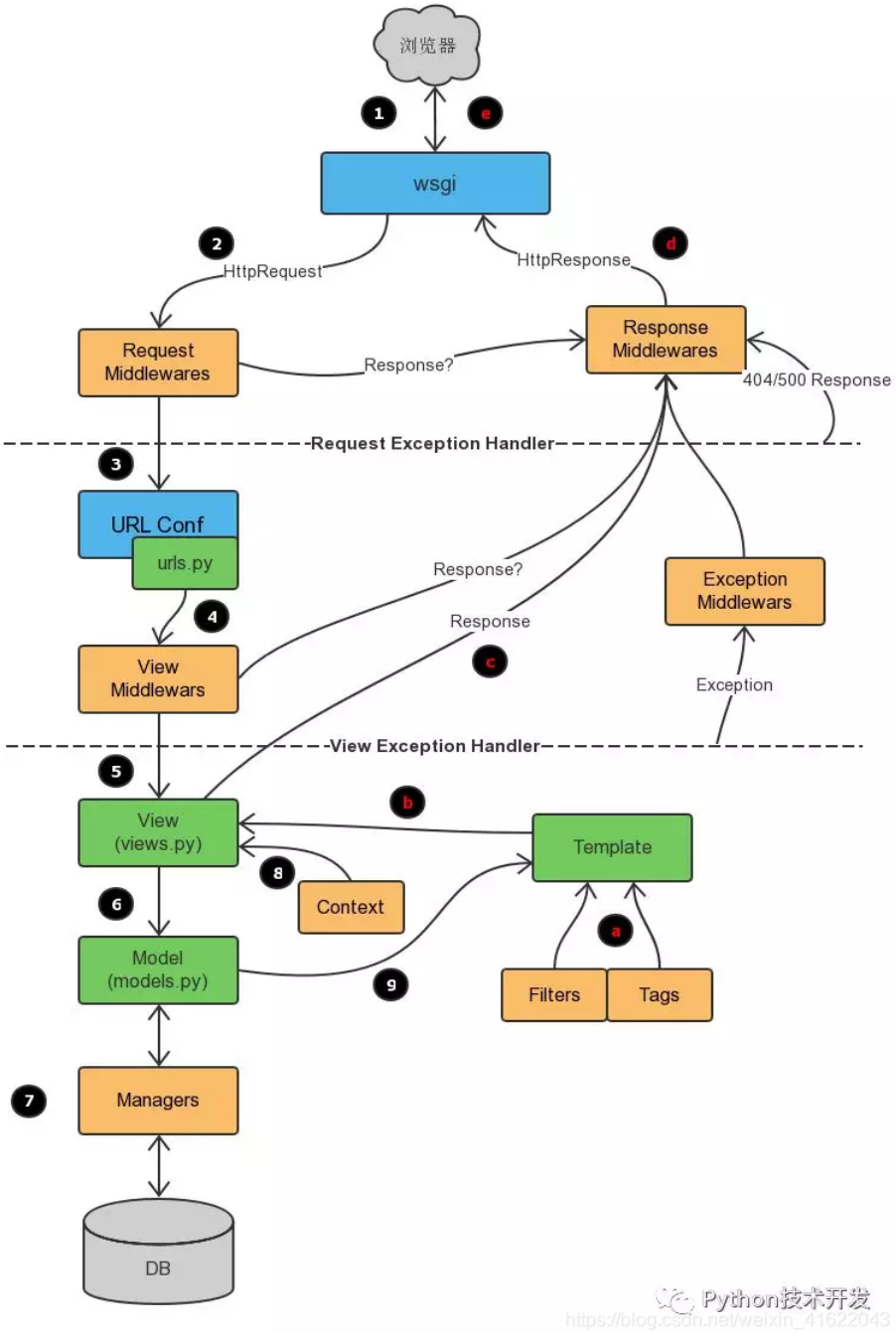
M 代表模型（Model）：负责业务对象和数据库的关系映射(ORM)。
T 代表模板 (Template)：负责如何把页面展示给用户(html)。
V 代表视图（View）：负责业务逻辑，并在适当时候调用Model和Template。

除了以上三层之外，还需要一个URL分发器，它的作用是将一个个URL的页面请求分发给不同的View处理，View再调用相应的Model和Template， MTV的响应模式。

返回目录

05.简述Django请求生命周期

初级 Django 生命周期



1. uWSGI服务器通过wsgi协议,将HttpRequest交给web框架 （Flask、Django）
2. 首先到达request中间件，对请求对象进行校验或添加数据，例如：csrf、request.session，如果验证不通过直接跳转到response中间件
3. 通过URL配置文件找到urls.py文件
4. 根据浏览器发送的URL，通过视图中间件去匹配不同的视图函数或视图类，如果没有找到相对应的视图函数，就直接跳转到response中间件
5. 在视图函数或视图类中进行业务逻辑处理，处理完返回到response中间件
6. 模型类通过ORM获取数据库数据，并返回序列化json或渲染好的Template到response中间件
7. 所有最后离开的响应都会到达response中间件，对响应的数据进行处理，返回HttpResponse给wsgi
8. wsgi经过uWSGI服务器,将响应的内容发送给浏览器。

返回目录

07.什么是WSGI

初级 wsgi Django

WSGI(Python Web Server Gateway Interface，即Web服务器网关接口)是Python定义的Web服服务器和Web应用程序或框架之间的一种简单而通用的接口。它是Python为了解决Web服务器端与客户端之间的通信问题而产生的，它基于现存的CGI标准而设计。其定义了Web服务器如何与Python应用程序进行交互，让Python写的Web应用程序可以和Web服务器对接起来。

返回目录

08.uwsgi、uWSGI和WSGI的区别

中级 uwsgi uWSGI WSGI

```
graph LR
Nginx-- uwsgi ---uWSGI
uWSGI-- WSGI ---Django
```

- uwsgi：是uWSGI服务器实现的独有协议，用于Nginx服务与uWSGI服务的通信规范
- uWSGI：是一个Web服务器，它实现了WSGI/uwsgi/HTTP等协议，用于接收Nginx转发的动态请求，处理后发个python应用程序
- WSGI：用在python web框架（Django/Flask）编写的应用程序与web服务器之间的规范

返回目录

09.Django的HttpRequest对象是在什么时候创建的？

中级 WSGI

```
class WSGIHandler(base.BaseHandler):
    request = self.request_class(environ)
```

请求走到WSGIHandler类的时候，执行cell方法，将environ封装成了request

返回目录

10.什么是中间件并简述其作用

初级 中间件 Django

中间件是一个用来处理Django请求和响应的框架级钩子。它是一个轻量、低级别的插件系统，用于在全局范围内改变Django的输入和输出。每个中间件组件都负责做一些特定的功能。

返回目录

11.列举django中间件的5个方法，以及django中间件的应用场景

初级 中间件 Django

- process_request : 请求进来时,权限认证
- process_view : 路由匹配之后,能够得到视图函数
- process_exception : 异常时执行
- process_template_responseprocess : 模板渲染时执行
- process_response : 请求有响应时执行

返回目录

12.简述Django对http请求的执行流程

初级 Django

在接受一个Http请求之前的准备,需启动一个支持WSGI网关协议的服务器监听端口等待外界的Http请求，比如Django自带的开发者服务器或者uWSGI服务器。 Django服务器根据WSGI协议指定相应的Handler来处理Http请求。此时服务器已处于监听状态，可以接受外界的Http请求,当一个http请求到达服务器的时候,Django服务器根据WSGI协议从Http请求中提取出必要的参数组成一个字典（environ）并传入Handler中进行处理。在Handler中对已经符合WSGI协议标准规定的http请求进行分析，比如加载Django提供的中间件，路由分配，调用路由匹配的视图等。最后返回一个可以被浏览器解析的符合Http协议的HttpResponse。

返回目录

13.Django中session的运行机制是什么

初级 Django

django的session存储可以利用中间件来实现。需要在 `settings.py` 文件中注册APP、设置中间件用于启动。设置存储模式（数据库/缓存/混合存储）和配置数据库缓存用于存储，生成django_session表单用于读写。

返回目录

14. 什么是CSRF，请描述其攻击原理，在Django中如何解决

初级 Django

CSRF（cross-site request forgery）简称跨站请求伪造。例如，你访问了信任网站A,然后网站A会用保存你的个人信息并返回给你的浏览器一个cookie，然后呢，在cookie的过期时间之内，你去访问了恶意网站B，它给你返回一些恶意请求代码，要求你去访问网站A，而你的浏览器在收到这个恶意请求之后，在你不知情的情况下，会带上保存在本地浏览器的cookie信息去访问网站A，然后网站A误以为是用户本身的操作，导致来自恶意网站C的攻击代码会被执行：发邮件，发消息，修改你的密码，购物，转账，偷窥你的个人信息，导致私人信

息泄漏和账户财产安全受到威胁。

在post请求时，form表单或ajax里添加csrf_token，服务端开启CSRF中间件进行验证。

解决原理是页面添加csrf_token值后，用户通过URL访问（GET请求）该页面时，Django会在响应中自动帮我们生成cookie信息，返回给浏览器，同时在前端代码会生成一个csrf_token值，然后当你POST提交信息时，Django会自动比对cookie里和前端form表单或ajax提交上来的csrf_token值，两者一致，说明是当前浏览器发起的正常请求并处理业务逻辑返回响应，那么第三方网站拿到你的cookie值为什么不能通过验证呢，因为他没你前端的那个随机生成的token值，他总不能跑到你电脑面前查看你的浏览器前端页面自动随机生成的token值吧。

> 注意：你打开浏览器访问某个url（页面），默认是get请求，也就是说，你只要访问了url，对应的视图函数里只要不是if xx == post的逻辑就会执行，所以你打开页面，他会先生成cookie（token）值，返回给浏览器，然后你提交表单，或者发ajax请求时，会将浏览器的cookie信息（token值）发送给服务器进行token比对，这个过程相对于你发起了两次请求，第一次是get，第二次才是post，搞清楚这个，你才能明白csrf_token是怎么比对的。

返回目录

15. Django中CSRF的实现机制

初级 Django

1. django第1次响应来自某个客户端的请求时,服务器随机产生1个token值，把这个token保存在session中;同时服务器把这个token放到cookie中交给前端页面；
2. 该客户端再次发起请求时，把这个token值加入到请求数据或者头信息中,一起传给服务器；
3. 服务器校验前端请求带过来的token和session里的token是否一致。

返回目录

16.什么是跨域请求，其有哪些方式

初级 Django

- **跨域是指一个域下的文档或脚本试图去请求另一个域下的资源。**

方式如下：

1. 资源跳转： a链接、重定向、表单提交
2. 资源嵌入： link/script/img/frame/dom等标签，还有样式中background:url()、@font-face()等文件外链
3. 脚本请求： js发起的ajax请求、dom和js对象的跨域操作等

返回目录

17.跨域请求Django是如何处理的

初级 Django

使用第三方工具 **django-cors-headers** 即可彻底解决

- 注册app
- 添加中间件
- 配置运行跨域请求方法

返回目录

16.什么是信号量

初级 Django

Django包含一个"信号调度程序"，它有助于在框架中的其他位置发生操作时通知分离的应用程序。简而言之，信号允许某些发送者通知一组接收器已经发生了某些动作。当许多代码可能对同一事件感兴趣时，它们特别有用.

返回目录

17.web框架的本质是什么

初级 Django

- web框架本质是一个socket服务端，用户的浏览器是一个socket客户端。

返回目录

18.谈谈你对restful规范的认识

初级 Django

restful是一种软件架构设计风格，并不是标准，它只是提供了一组设计原则和约束条件，主要用于客户端和服务器交互类的软件。 就像设计模式一样，并不是一定要遵循这些原则，而是基于这个风格设计的软件可以更简洁，更有层次，我们可以根据开发的实际情况，做相应的改变。

它里面提到了一些规范，例如

1. restful 提倡面向资源编程,在url接口中尽量要使用名词，不要使用动词
2. 在url接口中推荐使用Https协议，让网络接口更加安全
3. 可以根据Http不同的method，进行不同的资源操作
4. 在url中可以体现版本号
5. url中可以体现是否是API接口
6. url中可以添加条件去筛选匹配
7. 响应式应该设置状态码
8. 有返回值，而且格式为统一的json格式
9. 返回错误信息
10. 返回结果中要提供帮助链接，即API最好做到Hypermedia

返回目录

19.Django中如何加载初始化数据

初级 Django

Django在创建对象时在调用save()方法后，ORM框架会把对象的属性写入到数据库中，实现对数据库的初始化；通过操作对象，查询数据库，将查询集返回给视图函数，通过模板语言展现在前端页面。

返回目录

20.Django缓存系统类型有哪些

初级 Django

1. 全站缓存，较少使用

```
MIDDLEWARE_CLASSES = (
    ‘django.middleware.cache.UpdateCacheMiddleware’,  # 第一
    'django.middleware.common.CommonMiddleware',
    ‘django.middleware.cache.FetchFromCacheMiddleware’,  # 最后
)
```

2. 视图缓存，用户视图函数或视图类中

```
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())
```

3. 模板缓存，指缓存不经常变换的模板片段

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

{% cache 2 'name' %} # 存的key
    <h3>缓存:-----:{{ t }}</h3>
{% endcache %}
```

返回目录

21.请简述Django下的（内建）缓存机制

初级 Django

Django根据设置的缓存方式，浏览器第一次请求时，cache会缓存单个变量或整个网页等内容到硬盘或者内存中，同时设置response头部，当浏览器再次发起请求时，附带f-Modified-Since请求时间到Django，Django发现f-Modified-Since会先去参数之后，会与缓存中的过期时间相比较，如果缓存时间比较新，则会重新请求数据，并缓存起来然后返回response给客户端，如果缓存没有过期，则直接从缓存中提取数据，返回给response给客户端。

返回目录

22.什么是ASGI，简述WSGI和ASGI的关系与区别

初级 Django

ASGI是异步网关协议接口，一个介于网络协议服务和Python应用之间的标准接口，能够处理多种通用的协议类型，包括HTTP，HTTP2和WebSocket。

WSGI是基于HTTP协议模式的，不支持WebSocket，而ASGI的诞生则是为了解决Python常用的WSGI不支持当前Web开发中的一些新的协议标准。同时，ASGI对于WSGI原有的模式的支持和WebSocket的扩展，即ASGI是WSGI的扩展。

返回目录

23.Django如何实现websocket

初级 Django

django实现websocket使用channels。==channels通过http协议升级到websocket协议，保证实时通讯。==也就是说，我们完全可以用channels实现我们的即时通讯。而不是使用长轮询和计时器方式来保证伪实时通讯。==他使用asgi协议而不是wsgi协议，他通过改造django框架，使django既支持http协议又支持websocket协议。==

返回目录

25.列举django的核心组件

初级 Django

- 用于创建模型的对象关系映射；
- 为最终用户设计较好的管理界面；
- URL设计；
- 设计者友好的模板语言；
- 缓存系统。

返回目录

26.Django本身提供了runserver，为什么不能用来部署

初级 Django

1. runserver方法是调试 Django 时经常用到的运行方式，它使用Django自带的 WSGI Server 运行，==主要在测试和开发中使用，并且 runserver 开启的方式也是单进程。==
2. uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http 等协议。注意uwsgi是一种通信协议，而uWSGI是实现uwsgi协议和WSGI协议的 Web 服务器。==uWSGI具有超快的性能、低内存占用和多app管理等优点，并且搭配着Nginx就是一个生产环境了，能够将用户访问请求与应用 app 隔离开，实现真正的部署 。== 相比来讲，支持的并发量更高，方便管理多进程，发挥多核的优势，提升性能。

返回目录

返回目录

27.ajax请求的csrf解决方法

高级 Django

1. 首先在你需要发起ajax post请求的页面的里面随便一个地方加上 {% crsr_token %}
2. 在发起ajax post 请求时，组织json参数时，将浏览器cookie中的值赋予加入json中，键名为'csrfmiddlewaretoken'

返回目录

路由层

01.路由优先匹配原则是什么

初级 路由 Django

在url匹配列表中==位置优先匹配==

- 如第1条和第2条同时满足匹配规则，则优先匹配第1条。
- 如第1条为正则模糊匹配，第2条为精确匹配，也是优先匹配第1条

返回目录

02.urlpatterns中的name与namespace的区别

初级 Django

name:给路由起一个别名
namespace:防止多个应用之间的路由重复

返回目录

03.Django路由系统中include是干嘛用的？

初级 Django

include用作路由转发，通常，我们会在每个app里，各自创建一个urls.py路由模块，然后从根路由出发，将app所属的url请求，全部转发到相应的urls.py模块中。

返回目录

04.Django2.x中的path与django1.x里面的URL有什么区别

初级 Django

path与url是两个不同的模块,效果都是响应返回页面, path调用的是python第三方模块或框架,而url则是自定义的模块。url默认支持正则表达式，而path不支持，正则表达式需要使用另外一个函数re_path。

```
# django 2.x
django.urls path
# django 1.x
django.conf.urls url
```

答案来源

返回目录

05.Django重定向的几种方法，用的什么状态码

初级 Django

- HttpResponse
- Redirect
- Reverse
- 状态码：302,301

返回目录

模型层

01.命令migrate 和makemigrations的差别

初级 Django

- makemigrations:生成迁移文件
- migrate:执行迁移

返回目录

02.Django的Model的继承有几种形式

初级 Django

1. 用父类model来保存在子类model中重复的信息。父类model是不使用的也就是不生成单独的数据表，这种情况下使用抽象基类继承 Abstract base classes。
2. 如从现有的model继承并让每个model都有自己的数据表，那么使用多重表继承 Multi-table inheritance。
3. 如只在 model 中修改 Python 级的行为，而不涉及字段改变。 代理 model (Proxy models) 适用于这种场合

返回目录

03.class Meta中的元信息字段有哪些

初级 Django

1. Model类可以通过元信息类设置索引和排序信息
2. 元信息是在Model类中定义一个Meta子类

```
class Meta:
    # ---常用
    db_table = 'table_name'  # 自定义表名
    index_together = ('tag1', 'tag2')  # 联合索引
    unique_together = ('tag3', 'tag4')  # 联合唯一索引
    verbose_name = 'table_name'  # /admin/中显示的表名称
    verbose_name_plural = verbose_name
    ordering = 'ordering_tag'  # 排序字段#这个选项是指定，模型的复数形式是什么
    abstract =True  # 抽象基类

    # ---非常用
    # app_label这个选项只在一种情况下使用，就是你的模型类不在默认的应用程序包下的
models.py文件中，
    # 这时候你需要指定你这个模型类是那个应用程序的。
    # 比如你在其他地方写了一个模型类，而这个模型类是属于myapp的
    app_label='myapp'

    # db_table 是用于指定自定义数据库表名的。
    db_table='my_owner_table'

    # 有些数据库有数据库表空间，比如Oracle。你可以通过db_tablespace来
    # 指定这个模型对应的数据库表放在哪个数据库表空间。
    db_tablespace

    # 由于Django的管理方法中有个lastest()方法，就是得到最近一行记录。
    # 如果你的数据模型中有 DateField 或 DateTimeField 类型的字段，
```

```
    # 你可以通过这个选项来指定lastest()是按照哪个字段进行选取的
    get_latest_by = "order_date"

    # 由于Django会自动根据模型类生成映射的数据库表，如果你不希望Django这么做，可以把
managed的值设置
    # 为False。默认值为True,这个选项为True时Django可以对数据库表进行 migrate或
migrations、
    # 删除等操作。在这个时间Django将管理数据库中表的生命周期如果为False的时候，不会对数
据库表进行
    # 创建、删除等操作。可以用于现有表、数据库视图等，其他操作是一样的。
    managed

    # permissions主要是为了在Django Admin管理模块下使用的，如果你设置了这个属性可以让
    # 指定的方法权限描述更清晰可读。要创建一个对象所需要的额外的权限. 如果一个对象有
admin 设置,
    # 则每个对象的添加,删除和改变权限会人(依据该选项)自动创建.
    # 下面这个例子指定了一个附加权限: can_deliver_pizzas:
    permissions = (("can_deliver_pizzas", "Can deliver pizzas"),)

    # 这个选项一般用于多对多的关系中，它指向一个关联对象。就是说关联对象找到这个对象后它
是经过排序的。
    # 指定这个属性后你会得到一个get_XXX_order()和set_XXX_order（）的方法,
    # 通过它们你可以设置或者回去排序的对象。
    order_with_respect_to = 'pizza'
```

返回目录

04.Django模型类关系有哪几种

初级 Django

- 一对一关系：OneToOneField
- 一对多关系：ForeignKey
- 多对多关系：ManyToManyField

返回目录

05.外键有什么用，什么时候合适使用外接，外键一定需要索引吗？

中级 Django

- 程序很难100%保证数据的完整性,而用外键即使在数据库服务器宕机或异常的时候,也能够最大限度的保证数据的一致性和完整性。
- 如果项目性能要求不高,安全要求高,建议使用外键，如果项目性能要求高,安全自己控制，不用外键，因为外键查询比较慢。
- 加入外键的主要问题就是影响性能,因此加入索引能加快关联查询的速度。

返回目录

06.Primary Key和Unique Key的区别

中级 Django

- Primary key与Unique Key都是唯一性约束。
- Primary key是主键，一个表只能由一个，Unique key是唯一键，一个表可以有多个唯一键字段。
- Primary key 必须不能为空，Unique Key 可为空。

返回目录

07.DateTimeField类型中的auto_now与auto_now_add有什么区别

初级 Django

- DateTimeField.auto_now ==用于记录更新时间==

> 这个参数的默认值为false，设置为true时，能够在保存该字段时，将其值设置为当前时间，并且每次修改model，都会自动更新。因此这个参数在需要存储“最后修改时间”的场景下，十分方便。需要注意的是，设置该参数为true时，并不简单地意味着字段的默认值为当前时间，而是指字段会被“强制”更新到当前时间，你无法程序中手动为字段赋值；如果使用django再带的admin管理器，那么该字段在admin中是只读的。

- DateTimeField.auto_now_add ==用于记录创建时间==

> 这个参数的默认值也为False，设置为True时，会在model对象第一次被创建时，将字段的值设置为创建时的时间，以后修改对象时，字段的值不会再更新。该属性通常被用在存储“创建时间”的场景下。与auto_now类似，auto_now_add也具有强制性，一旦被设置为True，就无法在程序中手动为字段赋值，在admin中字段也会成为只读的。

返回目录

08.当删除一个外键的时候，其关联的表有几种处理方式

初级 Django

有6种处理方式：

1. 同时删除父表和子表
2. 阻止删除父表
3. 子表设置为空
4. 子表设置为默认值
5. 子表什么都不做
6. 设置为一个传递给SET()的值或者一个回调函数的返回值

> **CASCADE**：代表删除联级，父表（少类表）被删除的记录在子表（多类表）中所有字段也会被对应删除，模拟SQL语言中的ON DELETE CASCADE约束，将定义有外键的模型对象同时删除！（该操作为当前Django版本的默认操作！）

> **PROTECT**：阻止上面的删除操作，但是弹出ProtectedError异常

> **SET_NULL**：代表父表（少类表）被删除后子表（多类表）对应的外键字段会设置为null，只有当字段设置了null=True，blank=True时，方可使用该值。

> **SET_DEFAULT**:代表父表（少类表）被删除后子表（多类表）对应的外键字段会设置为默认值。只有当字段设置了default参数时，方可使用。

> **DO_NOTHING**：什么也不做，一切看数据库级别的约束

> **SET()**：设置为一个传递给SET()的值或者一个回调函数的返回值。注意大小写，用得很少。

返回目录

09.如何通过外键，子表查询父表和父表查询子表

中级 Django

父表和子表关系如下：

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=64)
    age = models.IntegerField()
    tel = models.CharField(max_length=64)

    @property
    def all_cars(self):
        '''返回全部信息'''
        return self.cars.all()

    @property
    def info(self):
        '''返回部分信息'''
        return '%s %s' % (self.name, self.tel)

class Car(models.Model):
    owner = models.Foreignkey(Person, related_name='cars')
    name = models.CharField(max_length=64)
    price = models.FloatField()
```

子表查询父表

```
car = Car.objects.get(id=1)
# 查询该车的车主
owner = car.owner
```

父表查询子表

```
Tom = Person.objects.get(id=1)
# 查询此人有多少车
# 方式一:
# Django默认每个主表对象都有一个外键的属性，可以通过它来查询所有属于主表的子表信息
# 查询方式：主表.子表_set()，返回值为一个queryset对象
Tom.Car_set().all()
# 方式二：
# 通过在外键中设置related_name属性值既可
Tom.cars.all()
# 方式三：
# 通过@property装饰器在model中预定义方法实现
Tom.all_cars
```

返回目录

10.谈谈 GenericForeignkey 和 GenericRelation

高级 Django

GenericForeignkey 和 GenericRelation 的方法能够解决多外键的表单产生的大量沉余数据。通过 ContentType的查询，起到一个自动一对多的作用，能和任何模型都能连接起来，保证了代码的干净。避免了创建大量无用的空数据，有效减少存储空间和服务器压力。

返回目录

11.django中怎么写原生SQL

中级 Django SQL

1. 使用extra

```
# 查询人民邮电出版社出版并且价格大于50元的书籍
Book.objects.filter(publisher__name='人民邮电出版社').extra(where=['price>50'])
```

2. 使用raw

```
books=Book.objects.raw('select * from hello_book')

for book in books:
    print book
```

3. 使用游标

```
from django.db import connection
cursor = connection.cursor()
cursor.execute("insert into hello_author(name) values ('特朗普')")
cursor.execute("update hello_author set name='普京' WHERE name='特朗普'")
cursor.execute("delete from hello_author where name='普京'")
cursor.execute("select * from hello_author")
cursor.fetchone()
cursor.fetchall()
```

返回目录

12.谈一谈你对ORM的理解

中级 ORM

ORM是"对象-关系-映射"的简称。

ORM是MVC或者MVC框架中包括一个重要的部分，它实现了数据模型与数据库的解耦，即数据模型的设计不需要依赖于特定的数据库，通过简单的配置就可以轻松更换数据库，这极大的减轻了开发人员的工作量，不需要面对因数据库变更而导致的无效劳动。

13.如何使用Django ORM批量创建数据

中级 Django ORM

可以使用`django.db.models.query.QuerySet.bulk_create()`批量创建对象，减少SQL查询次数。
例子如下：

```
# 创建一个空列表
querysetlist=[]

# 把创建的对象添加入列表中
for i in resultlist:
    querysetlist.append(Account(name=i))

# 批量创建
Account.objects.bulk_create(querysetlist)
```

返回目录

14.列举django ORM中操作QuerySet对象的方法(至少5个)

初级 Django ORM

方法	作用

方法	作用
all()	查询所有结果
filter()	过滤查询对象。获取不到返回None。
get()	返回与所给筛选条件相匹配的对象，返回结果有且只有1个。如果符合筛选条件的对象超过1个或者没有都会抛出错误。
exclude()	排除满足条件的对象
order_by()	对查询结果排序
reverse()	对查询结果反向排序
count()	返回数据库中匹配查询(QuerySet)的对象数量。
first()	返回第一条记录
last()	返回最后一条记录
exists()	如果QuerySet包含数据，就返回True，否则返回False
values()	返回包含对象具体值的字典的QuerySet
values_list()	与values()类似，只是返回的是元组而不是字典。
distinct()	对查询集去重

返回目录

15.ORM如何取消级联

初级 Django

```
user = models.ForeignKey(User,blank=True,null=True,on_delete=models.SET_NULL)
```

在父表被删除，null为True的时候就会取消级联

返回目录

16.查询集的2大特性？什么是惰性执行

中级 Django

特性：

1. 惰性执行
2. 缓存

惰性执行是指创建查询集不会访问数据库，直到调用数据时，才会访问数据库。

返回目录

17.查询集返回的列表过滤器有哪些

中级 Django

all()：返回所有数据 filter()：返回满足条件的数据 exclude()：返回满足条件之外的数据，相当于sql语句中where部分的not关键字 order_by()：排序

返回目录

18.selected_related与prefetch_related有什么区别

高级 Django

在Django中当创建一个查询集的时候，并没有跟数据库发生任何交互。因此我们可以对查询集进行级联的filter等操作，只有在访问Queryset的内容的时候，Django才会真正进行数据库的访问。而多频率、复杂的数据库查询往往是性能问题最大的根源。

不过我们实际开发中，往往需要访问到外键对象的其他属性。如果按照默认的查询方式去遍历取值，那么会造成多次的数据库查询，效率可想而知。

在查询对象集合的时候，把指定的外键对象也一并完整查询加载，避免后续的重复查询。使用select_related() 和 prefetch_related() 可以很好的减少数据库请求的次数，从而提高性能。

1. **select_related**适用于一对一字段（OneToOneField）和外键字段（ForeignKey）查询；
2. **prefetch_related**适用多对多字段（ManyToManyField）和一对多字段的查询。（或许你会有疑问，没有一个叫OneToManyField的东西啊。实际上， ForeignKey就是一个多对一的字段，而被ForeignKey关联的字段就是一对多字段了）

返回目录

19.values()与values_list()有什么区别

初级 Django

- values : 读取字典的Queryset
- values_list : 读取元组的Queryset

返回目录

20.QueryDict和dict区别

高级 Django

在HttpRequest对象中, GET和POST属性是django.http.QueryDict类的实例。QueryDict类似字典的自定义

类，用来处理单键对应多值的情况。在 HttpRequest 对象中,属性 GET 和 POST 得到的都是 django.http.QueryDict 所创建的实例。这是一个django 自定义的类似字典的类，用来处理同一个键带多个值的情况。在 python 原始的字典中，当一个键出现多个值的时候会发生冲突，只保留最后一个值。而在 HTML 表单中，通常会发生一个键有多个值的情况，例如，多选框就是一个很常见情况。request.POST 和request.GET 的QueryDict 在一个正常的请求/响应循环中是不可变的。若要获得可变的版本，需要使用.copy()方法。

- python dict当一个键出现多个值的时候会发生冲突，只保留最后一个值。
- QueryDict是类似字典的自定义类，用来处理单键对应多值的情况。

返回目录

21.Django中查询Q和F的区别

中级 Django ORM

- Q查询：对数据的多个字段联合查询（常和且或非"&|~"进行联合使用）
- F查询：对数据的不同字段进行比较（常用于比较和更新，对数据进行加减操作）

返回目录

视图层

01.简述什么是FBV和CBV

初级 Django

FBV（function base views）使用视图函数处理请求
CBV（class base views）使用视图类处理请求

返回目录

02.如何给CBV的程序添加装饰器

初级 Django

```
from django.utils.decorators import method_decorator


@method_decorator(check_login)
def post(self, request):
    '''给方法加'''
    ...

@method_decorator(check_login)
def dispatch(self, request, *args, **kwargs):
    '''给dispatch加'''
    ...
```

```
@method_decorator(check_login, name="get")
@method_decorator(check_login, name="post")
class HomeView(View):
    '''给类加'''
    ...
```

导入 method_decorator 装饰器

1. 给方法加
2. 给dispatch加
3. 给类加

返回目录

03.常用视图响应的方式有哪些

初级 Django

- 常用视图响应的方式有4种方式**redirect**、**Response**、**HttpResponse**和**JsonResponse**

```
return Response({content=响应体, content_type=响应体数据类型, status=状态码)
return HttpResponse(content=响应体, content_type=响应体数据类型, status=状态码)
return JsonResponse({‘city’: ‘beijing’, ‘subject’:
‘python’},status=response.status_code)
return redirect(‘/index.html’)
```

返回目录

04.在视图函数中，常用的验证装饰器有哪些？

中级 Django

装饰器	用途
@login_required()	检查用户是否通过身份验证
@group_required()	检查用户是否属于有权限的用户组访问
@anonymous_required()	检验用户是否已经登录
@superuser_only()	它只允许超级用户才能访问视图
@ajax_required	用于检查请求是否是AJAX请求
@timeit	用于改进某个视图的响应时间，或者只想知道运行需要多长时间

详情
返回目录

05.视图函数和视图类的区别？

高级 Django

	优点	缺点
视图函数	容易实现跟理解；流程简单；直接使用装饰器	代码难以重用；处理HTTP请求时要有分支表达式
视图类	易拓展跟代码重用；可以用混合类继承；单独用类方法处理HTTP请求；有许多内置的通用视图函数	不容易去理解；代码流程负载；父类混合类中隐藏较多代码；使用装饰器时需要额外的导入或覆盖方法

返回目录

高阶

01.Django如何实现高并发？

高级 Django

1. Nginx + uWSGI + Django
2. Nginx + gunicorn + gevent + Django

详情
返回目录

02.如何提高Django应用程序的性能？

高级 Django

前端优化：

1. 减少 http 请求，减少数据库的访问量，比如使用雪碧图。
2. 使用浏览器缓存，将一些常用的 css，js，logo 图标，这些静态资源缓存到本地浏览器，通过设置 http 头中的 cache-control 和 expires 的属性，可设定浏览器缓存，缓存时间可以自定义。
3. 对 html，css，javascript 文件进行压缩，减少网络的通信量。

后端优化：

1. 合理的使用缓存技术，对一些常用到的动态数据，比如首页做一个缓存，或者某些常用的数据做个缓存，设置一定得过期时间，这样减少了对数据库的压力，提升网站性能。
2. 使用 celery 消息队列，将耗时的操作扔到队列里，让 worker 去监听队列里的任务，实现异步操作，比如发邮件，发短信。

3. 就是代码上的一些优化，补充：nginx 部署项目也是项目优化，可以配置合适的配置参数，提升效率，增加并发量。
4. 如果太多考虑安全因素，服务器磁盘用固态硬盘读写，远远大于机械硬盘，这个技术现在没有普及，主要是固态硬盘技术上还不是完全成熟， 相信以后会大量普及。
5. 服务器横向扩展。

返回目录

03.Django中当用户登录到A服务器进入登陆状态，下次被nginx代理到B服务器会出现什么影响

高级 Django

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

返回目录

04.谈谈对Celery的理解

高级 Django

Celery是由Python开发、简单、灵活、可靠的分布式任务队列，==其本质是生产者消费者模型，生产者发送任务到消息队列，消费者负责处理任务。== Celery侧重于实时操作，但对调度支持也很好，其每天可以处理数以百万计的任务。

特点：
简单：熟悉celery的工作流程后，配置使用简单
高可用：当任务执行失败或执行过程中发生连接中断，celery会自动尝试重新执行任务
快速：一个单进程的celery每分钟可处理上百万个任务
灵活：几乎celery的各个组件都可以被扩展及自定制

详情
返回目录

05.Celery有哪些应用场景

高级 Django

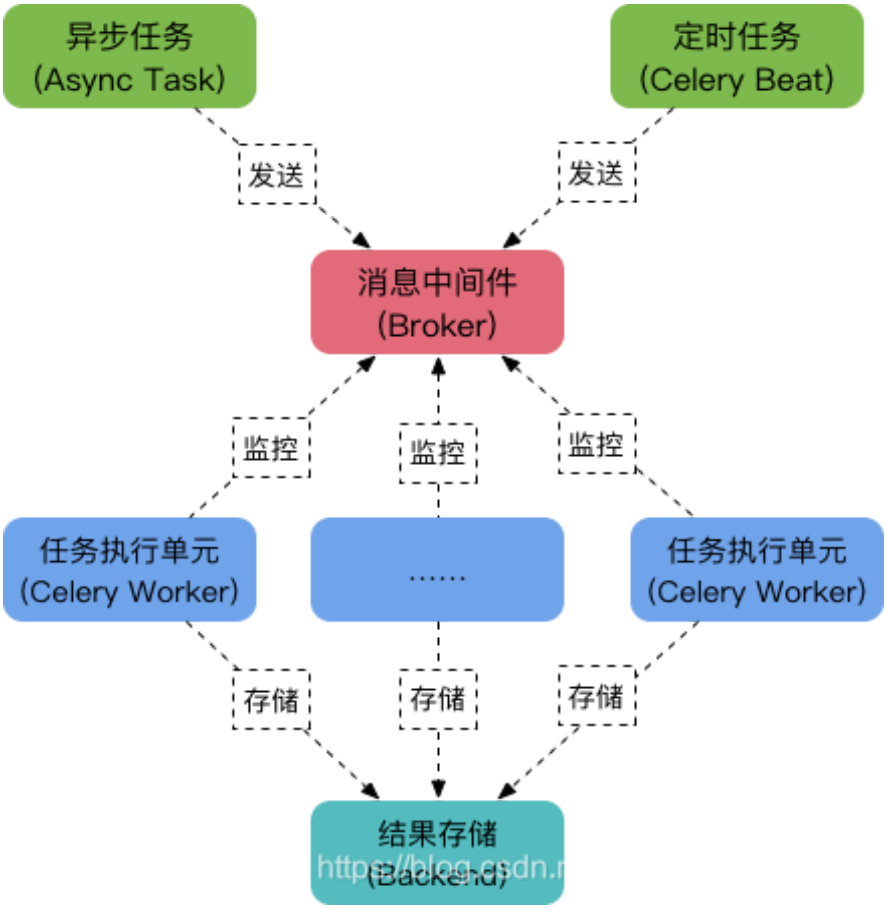
1. 异步任务：当用户在网站进行某个操作需要很长时间完成时，我们可以将这种操作交给Celery执行，直接返回给用户，等到Celery执行完成以后通知用户，大大提好网站的并发以及用户的体验感。例如：发送验证邮件
2. 定时任务：向定时清除沉余数据或批量在几百台机器执行某些命令或者任务，此时Celery可以轻松搞定。

详情
返回目录

06.Celery的工作原理是什么

高级 Django

Celery由以下三部分构成：消息中间件(Broker)、任务执行单元Worker、结果存储(Backend)，如下图：



工作原理： 任务模块Task包含异步任务和定时任务。其中，异步任务通常在业务逻辑中被触发并发往消息队列，而定时任务由Celery Beat进程周期性地将任务发往消息队列；任务执行单元Worker实时监视消息队列获取队列中的任务执行；Woker执行完任务后将结果保存在Backend中;

消息中间件Broker

消息中间件Broker官方提供了很多备选方案，支持RabbitMQ、Redis、Amazon SQS、MongoDB、Memcached 等，官方推荐RabbitMQ。

任务执行单元Worker

Worker是任务执行单元，负责从消息队列中取出任务执行，它可以启动一个或者多个，也可以启动在不同的机器节点，这就是其实现分布式的核心。

结果存储Backend

Backend结果存储官方也提供了诸多的存储方式支持：RabbitMQ、Redis、Memcached,SQLAlchemy, Django ORM、Apache Cassandra、Elasticsearch。

详情
返回目录

数据库模块

MySQL

01.NoSQL和SQL数据库之间有什么不同？

初级 MySQL

- SQL数据库主要称为关系数据库;
- NoSQL数据库主要称为非关系数据库或分布式数据库;

- **结构**

> SQL数据库是最广泛使用的选项之一，使其成为安全的选择，尤其适用于复杂的查询。但它具有一定的限制性。SQL数据库是基于表的，它要求您在使用之前使用预定义模式来确定数据的结构。此外，您的所有数据都必须遵循相同的结构。这可能需要大量的前期准备，这意味着结构的变化既困难又对整个系统造成破坏。

> NoSQL数据库具有非结构化数据的动态模式。他是键值对，基于文档的，图形数据库或宽列存储。数据以多种方式存储，这意味着它可以是面向文档，面向列，基于图形或组织为KeyValue存储。这种灵活性意味着可以在没有首先定义结构的情况下创建文档。每个文档也可以有自己独特的结构。语法因数据库而异，您可以随时添加字段。

- **可伸缩性**

> SQL数据库是可垂直扩展的。这意味着您可以通过增加RAM，CPU或SSD等功能来增加单个服务器的负载。

> NoSQL数据库可以横向扩展。这意味着您可以通过分片或在NoSQL数据库中添加更多服务器来处理更多流量。

- **遵循的属性**

> SQL数据库遵循ACID属性（原子性，一致性，隔离性和持久性）

> NoSQL数据库遵循Brewers CAP定理（一致性，可用性和分区容差）。

SQL	NoSQL
关系数据库管理系统	非关系或分布式数据库系统
静态或预定义的架构	动态架构
不适用于分层数据存储	适合分层数据存储
适合复杂查询	不太适合复杂的查询
可垂直扩展	可横向扩展
Mysql/Oracle/PostgreSQL	MongoDB/Redis/RavenDB

返回目录

02.MySQL支持哪些存储引擎?

初级 MySQL

MySQL支持多种存储引擎,比如InnoDB,MyISAM,Memory,Archive等等.在大多数的情况下,直接选择使用InnoDB引擎都是最合适的,InnoDB也是MySQL的默认存储引擎.

返回目录

03.InnoDB和MyISAM有什么区别?

初级 MySQL

1. InnoDB支持事务，而MyISAM不支持事务
2. InnoDB支持行级锁，而MyISAM支持表级锁
3. InnoDB支持MVCC, 而MyISAM不支持
4. InnoDB支持外键，而MyISAM不支持
5. InnoDB不支持全文索引，而MyISAM支持

返回目录

04.数据表类型有哪些

初级 MySQL

1. MyISAM
2. InnoDB
3. HEAP
4. BOB
5. ARCHIVE
6. CSV

返回目录

05.简单描述mysql中，索引，主键，唯一索引，联合索引的区别，对数据库的性能有什么影响（从读写两方面）

中级 MySQL

- 索引是一种特殊的文件，它们包含着对数据表里所有记录的引用指针。
- 普通索引(由关键字KEY或INDEX定义的索引)的唯一任务是加快对数据的访问速度。普通索引允许被索引的数据列包含重复的值。
- 唯一索引可以保证数据记录的唯一性。在数据列创建索引的时候添加关键字UNIQUE把它定义为唯一索引，能确保定某个数据列只包含彼此各不相同的值。

- 主键，是一种特殊的唯一索引，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。
- 联合索引可以涵盖多个数据列联合查询。
- 索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件。

返回目录

06.主键、外键和索引的区别

初级 MySQL

定义：

- 主键：唯一标识一条记录，不能有重复的，不允许为空
- 外键：表的外键是另一表的主键, 外键可以有重复的, 可以是空值
- 索引：该字段没有重复值，但可以有一个空值

作用：

- 主键：用来保证数据完整性
- 外键：用来和其他表建立联系用的
- 索引：是提高查询排序的速度

个数：

- 主键：主键只能有一个
- 外键：一个表可以有多个外键
- 索引：一个表可以有多个唯一索引

返回目录

07.索引的底层实现原理

初级 MySQL

实现原理：B+树，经过优化的B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此InnoDB建议为大部分表使用默认自增的主键作为主索引。

返回目录

08.建立索引有什么原则

初级 MySQL

1. 选择唯一性建立索引
2. 为经常需要排序、分组和联合操作的字段建立索引

3. 为常作为查询条件的字段建立索引
4. 限制索引的数目
5. 尽量使用数据量少的索引
6. 尽量使用前缀来索引
7. 删除不再使用或者很少使用的索引

返回目录

09.索引的优点和缺点是什么

初级 MySQL

优点

1. 快速访问数据表中的特定信息，提高检索速度
2. 创建唯一性索引，保证数据库表中每一行数据的唯一性。
3. 加速表和表之间的连接
4. 减少查询中分组和排序的时间

缺点：

1. 创建索引和维护索引需要耗费时间，随着数据量增加而增加；
2. 每个索引都需要占用物理空间，随着数据量增加而增加
3. 当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

返回目录

10.什么情况下不宜建立索引

中级 MySQL

- 对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引。
- 对于一些特殊的数据类型，不宜建立索引，比如文本字段（text）等

返回目录

11.什么是事务，其有哪些特性

初级 MySQL

- 事务是一系列的数据库操作，是数据库应用的基本逻辑单位。事务是被绑定在一起作为一个逻辑工作单元的SQL语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。要将有组语句作为事务考虑，就需要通过ACID测试，即原子性，一致性，隔离性和持久性。==简单一句话：事务就是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。==

事务特性：

1. **原子性**：即不可分割性，事务要么执行，要么不执行。
2. **一致性**：事务的执行使得数据库从一种一致性的正确状态转换成另一种一致性的正确状态
3. **隔离性**：事务操作之间彼此独立和透明互不影响。在事务正确提交之前，不允许把该事务对数据的任何修改提供给其他事务，
4. **持久性**：事务正确提交后，其结果将永久保存在数据库中。即使系统崩溃、修改的数据也不会丢失。

返回目录

12.事务有哪些隔离级别

初级 MySQL

1. **未提交读(Read Uncommitted)**：允许脏读，其他事务只要修改了数据，即使未提交，本事务也能看到修改后的数据值。也就是可能读取到其他会话中未提交事务修改的数据。
2. **已提交读(Read Committed)**：如果一个事务只能读到另一个已经提交的事务修改过的数据，并且其他事务每对该数据进行一次修改并提交后，该事务都能查询得到最新值，那么这种隔离级别就称之为已提交读。Oracle等多数数据库默认都是该级别 (不重复读)。
3. **可重复读(Repeated Read)**：可重复读。一个事务只能读到另一个已经提交的事务修改过的数据，但是第一次读过某条记录后，即使其他事务修改了该记录的值并且提交，该事务之后再读该条记录时，读到的仍是第一次读到的值，而不是每次都读到不同的数据。那么这种隔离级别就称之为可重复读。它是mysql默认的隔离级别。
4. **串行读(Serializable)**：如果一个事务先根据某些条件查询出一些记录，之后另一个事务又向表中插入了符合这些条件的记录，原先的事务再次按照该条件查询时，能把另一个事务插入的记录也读出来，那就意味着发生了幻读。

返回目录

13.Mysql中的事务回滚机制概述

初级 MySQL

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位，事务回滚是指将该事务已经完成的对数据库的更新操作撤销。

如要同时修改数据库中两个不同表时，如果它们不是一个事务的话，当第一个表修改完，可能第二个表修改过程中出现了异常而没能修改，此时就只有第二个表依旧是未修改之前的状态，而第一个表已经被修改完毕。而当你把它们设定为同一个事务的时候，当第一个表修改完，第二表修改出现异常而没能修改，第一个表和第二个表都要回到未修改的状态，这就是所谓的事务回滚

返回目录

14.SQL注入的主要特点

中级 MySQL

变种极多，攻击简单，危害极大

主要危害：

1. 未经授权操作数据库的数据
2. 恶意篡改网页
3. 私自添加系统账号或者使数据库使用者账号
4. 网页挂木马

返回目录

15.SQL语言包括哪几部分？每部分都有哪些操作关键字？

初级 MySQL

SQL语言包括数据定义(DDL)、数据操纵(DML),数据控制(DCL)和数据查询（DQL）四个部分。

- 数据定义：Create Table,Alter Table,Drop Table, Craete/Drop Index等
- 数据操纵：Select ,insert,update,delete,
- 数据控制：grant,revoke
- 数据查询：select

返回目录

16.完整性约束包括哪些？

中级 MySQL

数据完整性是指数据的精确和可靠性。

分为以下四类：

1. **实体完整性**：规定表的每一行在表中是惟一的实体。
2. **域完整性**：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。
3. **参照完整性**：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。
4. **用户定义的完整性**：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。

与表有关的约束：包括列约束(NOT NULL（非空约束）)和表约束(PRIMARY KEY、foreign key、check、UNIQUE)。

返回目录

17.什么是锁？

中级 MySQL

数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

加锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。

基本锁类型：锁包括行级锁和表级锁

返回目录

18.什么叫视图？游标是什么？

中级 MySQL

- 视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。
- 游标是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

返回目录

19.NULL是什么意思

初级 MySQL

NULL这个值表示UNKNOWN(未知):它不表示“”(空字符串)。对NULL这个值的任何比较都会生产一个NULL值。您不能把任何值与一个 NULL值进行比较，并在逻辑上希望获得一个答案。

返回目录

20.SQL语句中‘相关子查询’与‘非相关子查询’有什么区别

中级 MySQL

子查询：嵌套在其他查询中的查询称之。子查询又称内部，而包含子查询的语句称之外部查询（又称主查询）。所有的子查询可以分为两类，即相关子查询和非相关子查询

1. 非相关子查询是独立于外部查询的子查询，子查询总共执行一次，执行完毕后将值传递给外部查询。
2. 相关子查询的执行依赖于外部查询的数据，外部查询执行一行，子查询就执行一次。

故非相关子查询比相关子查询效率高

返回目录

21.char和varchar的区别

中级 MySQL

char是一种固定长度的类型，每个值都占用N个字节，如果某个长度小于N， MySQL就会在它的右边用空格字符补足。合适存储具有近似得长度（md5值,身份证，手机号）,长度比较短小的字符串。

varchar则是一种可变长度的类型，每个值只占用刚好够用的字节再加上一个用来记录其长度的字节。适合经常更新得字符串，更新时不会出现页分裂得情况，避免出现存储碎片，获得更好的io性能。

返回目录

22.谈谈三个范式

中级 MySQL

- 第一范式: 每个列都不可以再拆分
- 第二范式: 非主键列完全依赖于主键,而不能是依赖于主键的一部分.
- 第三范式: 非主键列只依赖于主键,不依赖于其他非主键.

在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有足够的理由.比如性能. 事实上我们经常会为了性能而妥协数据库的设计.

返回目录

23.如何进行SQL优化

高级 MySQL

1. 选择正确的存储引擎

> 以 MySQL为例，包括有两个存储引擎 MyISAM 和 InnoDB，每个引擎都有利有弊。 MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM 对于 SELECT COUNT(*) 这类的计算是超快无比的。

> InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。但是它支持“行锁”，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。

2. 优化字段的数据类型

> 记住一个原则，越小的列会越快。如果一个表只会有几列罢了（比如说字典表，配置表），那么，我们就没有理由使用 INT 来做主键，使用 MEDIUMINT, SMALLINT 或是更小的 TINYINT 会更经济一些。如果你不需要记录时间，使用 DATE 要比 DATETIME 好得多。当然，你也需要留够足够的扩展空间。

3. 为搜索字段添加索引

> 索引并不一定就是给主键或是唯一的字段。如果在你的表中，有某个字段你总要会经常用来做搜索，那么最好是为其建立索引，除非你要搜索的字段是大的文本字段，那应该建立全文索引。

4. 避免使用Select *从数据库里读出越多的数据，那么查询就会变得越慢。

> 如果你的数据库服务器和WEB服务器是两台独立的服务器的话，这还会增加网络传输的负载。即使你要查询数据表的所有字段，也尽量不要用*通配符，善用内置提供的字段排除定义也许能给带来更多的便

> 利。

5. 使用 ENUM 而不是 VARCHAR

> ENUM 类型是非常快和紧凑的。在实际上，其保存的是 TINYINT，但其外表上显示为字符串。这样一来，用这个字段来做一些选项列表变得相当的完美。例如，性别、民族、部门和状态之类的这些字段的取值是有限而且固定的，那么，你应该使用 ENUM 而不是 VARCHAR。

6. 尽可能的使用 NOT NULL

> 除非你有一个很特别的原因去使用 NULL 值，你应该总是让你的字段保持 NOT NULL。NULL其实需要额外的空间，并且，在你进行比较的时候，你的程序会更复杂。 当然，这里并不是说你就不能使用NULL了，现实情况是很复杂的，依然会有些情况下，你需要使用NULL值。

7. 固定长度的表会更快

> 如果表中的所有字段都是“固定长度”的，整个表会被认为是 “static” 或 “fixed-length”。例如，表中没有如下类型的字段： VARCHAR， TEXT， BLOB。只要你包括了其中一个这些字段，那么这个表就不是“固定长度静态表”了，这样，MySQL 引擎会用另一种方法来处理。固定长度的表会提高性能，因为MySQL搜寻得会更快一些，因为这些固定的长度是很容易计算下一个数据的偏移量的，所以读取的自然也会很快。而如果字段不是定长的，那么，每一次要找下一条的话，需要程序找到主键。

返回目录

24.说说对SQL语句优化有哪些方法

中级 MySQL

1. Where子句中：where表之间的连接必须写在其他Where条件之前，那些可以过滤掉最大数量记录的条件必须写在Where子句的末尾.HAVING最后。
2. 用EXISTS替代IN、用NOT EXISTS替代NOT IN。
3. 避免在索引列上使用计算
4. 对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by
5. 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描
6. 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描

返回目录

25.实践中如何优化MySQL

高级 MySQL

1. SQL语句及索引的优化
2. 数据库表结构的优化
3. 系统配置的优化
4. 硬件的优化

返回目录

26.如何设计一个高并发的系统

高级 MySQL

1. 数据库的优化，包括合理的事务隔离级别、SQL语句优化、索引的优化
2. 使用缓存，尽量减少数据库IO
3. 分布式数据库、分布式缓存
4. 服务器的负载均衡

返回目录

27.锁的优化策略

高级 MySQL

1. 读写分离
2. 分段加锁
3. 减少锁持有的时间
4. 多个线程尽量以相同的顺序去获取资源

返回目录

28.MySQL数据库作发布系统的存储，一天五万条以上的增量，预计运维三年,怎么优化

1. 设计良好的数据库结构，允许部分数据冗余，尽量避免join查询，提高效率。
2. 选择合适的表字段数据类型和存储引擎，适当的添加索引。
3. mysql库主从读写分离。
4. 找规律分表，减少单表中的数据量提高查询速度。
5. 添加缓存机制，比如memcached， apc等。
6. 不经常改动的页面，生成静态页面。
7. 书写高效率的SQL。比如不使用 SELECT *

返回目录

29.对于大流量的网站,您采用什么样的方法来解决各页面访问量统计问题

高级 MySQL

1. 确认服务器是否能支撑当前访问量。
2. 优化数据库访问。
3. 禁止外部访问链接（盗链）,比如图片盗链。
4. 控制文件下载。
5. 使用不同主机分流。
6. 使用浏览统计软件，了解访问量，有针对性的进行优化。

返回目录

30.主键使用自增ID还是UUID?

高级 MySQL

推荐使用自增ID,不要使用UUID。

因为在InnoDB存储引擎中，主键索引是作为聚簇索引存在的，也就是说，主键索引的B+树叶子节点上存储了主键索引以及全部的数据(按照顺序)，如果主键索引是自增ID，那么只需要不断向后排列即可。如果是UUID，由于到来的ID与原来的大小不确定，会造成非常多的数据插入或移动，然后导致产生很多的内存碎片,进而造成插入性能的下降。

总之,在数据量大一些的情况下,用自增主键性能会好一些。

返回目录

31.超大分页怎么处理?

高级 MySQL

超大的分页一般从三个方向上来解决.

- **优化SQL查询语句**
  数据库层面,这也是我们主要集中关注的(虽然收效没那么大),类似于select * from table where age > 20 limit 1000000,10这种查询其实也是有可以优化的余地的. 这条语句需要load1000000数据然后基本上全部丢弃,只取10条当然比较慢. 当时我们可以修改为select * from table where id in (select id from table where age > 20 limit 1000000,10).这样虽然也load了一百万的数据,但是由于索引覆盖,要查询的所有字段都在索引中,所以速度会很快. 同时如果ID连续的好,我们还可以select * from table where id > 1000000 limit 10,效率也是不错的,优化的可能性有许多种,但是==核心思想都一样,就是减少load的数据。==
- **优化产品设计**
  从需求的角度减少这种请求....主要是不做类似的需求(直接跳转到几百万页之后的具体某一页.只允许逐页查看或者按照给定的路线走,这样可预测,可缓存)以及防止ID泄漏且连续被人恶意攻击。
- **使用缓存**
  解决超大分页,其实主要是靠缓存,可预测性的提前查到内容,缓存至redis等k-V数据库中,直接返回即可。

返回目录

32.关心过业务系统里面的sql耗时吗?统计过慢查询吗?对慢查询都怎么优化过?

高级 MySQL

在业务系统中,除了使用主键进行的查询,其他的我都会在测试库上测试其耗时,慢查询的统计主要由运维在做,会定期将业务中的慢查询反馈给我们。

慢查询的优化首先要搞明白慢的原因是什么？
是查询条件没有命中索引？
是load了不需要的数据列？
还是数据量太大？

所以优化也是针对这三个方向来的,

1. 首先分析语句,看看是否load了额外的数据,可能是查询了多余的行并且抛弃掉了，可能是加载了许多结果中并不需要的列，对语句进行分析以及重写。
2. 分析语句的执行计划,然后获得其使用索引的情况,之后修改语句或者修改索引，使得语句可以尽可能的命中索引。
3. 如果对语句的优化已经无法进行，可以考虑表中的数据量是否太大，如果是的话可以进行横向或者纵向的分表。
4. 推荐使用自增ID,不要使用UUID.

33.Modelfirst和DBfirst区别

中级 Django

Model First 就是代表model优先，那么前提也就是先创建model，然后根据model自动建立数据库。
Datebase First 就是代表数据库优先，那么前提就是先创建数据库。

区别
当数据结构发生变化的时候， **Datebase First**选择的是从数据库生成。 **Model First**选择的是从模型生成数据库。

原因
在数据结构发生变化的时候，**Datebase First**编程方式中是选择从数据库更新模型，因此就导致了**Datebase First**是以数据库为主；而**Model First**选择的是空模型生成。

Redis模块

01. 什么是Redis？

初级 Redis

Redis本质上是一个Key-Value类型的内存数据库，很像memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据flush到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知性能最快的Key-Value DB。Redis的出色之处不仅仅是性能，Redis最大的魅力是支持保存多种数据结构，此外单个value的最大限制是1GB，不像 memcached只能保存1MB的数据，因此Redis可以用来实现很多有用的功能，比方说用他的List来做FIFO双向链表，实现一个轻量级的高性 能消息队列服务，用他的Set可以做高性能的tag系统等等。另外Redis也可以对存入的Key-Value设置expire时间，因此也可以被当作一 个功能加强版的memcached来用。 Redis的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此Redis适合的场景主要局限在较小数据量的高性能操作和运算上。

返回目录

02. Redis相比memcached有哪些优势？

初级 Redis

1. memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型
2. redis的速度比memcached快很多
3. redis可以持久化其数据

返回目录

03. Redis支持哪几种数据类型？

初级 Redis

String、List、Set、Sorted Set、hashes

返回目录

04. Redis主要消耗什么物理资源？

初级 Redis

redis是一种基于内存的高性能数据库--- 主要消耗于内存

返回目录

05. Redis的全称是什么？

初级 Redis

Remote Dictionary Server。

返回目录

06. Redis有哪几种数据淘汰策略？

初级 Redis

- noeviction:返回错误当内存限制达到并且客户端尝试执行会让更多内存被使用的命令（大部分的写入指令，但DEL和几个例外）
- allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。
- volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键,使得新添加的数据有空间存放。
- allkeys-random: 回收随机的键使得新添加的数据有空间存放。
- volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。
- volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键,使得新添加的数据有空间存放。

返回目录

07. Redis官方为什么不提供Windows版本？

初级 Redis

因为目前Linux版本已经相当稳定，而且用户量很大，无需开发windows版本，反而会带来兼容性等问题。

返回目录

08. 一个字符串类型的值能存储最大容量是多少？

初级 Redis

512M

返回目录

09. 为什么Redis需要把所有数据放到内存中？

初级 Redis

Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度为严重影响redis的性能。在内存越来越便宜的今天，redis将会越来越受欢迎。 如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

返回目录

10. Redis集群方案应该怎么做？都有哪些方案？

中级 Redis

1. twemproxy，大概概念是，它类似于一个代理方式，使用方法和普通redis无任何区别，设置好它下属的多个redis实例后，使用时在本需要连接redis的地方改为连接twemproxy，它会以一个代理的身份接收请求并使用一致性hash算法，将请求转接到具体redis，将结果再返回twemproxy。使用方式简便(相对redis只需修改连接端口)，对旧项目扩展的首选。 问题：twemproxy自身单端口实例的压力，使用一致性hash后，对redis节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。
2. codis，目前用的最多的集群方案，基本和twemproxy一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新hash节点。
3. redis cluster3.0自带的集群，特点在于他的分布式算法不是一致性hash，而是hash槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。
4. 在业务代码层实现，起几个毫无关联的redis实例，在代码层，对key 进行hash计算，然后去对应的redis实例操作数据。 这种方式对hash层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

返回目录

11. Redis集群方案什么情况下会导致整个集群不可用？

中级 Redis

有A， B， C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用。

返回目录

12. MySQL里有2000w数据，redis中只存20w的数据，如何保证redis中的数据都是热点数据？

中级 Redis

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。

返回目录

13. Redis有哪些适合的场景？

中级 Redis

1. 会话缓存（Session Cache）

> 最常用的一种使用Redis的情景是会话缓存（session cache）。用Redis缓存会话比其他存储（如Memcached）的优势在于：Redis提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？ 幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用Redis来缓存会话的文档。甚至广为人知的商业平台Magento也提供Redis的插件。

2. 全页缓存（FPC）

> 除基本的会话token之外，Redis还提供很简便的FPC平台。回到一致性问题，即使重启了Redis实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似PHP本地FPC。再次以Magento为例，Magento提供一个插件来使用Redis作为全页缓存后端。 此外，对WordPress的用户来说，Pantheon有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

3.队列

> Reids在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得Redis能作为一个很好的消息队列平台来使用。Redis作为队列使用的操作，就类似于本地程序语言（如Python）对 list 的 push/pop 操作。

如果你快速的在Google中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用Redis创建非常好的后端工具，以满足各种队列需求。例如，Celery有一个后台就是使用Redis作为broker，你可以从这里去查看。

4. 排行榜/计数器

> Redis在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单，Redis只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的10个用户–我们称之为“user_scores”，我们只需要像下面一样执行即可：

> 当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行： ZRANGE user_scores 0 10 WITHSCORES Agora Games就是一个很好的例子，用Ruby实现的，它的排行榜就是使用Redis来存储数据的，你可以在这里看到。

5. 发布/订阅

> 最后（但肯定不是最不重要的）是Redis的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用Redis的发布/订阅功能来建立聊天系统！（不，这是真的，你可以去核实）。

返回目录

14.Redis为什么快？除了他是内存型数据库外，还有什么原因

15. Redis和Redisson有什么关系？

中级 Redis

Redisson是一个高级的分布式协调Redis客服端，能帮助用户在分布式环境中轻松实现一些Java的对象 (Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog)。

返回目录

16. Redis与Redisson对比有什么优缺点？

中级 Redis

Jedis是Redis的Java实现的客户端，其API提供了比较全面的Redis命令的支持；Redisson实现了分布式和可扩展的Java数据结构，和Jedis相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等Redis特性。Redisson的宗旨是促进使用者对Redis的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

返回目录

17. Redis如何设置密码及验证密码？

初级 Redis

设置密码：config set requirepass 123456

授权密码：auth 123456

返回目录

18. 说说Redis哈希槽的概念？

初级 Redis

Redis集群没有使用一致性hash,而是引入了哈希槽的概念，Redis集群有16384个哈希槽，每个key通过CRC16校验后对16384取模来决定放置哪个槽，集群的每个节点负责一部分hash槽。

返回目录

19. Redis集群的主从复制模型是怎样的？

中级 Redis

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有N-1个复制品.

返回目录

20. Redis集群会有写操作丢失吗？为什么？

中级 Redis

Redis并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

返回目录

21. Redis集群之间是如何复制的？

初级 Redis

异步复制

返回目录

22. Redis集群最大节点个数是多少？

初级 Redis

16384个。

返回目录

23. Redis集群如何选择数据库？

初级 Redis

Redis集群目前无法做数据库选择，默认在0数据库。

返回目录

24. 怎么测试Redis的连通性？

初级 Redis

ping

返回目录

25. Redis中的管道有什么用？

初级 Redis

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应。这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多POP3协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

返回目录

26. 怎么理解Redis事务？

初级 Redis

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

返回目录

27. Redis事务相关的命令有哪几个？

初级 Redis

MULTI、EXEC、DISCARD、WATCH

返回目录

28. Redis key的过期时间和永久有效分别怎么设置？

初级 Redis

EXPIRE和PERSIST命令。

返回目录

29. Redis如何做内存优化？

中级 Redis

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的web系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的key,而是应该把这个用户的所有信息存储到一张散列表里面.

返回目录

30. Redis回收进程如何工作的？

中级 Redis

一个客户端运行了新的命令，添加了新的数据。

Redi检查内存使用情况，如果大于maxmemory的限制, 则根据设定好的策略进行回收。

一个新的命令被执行，等等。

所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下。

如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

返回目录

31. Redis回收使用的是什么算法？

中级 Redis

LRU算法

返回目录

32. Redis如何做大量数据插入？

中级 Redis

Redis2.6开始redis-cli支持一种新的被称之为pipe mode的新模式用于执行大量数据插入工作。

返回目录

33. 为什么要做Redis分区？

中级 Redis

分区可以让Redis管理更大的内存，Redis将可以使用所有机器的内存。如果没有分区，你最多只能使用一台机器的内存。分区使Redis的计算能力通过简单地增加计算机得到成倍提升,Redis的网络带宽也会随着计算机和网卡的增加而成倍增长。

返回目录

34. 你知道有哪些Redis分区实现方案？

中级 Redis

客户端分区就是在客户端就已经决定数据会被存储到哪个redis节点或者从哪个redis节点读取。大多数客户端已经实现了客户端分区。

代理分区 意味着客户端将请求发送给代理，然后代理决定去哪个节点写数据或者读数据。代理根据分区规则决定请求哪些Redis实例，然后根据Redis的响应结果返回给客户端。redis和memcached的一种代理实现就是Twemproxy

查询路由(Query routing) 的意思是客户端随机地请求任意一个redis实例，然后由Redis将请求转发给正确的Redis节点。Redis Cluster实现了一种混合形式的查询路由，但并不是直接将请求从一个redis节点转发到另一个redis节点，而是在客户端的帮助下直接redirected到正确的redis节点。

返回目录

35. Redis分区有什么缺点？

中级 Redis

涉及多个key的操作通常不会被支持。例如你不能对两个集合求交集，因为他们可能被存储到不同的Redis实例（实际上这种情况也有办法，但是不能直接使用交集指令）。

同时操作多个key,则不能使用Redis事务.

分区使用的粒度是key，不能使用一个非常长的排序key存储一个数据集（The partitioning granularity is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set）.

当使用分区的时候，数据处理会非常复杂，例如为了备份你必须从不同的Redis实例和主机同时收集RDB / AOF文件。

分区时动态扩容或缩容可能非常复杂。Redis集群在运行时增加或者删除Redis节点，能做到最大程度对用户透明地数据再平衡，但其他一些客户端分区或者代理分区方法则不支持这种特性。然而，有一种预分片的技术也可以较好的解决这个问题。

返回目录

36. Redis持久化数据和缓存怎么做扩容？

高级 Redis

如果Redis被当做缓存使用，使用一致性哈希实现动态扩容缩容。

如果Redis被当做一个持久化存储使用，必须使用固定的keys-to-nodes映射关系，节点的数量一旦确定不能变化。否则的话(即Redis节点需要动态变化的情况），必须使用可以在运行时进行数据再平衡的一套系统，而当前只有Redis集群可以做到这样。

返回目录

37. 分布式Redis是前期做还是后期规模上来了再做好？为什么？

高级 Redis

既然Redis是如此的轻量（单实例只使用1M内存）,为防止以后的扩容，最好的办法就是一开始就启动较多实例。即便你只有一台服务器，你也可以一开始就让Redis以分布式的方式运行，使用分区，在同一台服务器上启动多个实例。

一开始就多设置几个Redis实例，例如32或者64个实例，对大多数用户来说这操作起来可能比较麻烦，但是从长久来看做这点牺牲是值得的。

这样的话，当你的数据不断增长，需要更多的Redis服务器时，你需要做的就是仅仅将Redis实例从一台服务迁移到另外一台服务器而已（而不用考虑重新分区的问题）。一旦你添加了另一台服务器，你需要将你一半的Redis实例从第一台机器迁移到第二台机器。

返回目录

38. Twemproxy是什么？

高级 Redis

Twemproxy是Twitter维护的（缓存）代理系统，代理Memcached的ASCII协议和Redis协议。它是单线程程序，使用c语言编写，运行起来非常快。它是采用Apache 2.0 license的开源软件。 Twemproxy支持自动分区，如果其代理的其中一个Redis节点不可用时，会自动将该节点排除（这将改变原来的keys-instances的映射关系，所以你应该仅在把Redis当缓存时使用Twemproxy)。Twemproxy本身不存在单点问题，因为你可以启动多个Twemproxy实例，然后让你的客户端去连接任意一个Twemproxy实例。 Twemproxy是Redis客户端和服务器端的一个中间层，由它来处理分区功能应该不算复杂，并且应该算比较可靠的。

返回目录

39. 支持一致性哈希的客户端有哪些？

高级 Redis

Redis-rb、Predis等。

返回目录

40. Redis与其他key-value存储有什么不同？

高级 Redis

Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。

Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，应为数据量不能大于硬件内存。在内存数据库方面的另一个优点是， 相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。 同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

41. Redis的内存占用情况怎么样？

高级 Redis

给你举个例子： 100万个键值对（键是0到999999值是字符串“hello world”）在我的32位的Mac笔记本上 用了100MB。同样的数据放到一个key里只需要16MB， 这是因为键值有一个很大的开销。 在Memcached上执行也是类似的结果，但是相对Redis的开销要小一点点，因为Redis会记录类型信息引用计数等等。

当然，大键值对时两者的比例要好很多。

64位的系统比32位的需要更多的内存开销，尤其是键值对都较小时，这是因为64位的系统里指针占用了8个字节。 但是，当然，64位系统支持更大的内存，所以为了运行大型的Redis服务器或多或少的需要使用64位的系统。

返回目录

42. 都有哪些办法可以降低Redis的内存使用情况呢？

高级 Redis

如果你使用的是32位的Redis实例，可以好好利用Hash,list,sorted set,set等集合类型数据，因为通常情况下很多小的Key-Value可以用更紧凑的方式存放到一起。

返回目录

43. 查看Redis使用情况及状态信息用什么命令？

中级 Redis

info

返回目录

44. Redis的内存用完了会发生什么？

高级 Redis

如果达到设置的上限，Redis的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将Redis当缓存来使用配置淘汰机制，当Redis达到内存上限时会冲刷掉旧的内容。

返回目录

45. Redis是单线程的，如何提高多核CPU的利用率？

高级 Redis

可以在同一个服务器部署多个Redis的实例，并把他们当作不同的服务器来使用，在某些时候，无论如何一个服务器是不够的， 所以，如果你想使用多个CPU，你可以考虑一下分片（shard）。

返回目录

46. 一个Redis实例最多能存放多少的keys？List、Set、Sorted Set他们最多能存放多少元素？

高级 Redis

理论上Redis可以处理多达232的keys，并且在实际中进行了测试，每个实例至少存放了2亿5千万的keys。我们正在测试一些较大的值。

任何list、set、和sorted set都可以放232个元素。

换句话说，Redis的存储极限是系统中的可用内存值。

返回目录

47. Redis常见性能问题和解决方案？

高级 Redis

1. Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件
2. 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次
3. 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内
4. 尽量避免在压力很大的主库上增加从库
5. 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

返回目录

48. Redis提供了哪几种持久化方式？

高级 Redis

RDB持久化方式能够在指定的时间间隔能对你的数据进行快照存储.

AOF持久化方式记录每次对服务器写的操作,当服务器重启的时候会重新执行这些命令来恢复原始的数据,AOF命令以redis协议追加保存每次写的操作到文件末尾.Redis还能对AOF文件进行后台重写,使得AOF文件的体积不至于过大.

如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式.

你也可以同时开启两种持久化方式, 在这种情况下, 当redis重启的时候会优先载入AOF文件来恢复原始的数据,因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整.

最重要的事情是了解RDB和AOF持久化方式的不同,让我们以RDB持久化方式开始。

返回目录

49. 如何选择合适的持久化方式？

高级 Redis

一般来说， 如果想达到足以媲美PostgreSQL的数据安全性， 你应该同时使用两种持久化功能。如果你非常关心你的数据， 但仍然可以承受数分钟以内的数据丢失，那么你可以只使用RDB持久化。

有很多用户都只使用AOF持久化，但并不推荐这种方式：因为定时生成RDB快照（snapshot）非常便于进行数据库备份， 并且 RDB 恢复数据集的速度也要比AOF恢复的速度要快，除此之外， 使用RDB还可以避免之前提到的AOF程序的bug。

返回目录

50. 修改配置不重启Redis会实时生效吗？

高级 Redis

针对运行实例，有许多配置选项可以通过 CONFIG SET 命令进行修改，而无需执行任何形式的重启。 从 Redis 2.2 开始，可以从 AOF 切换到 RDB 的快照持久性或其他方式而不需要重启 Redis。检索 'CONFIG GET * 命令获取更多信息。但偶尔重新启动是必须的，如为升级 Redis 程序到新的版本，或者当你需要修改某些目前 CONFIG 命令还不支持的配置参数的时候。

返回目录

部署模块

Nginx

01.什么是Nginx

初级 Nginx

Nginx是一个高性能代理服务，接收客户端发送过来的HTTP请求和websocket请求，响应静态文件请求和转发动态请求。

返回目录

02.负载均衡什么意思

中级 Nginx

负载均衡（Server Load Balancer）是将访问流量根据转发策略分发到后端多台服务器的流量分发控制服务。负载均衡扩展了应用的服务能力，增强了应用的可用性。

返回目录

03.如何部署Django项目

高级 Nginx

1. Nginx+uWSGI+Django
2. Nginx+Tornado+Django

返回目录

04.为什么正式部署时不要开启DEBUG = True配置

中级 Nginx

因为对于一个在线网站, 将路由暴露出来, 是一件非常危险的事情, 所以我们要关掉django的debug模式

返回目录

常用算法模块

01.你知道的python排序算法

中级 python

1. 冒泡排序（Bubble Sort）
2. 选择排序（Selection Sort）

3. 插入排序（Insertion Sort）
4. 希尔排序（Shell Sort）
5. 归并排序（Merge Sort）
6. 快速排序（Quick Sort）
7. 堆排序（Heap Sort）
8. 计数排序（Counting Sort）
9. 桶排序（Bucket Sort）
10. 基数排序（Radix Sort）

详情
返回目录

02.你了解的高级语言中的垃圾回收机制有哪些？Python中用的是什么？

高级 python

详细
返回目录

03.介绍下你知道的缓存相关的算法

高级

详细
返回目录

04.介绍下你知道的负载均衡相关的算法

高级

- **轮询**
  将所有请求，依次分发到每台服务器上，适合服务器硬件相同的场景。
  优点：服务器请求数目相同；
  缺点：服务器压力不一样，不适合服务器配置不同的情况；
- **随机**
  请求随机分配到各台服务器上。
  优点：使用简单；
  缺点：不适合机器配置不同的场景
- **最少链接**
  将请求分配到连接数最少的服务器上（目前处理请求最少的服务器）。
  优点：根据服务器当前的请求处理情况，动态分配；
  缺点：算法实现相对复杂，需要监控服务器请求连接数；
- **Hash（源地址散列）**
  根据IP地址进行Hash计算，得到IP地址。

优点：将来自同一IP地址的请求，同一会话期内，转发到相同的服务器；实现会话粘滞。
缺点：目标服务器宕机后，会话会丢失；

- **加权**
  在轮询，随机，最少链接，Hash等算法的基础上，通过加权的方式，进行负载服务器分配。
  优点：根据权重，调节转发服务器的请求数目；
  缺点：使用相对复杂；

详细
返回目录

其他模块

01.介绍你项目的架构设计

返回目录

02.Tornado是什么

高级 Tornado

Tornado是使用Python编写的Web服务器兼Web应用框架，与主流Web服务器框架不同的是，Tornado是异步非阻塞式服务器，得益于非阻塞式和对epoll模型的运用，Tornado是实时Web服务的一个理想框架，它非常适合开发长轮询、WebSocket和需要与每个用户建立持久连接的应用。

- 特点

1. 轻量级Web框架
2. 异步非阻塞IO处理方式
3. Tornado采用的单进程单线程异步IO的网络模式，其高性能源于Tornado基于Linux的Epoll（UNIX为kqueue）的异步网络IO。
4. 出色的抗负载能力
5. 不依赖多进程或多线程
6. WSGI全栈替代产品
7. WSGI把应用（Application）和服务器（Server）结合起来，Tornado既可以是WSGI应用也可以是WSGI服务。
8. 既是WebServer也是WebFramework

03.Tornado和Django区别

高级 Tornado

	Django	**Tornado**
优点	大和全（重量级框架）	少而精（轻量级框架）

	Django	Tornado
	自带orm， template， view	注重性能优越，速度快
	需要的功能也可以去找第三方的app	异步非阻塞，解决高并发 （请求处理是基于回调的非阻塞调用）
	注重高效开发	内嵌了HTTP服务器，合适websockets和长连接
	全自动化的管理后台 （只需要使用起ORM，做简单的定义， 就能自动生成数据库结构，全功能的管理后台）	单线程的异步网络程序， 默认启动时根据CPU数量运行多个实例； 利用CPU多核的优势
缺点	template不怎么好用（来自自身的缺点）	模板和数据库部分有很多第三方的模块可供选择，这样不利于封装为一个功能模块
	数据库用nosql不方便（来自自身的缺点）	
	如果功能不多，容易臃肿	

总结：

1. 要性能， Tornado 首选；
2. 要开发速度，Django 和 Flask 都行，区别是 Flask 把许多功能交给第三方库去完成了，因此 Flask 更为灵活。

综上所述：

- Django适合初学者或者小团队的快速开发，适合做管理类、博客类网站、或者功能十分复杂需求十分多的网站
- Tornado适合高度定制，适合访问量大，异步情况多的网站

详情

返回目录

云服务模块

01.你使用过哪些云服务技术？

高级 云

返回目录

02.常用的云技术有哪些，请列举4个

高级 云

- ECS实例
- OSS
- RDS
- CDN

返回目录