# 不出现的面试题、备份面试题（防止遗忘一些重要部分来这重新找）

* ****为了保证高可用，最好是以集群形态来部署 ZooKeeper，这样只要集群中大部分机器是可用的（能够容忍一定的机器故障），那么 ZooKeeper 本身仍然是可用的。****
* ****ZooKeeper有临时节点的概念。 当创建临时节点的客户端会话一直保持活动，瞬时节点就一直存在。而当会话终结时，瞬时节点被删除。持久节点是指一旦这个ZNode被创建了，除非主动进行ZNode的移除操作，否则这个ZNode将一直保存在Zookeeper上。****

## **String 、StringBuilder 、StringBuffer 的区别？**

****可变性****

String 类中使用 final 关键字修饰字符数组来保存字符串，private final char value[]，所以 String 对象是不可变的。

而 StringBuilder 与 StringBuffer 都继承自 AbstractStringBuilder 类，在 AbstractStringBuilder 中也是使用字符数组保存字符串char[]value 但是没有用 final 关键字修饰，所以这两种对象都是可变的。

****线程安全性****

String 中的对象是不可变的，也就可以理解为常量，线程安全。AbstractStringBuilder 是 StringBuilder 与 StringBuffer 的公共父类，定义了一些字符串的基本操作，如 expandCapacity、append、insert、indexOf 等公共方法。StringBuffer 对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。StringBuilder 并没有对方法进行加同步锁，所以是非线程安全的。

****性能****

每次对 String 类型进行改变的时候，都会生成一个新的 String 对象，然后将指针指向新的 String 对象。StringBuffer 每次都会对 StringBuffer 对象本身进行操作，而不是生成新的对象并改变对象引用。相同情况下使用 StringBuilder 相比使用 StringBuffer 仅能获得 10%~15% 左右的性能提升，但却要冒多线程不安全的风险。

****对于三者使用的总结：****

1. 操作少量的数据: 适用 String
2. 单线程操作字符串缓冲区下操作大量数据: 适用 StringBuilder
3. 多线程操作字符串缓冲区下操作大量数据: 适用 StringBuffer

## 谈谈 JVM 的内存结构和内存分配？

a) Java 内存模型

Java 虚拟机将其管辖的内存大致分三个逻辑部分：方法区(Method Area)、Java 栈和 Java 堆。

1、方法区是静态分配的，编译器将变量绑定在某个存储位置上，而且这些绑定不会在运行时改变。

常数池，源代码中的命名常量、String 常量和 static 变量保存在方法区。

2、Java Stack 是一个逻辑概念，特点是后进先出。一个栈的空间可能是连续的，也可能是不连续的。

最典型的 Stack 应用是方法的调用，Java 虚拟机每调用一次方法就创建一个方法帧（frame），退出该方法则对应的 方法帧被弹出(pop)。栈中存储的数据也是运行时确定的。

3、Java 堆分配(heap allocation)意味着以随意的顺序，在运行时进行存储空间分配和收回的内存管理模型。

堆中存储的数据常常是大小、数量和生命期在编译时无法确定的。Java 对象的内存总是在 heap 中分配。我们每天都在写代码，每天都在使用 JVM 的内存。

b) java 内存分配

1、基础数据类型直接在栈空间分配;

2、方法的形式参数，直接在栈空间分配，当方法调用完成后从栈空间回收;

3、引用数据类型，需要用 new 来创建，既在栈空间分配一个地址空间，又在堆空间分配对象的类变量;

4、方法的引用参数，在栈空间分配一个地址空间，并指向堆空间的对象区，当方法调用完后从栈空间回收;

5、局部变量 new 出来时，在栈空间和堆空间中分配空间，当局部变量生命周期结束后，栈空间立刻被回收，堆空间区域等待 GC 回收;

6、方法调用时传入的实际参数，先在栈空间分配，在方法调用完成后从栈空间释放;

7、字符串常量在 DATA 区域分配 ，this 在堆空间分配;

8、数组既在栈空间分配数组名称， 又在堆空间分配数组实际的大小！

## **讲**一下 **jvm 调优。**

a,堆大小设置

b,回收器选择

c,辅助信息

JVM 提供了大量命令行参数，打印信息，供调试使用；

## **讲一下 jvm 的组成。**

JVM 由类加载器子系统、运行时数据区、执行引擎以及本地方法接口组成

## & 和 && 的区别？

&运算符有两种用法：(1)按位与；(2)逻辑与。

&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true 整个表达式的值才是 true。

&&之所以称为短路运算是因为，如果&&左边的表达式的值是 false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是 null 而且不是空字符串，应当写为 username != null &&!username.equals("")，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的 equals 比较，否则会产生 NullPointerException 异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

## 字节流如何转为字符流？

字节输入流转字符输入流通过 InputStreamReader 实现，该类的构造函数可以传入 InputStream 对象。

字节输出流转字符输出流通过 OutputStreamWriter 实现，该类的构造函数可以传入 OutputStream 对象。

## 简述几种JAVA设计模式,例如:单例模式

一、工厂模式 :

我个人的理解是这样的，JAVA多态的一种体现，比如定义了一个接口或者抽象类的变量，可以由它的子类或者实现者来初始化它，但是我们在使用其中的抽象方法的时候，实际上是调用它的子类或者实现者重写的方法

1. 观察者模式 :

介绍

意图：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

主要解决：一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作。

何时使用：一个对象（目标对象）的状态发生改变，所有的依赖对象（观察者对象）都将得到通知，进行广播通知。

如何解决：使用面向对象技术，可以将这种依赖关系弱化。

关键代码：在抽象类里有一个 ArrayList 存放观察者们。

三、模板模式 :

四、单例模式 :

  定义：在内存中始终只有一个对象

  饿汉式

  特点:当前某一个一加载,就会创建一个对象

  需要将该类的无参构造私有化

  在成员变量创建该类的实例

需要提供公共的访问方法(实际开发中使用这种方式)

  懒汉式

  特点:并不是加载某个类就直接创建对象,而是需要的时候在创建对象

  1)懒加载(延迟加载)

2)线程的安全问题

  线程安全问题检验标准:

  a:是否是多线程环境

  b:是否有共享数据

   c:是否有多条语句对共享数据进行操作!

## Java 中异常分为哪些种类？

按照异常需要处理的时机分为编译时异常（也叫强制性异常）也叫 CheckedException 和运行时异常（也叫非强制性异常）也叫 RuntimeException。

请写出你最常见的 5 个 RuntimeException（2017-11-22-wzz）

## 下面列举几个常见的 RuntimeException？

1）java.lang.NullPointerException 空指针异常；出现原因：调用了未经初始化的对象或者是不存在的对象。

2）java.lang.ClassNotFoundException 指定的类找不到；出现原因：类的名称和路径加载错误；通常都是程序试图通过字符串来加载某个类时可能引发异常。

3）java.lang.NumberFormatException 字符串转换为数字异常；出现原因：字符型数据中包含非数字型字符。

4）SQLException SQL 异常，常见于操作数据库时的 SQL 语句错误。

5）java.lang.NoSuchMethodException 方法不存在异常。

## Java中Error与Exception的区别

Exception：

1．可以是可被控制(checked) 或不可控制的(unchecked)。

2．表示一个由程序员导致的错误。

3．应该在应用程序级被处理。

Error：

1．总是不可控制的(unchecked)。

2．经常用来用于表示系统错误或低层资源的错误。

3．如何可能的话，应该在系统级被捕捉。

## 工具类Collections中的常用方法?集合排序是怎么实现的?

下面是集合中常用的方法

//static void swap(List list, int i, int j) :将指定列表中的两个索引进行位置互换

//static void  sort(List<T> list) :按照列表中元素的自然顺序进行排序

//static void shuffle(List list):傻否，随机置换

//static void reverse(List list)  :反转

//static void fill(List list, Object obj) :使用指定的对象填充指定列表的所有元素

//static void copy(List dest, List src) :是把源列表中的数据覆盖到目标列表

//static int  binarySearch(List list, Object key) 使用二分查找法查找指定元素在指定列表的索引位置

Collections.sort（List list）称为自然排序，排序的对象需要实现可比较的接口，重写其compareTo（）方法，方法体中实现对象的比较大小规则

Collections.sort（List list，Comparator c），定制排序，或自定义排序，需编写匿名类，先新一个Comparator接口的比较器对象c，同时实现compare（）其方法

第一种方法不够灵活，实体类实现了媲美接口后，会增加耦合，如果在项目中不同的位置需要根据不同的属性调用排序方法时，需要反复修改比较规则（按名称还是按年龄），二者只能选择其一，会起冲突。第二种就很好地解决了这个问题。在需要的地方，创建个内部类的实例，重写其比较方法即可

## 用timer和task实现一个定时器,每隔1分钟执行一个函数A(),延时10秒执行.

public void schedule ( TimerTask task,long delay,long period ) ;

第一个参数是任务，第二个任务是多长时间后执行，第三个参数是间断时间

public class Test {  
 public static void main(String[] args) {  
 Timer timer = new Timer();  
 timer.schedule(new TimerTask() {  
 @Override  
 public void run() {  
 A(); }  
 }, 60\*1000,10000 );  
 }  
 }

## [java中的静态变量、静态方法与静态代码块详解与初始化顺序](https://www.cnblogs.com/lubocsu/p/5099558.html)

我们知道类的生命周期分为装载、连接、初始化、使用和卸载的五个过程。  
其中静态代码在类的初始化阶段被初始化。

而非静态代码则在类的使用阶段（也就是实例化一个类的时候）才会被初始化。

静态变量

可以将静态变量理解为类变量（与对象无关），而实例变量则属于一个特定的对象。

静态变量有两种情况：

静态变量是基本数据类型，这种情况下在类的外部不必创建该类的实例就可以直接使用

静态变量是一个引用。这种情况比较特殊，主要问题是由于静态变量是一个对象的引用，那么必须初始化这个对象之后才能将引用指向它。

因此如果要把一个引用定义成static的，就必须在定义的时候就对其对象进行初始化。

静态方法

与类变量不同，方法（静态方法与实例方法）在内存中只有一份，无论该类有多少个实例，都共用一个方法。

静态方法与实例方法的不同主要有：

静态方法可以直接使用，而实例方法必须在类实例化之后通过对象来调用。

在外部调用静态方法时，可以使用“类名.方法名”或者“对象名.方法名”的形式。

实例方法只能使用这种方式对象名.方法名。

静态方法只允许访问静态成员。而实例方法中可以访问静态成员和实例成员。

静态方法中不能

静态代码块

静态代码块主要用于类的初始化。它只执行一次，并且在同属于一个类的main函数之前执行。

静态代码块的特点主要有：

静态代码块会在类被加载时自动执行。

静态代码块只能定义在类里面，不能定义在方法里面。

静态代码块里的变量都是局部变量，只在块内有效。

一个类中可以定义多个静态代码块，按顺序执行。

静态代码块只能访问类的静态成员，而不允许访问实例成员。

静态代码块和静态函数的区别

java 静态代码块：

一般情况下,如果有些代码必须在项目启动前就执行的时候,需要使用静态代码块,这种代码是主动执行的，它只执行一次，并且在同属于一个类的main函数之前执行。

静态函数：

需要在项目启动的时候就初始化,在不创建对象的情况下,其他程序来调用的时候,需要使用静态方法,这种代码是被动执行的.

注意：

（1）静态变量是属于整个类的变量而不是属于某个对象的。注意不能把任何方法体内的变量声明为静态，例如：  
fun()  
{  
static int i=0;//非法。  
}

（2）一个类可以使用不包含在任何方法体中的静态代码块，当类被载入时，静态代码块被执行，且只被执行一次，静态块常用来执行类属性的初始化。例如：  
static  
{

}

主程序类中的的静态变量先于静态代码块初始化，其后进入主函数类(程序入口处)，其后根据静态函数的调用情况，才能选择性的初始化。

## HashMap和Hashtable的区别？

HashMap和Hashtable都实现了Map接口.

HashMap可以接受为null的键值(key)和值(value)，而Hashtable则不行。

HashMap是非synchronized，而Hashtable是synchronized，这意味着Hashtable是线程安全的，多个线程可以共享一个Hashtable；

由于Hashtable是线程安全的也是synchronized，所以在单线程环境下它比HashMap要慢。如果你不需要同步，只需要单一线程，那么使用HashMap性能要好过Hashtable。

HashMap不能保证随着时间的推移Map中的元素次序是不变的。

迭代器不一样.

## 讲讲HashMap特点、原理、底层的数据结构,为什么是线程不安全的

一、特点

实现Map接口

使用hash算法,然后基于数组+链表来实现的

它可以接收null的键值和值

非同步，

存储着Entry(hash, key, value, next)对象。

底层结构:

HashMap是数组和链表的结合体。

简述HashMap工作原理

 HashMap是基于散列法（又称哈希法hashing）的原理，使用put(key, value)存储对象到HashMap中，使用get(key)从HashMap中获取对象

为什么是线程不安全的

多线程同时操作HashMap时会造成数据丢失问题. 比如, 同时操作HashMap的put()方法.扩容时候同时调用resize操作都会有这问题.

1. for each map.entrySet()，用临时变量保存 map.entrySet()。

## Java1.8 新特性？

Lambda 表达式 − Lambda允许把函数作为参数传递进方法中。

方法引用 − 可以直接引用已有Java类的方法或构造器。

默认方法 − 默认方法就是一个在接口里面有了一个实现的方法。

新工具 − 新的编译工具，如：Nashorn引擎 jjs、 类依赖分析器jdeps。

Stream API −新添加的Stream API（java.util.stream） 把真正的函数式编程风格引入到Java中。

Date Time API − 加强对日期与时间的处理。

Optional 类 − Optional 类已经成为 Java 8 类库的一部分，用来解决空指针异常。

Nashorn, JavaScript 引擎 − Java 8提供了一个新的Nashorn javascript引擎，它允许我们在JVM上运行特定的javascript应用。

# 6.前端

## 写出你常用的js调试方式.

## **JS断点调试**

Sources断点

给一段代码添加断点的流程是“F12（Ctrl + Shift + I）打开开发工具”——“点击Sources菜单”——“左侧树中找到相应文件”——“点击行号列”即完成在当前行添加/删除断点操作。当断点添加完毕 后，刷新页面JS执行到断点位置停住，在Sources界面会看到当前作用域中所有变量和值，只需对每个值进行验证即可完成我们题设验证要求。

Debugger断点

通过在代码中添加”debugger;”语句，当代 码执行到该语句的时候就会自动断点。接下去的操作就跟在Sources面板添加断点调试几乎一模一样，唯一的区别在于调试完后需要删除该语句。

DOM断点调试

DOM断点，顾名思义就是在DOM元素上添加断点，进而达到调试的目的。

## js里有哪些对象?new出来的那种.

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象。

可以说，js中对象【除了null ,undefined】都是可以用new出来的。  
js才是真真正正的[面向对象语言](https://www.baidu.com/s?wd=%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1%E8%AF%AD%E8%A8%80&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://blog.csdn.net/weixin_42440871/article/details/_blank)。

## 说一下常用的 Linux 命令

1. 目录切换命令

* cd ..（或cd../）： 切换到上一层目录
* cd /： 切换到系统根目录
* cd ~： 切换到用户主目录

1. 目录的操作命令(增删改查)

* mkdir 目录名称： 增加目录 rmdir 删除目录
* ls或者ll（ll是ls -l的别名，ll命令可以看到该目录下的所有目录和文件的详细信息）

1. 查看目录信息

* find 目录 参数： 寻找目录（查）
* mv 目录名称 新目录名称： 修改目录的名称（改）
* mv 目录名称 目录的新位置： 移动目录的位置---剪切（改）
* cp -r 目录名称 目录拷贝的目标位置： 拷贝目录（改），-r代表递归拷贝
* rm [-rf] 目录: 删除目录（删）

1. 文件的操作命令（增删改查）

* touch 文件名称: 文件的创建（增）
* cat/more/less/tail 文件名称 文件的查看（查）
* vim 文件： 修改文件的内容（改）
* rm -rf 文件： 删除文件（删）

1. 压缩文件的操作命令

* tar -zcvf 打包压缩后的文件名 要打包压缩的文件
* tar [-xvf] x：代表解压

6. 修改文件/目录的权限的命令

* Chmod

7. 其他常用命令

* pwd： 显示当前所在位置
* grep 要搜索的字符串 要搜索的文件 --color： 搜索命令，--color代表高亮显示
* ps -ef/ps -aux： 这两个命令都是查看当前系统正在运行进程，两者的区别是展示格式不同。如果想要查看特定的进程可以使用这样的格式：ps aux|grep redis （查看包括redis字符串的进程），也可使用 pgrep redis -a。注意：如果直接用ps（（Process Status））命令，会显示所有进程的状态，通常结合grep命令查看某进程的状态。
* kill -9 进程的pid： 杀死进程（-9 表示强制终止。）先用ps查找进程，然后用kill杀掉。

1. 网络通信命令

* 查看当前系统的网卡信息：ifconfig
* 查看与某台机器的连接情况：ping

## Linux 中如何查看日志？

动态打印日志信息：tail –f 日志文件

## Linux授权命令是？

授权命令：chmod

## **Mysql 优化有没有工具**

回答：

三个 MySQL 性能测试工具：The MySQL Benchmark Suite、MySQL super-smack、MyBench。除了第一个为 MySQL 性能测试工具，其他两个都为压力测试工具。

## **索引**

MySQL索引使用的数据结构主要有****BTree索引**** 和 ****哈希索引**** 。对于哈希索引来说，底层的数据结构就是哈希表，因此在绝大多数需求为单条记录查询的时候，可以选择哈希索引，查询性能最快；其余大部分场景，建议选择BTree索引。

MySQL的BTree索引使用的是B树中的B+Tree，但对于主要的两种存储引擎的实现方式是不同的。

* ****MyISAM:**** B+Tree叶节点的data域存放的是数据记录的地址。在索引检索的时候，首先按照B+Tree搜索算法搜索索引，如果指定的Key存在，则取出其 data 域的值，然后以 data 域的值为地址读取相应的数据记录。这被称为“非聚簇索引”。
* ****InnoDB:**** 其数据文件本身就是索引文件。相比MyISAM，索引文件和数据文件是分离的，其表数据文件本身就是按B+Tree组织的一个索引结构，树的叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。这被称为“聚簇索引（或聚集索引）”。而其余的索引都作为辅助索引，辅助索引的data域存储相应记录主键的值而不是地址，这也是和MyISAM不同的地方。****在根据主索引搜索时，直接找到key所在的节点即可取出数据；在根据辅助索引查找时，则需要先取出主键的值，再走一遍主索引。**** ****因此，在设计表的时候，不建议使用过长的字段作为主键，也不建议使用非单调的字段作为主键，这样会造成主索引频繁分裂。****

## **查询缓存的使用**

执行查询语句的时候，会先查询缓存。不过，MySQL 8.0 版本后移除，因为这个功能不太实用

my.cnf加入以下配置，重启MySQL开启查询缓存

query\_cache\_type=1query\_cache\_size=600000

MySQL执行以下命令也可以开启查询缓存

set global query\_cache\_type=1;

set global query\_cache\_size=600000;

如上，****开启查询缓存后在同样的查询条件以及数据情况下，会直接在缓存中返回结果****。这里的查询条件包括查询本身、当前要查询的数据库、客户端协议版本号等一些可能影响结果的信息。因此任何两个查询在任何字符上的不同都会导致缓存不命中。此外，如果查询中包含任何用户自定义函数、存储函数、用户变量、临时表、MySQL库中的系统表，其查询结果也不会被缓存。

缓存建立之后，MySQL的查询缓存系统会跟踪查询中涉及的每张表，如果这些表（数据或结构）发生变化，那么和这张表相关的所有缓存数据都将失效。

****缓存虽然能够提升数据库的查询性能，但是缓存同时也带来了额外的开销，每次查询后都要做一次缓存操作，失效后还要销毁。**** 因此，开启缓存查询要谨慎，尤其对于写密集的应用来说更是如此。如果开启，要注意合理控制缓存空间大小，一般来说其大小设置为几十MB比较合适。此外，****还可以通过sql\_cache和sql\_no\_cache来控制某个查询语句是否需要缓存：****

select sql\_no\_cache count(\*) from usr;

## **具名参数是啥？jdbc如何实现具名参数？**

使用spring的jdbcTemplate-----为什么使用具名参数

在JDBC用法中，SQL参数是用占位符？表示，并且受到位置的限制，定位参数的问题在于，一旦参数的位置发生变化，必须改变参数的绑定，在SpringJDBC中，绑定SQL参数的另一种选择是使用具名参数，SQL具名参数是按照名称绑定，而不是位置绑定。

什么是具名参数？

具名参数:SQL参数按名称(以冒号开头)而不是按位置进行指定. 具名参数更易于维护, 也提升了可读性.

具名参数只在NamedParameterJdbcTemplate 中得到支持。（SImpleJdbcTemplate已过时了）

jdbc如何实现具名参数？

 NamedParameterJdbcTemplate内部包含了一个JdbcTemplate，所以JdbcTemplate能做的事情NamedParameterJdbcTemplate都能干，NamedParameterJdbcTemplate相对于JdbcTemplate主要增加了参数可以命名的功能。

## mysql **索引失效的场景有哪些？like 做模糊查询的时候会失效吗？**

1.WHERE 字句的查询条件里有不等于号（WHERE column!=…），MYSQL 将无法使

用索引。

2.类似地，如果 WHERE 字句的查询条件里使用了函数（如： WHERE DAY(colu mn)=…）， MYSQL 将无法使用索引。

3.如果 WHERE 子句的查询条件里使用了比较操作符 LIKE 和 REGEXP，MYSQL

只有在搜索模板的第一个字符不是通配符的情况下才能使用索引。比如说，如果查询条件是 LIKE 'abc%',MYSQL 将使用索引；如果条件是 LIKE '%abc'，MYSQ

L 将不使用索引。

4.如果某个数据列里包含着许多重复的值，就算为它建立了索引也不会有很好的效果。比如说，如果某个数据列里包含了净是些诸如 “0/1”或“Y/N”等值，就没有必要为它创建一个索引。

5.索引有用的情况下就太多了。基本只要建立了索引，除了上面提到的索引不会使用的情况下之外，其他情况只要是使用在 WHERE 条件里，ORDER BY 字段，联表字段，一般都是有效的。 建立索引要的就是有效果。 不然还用它干吗？ 如果不能确定在某个字段上建立的索引是否有效果，只要实际进行测试下比较下执行时间就知道。

6.如果条件中有 or(并且其中有 or 的条件是不带索引的)，即使其中有条件带索引也不会使用(这也是为什么尽量少用 or 的原因)。注意：要想使用 or，又想让索引生效，只能将 or 条件中的每个列都加上索引

7.如果 mysql 估计使用全表扫描要比使用索引快,则不使用索引

问题二： Like 模糊查询，建立索引会失效

#### **Redis 分布式锁理解**

回答：

实现思想

获取锁的时候，使用 setnx 加锁，并使用 expire 命令为锁添加一个超时时间，超过该时间则自动释放

锁，锁的 value 值为一个随机生成的 UUID，通过此在释放锁的时候进行判断。

获取锁的时候还设置一个获取的超时时间，若超过这个时间则放弃获取锁。

释放锁的时候，通过 UUID 判断是不是该锁，若是该锁，则执行 delete 进行锁释放。

#### Redis在SSM框架里是怎么做的？

1. 安装好redis。输入启动指令启动redis。
2. 因为缓存对于我们整个的系统来说是通用功能。广告需要用，其它数据可能也会用到，所以我们将配置放在公共组件项目（pyg-common）中。在项目中引入jedis，spring-data-redis的jar包。
3. 创建配置文件。
4. 在服务层编写业务代码，把广告数据存入redis中。
5. 查看控制台得知先从数据库查询数据，然后存入redis中。之后就一直从redis中获取数据。

#### **redis 缓存目的？**

主要应用在门户网站首页广告信息的缓存。因为门户网站访问量较大，将广告缓存到 redis 中，可以降低数据库访问压力，提高查询性能。

使用redis保存页面中广告数据、搜索面板数据、查看商品的商品信息、购物车信息、生成订单号等.

### redis快速的原因

同问:Redis是个单线程程序，每秒100000QPS，为什么会这么快？

答：

1.完全基于内存，大部分都是内存操作，非常快速，数据存储在内存中，例如HashMap，优势是查找和操作的时间复杂度 都是O(1)；

2.数据结构简单，对数据操作也简单，Redis中的数据结构是专门进行设计的；

3.采用单线程，不存在多进程的切换而消耗cpu，不用去考虑各种锁的问题. 不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4.使用异步非阻塞IO；(重点说)

[备注:

同步/异步:是否主动读写数据;阻塞/非阻塞:是否需要等待.

**同步：执行一个操作之后，等待结果，然后才继续执行后续的操作。**

**异步：执行一个操作后，可以去执行其他的操作，然后等待通知再回来执行刚才没执行完的操作。**

**阻塞：进程给CPU传达一个任务之后，一直等待CPU处理完成，然后才执行后面的操作。**

**非阻塞：进程给CPU传达任务后，继续处理后续的操作，隔断时间再来询问之前的操作是否完成。这样的过程其实也叫轮询。**

1. 底层自己构建了VM机制，因为一般系统调用系统函数的话会浪费一定的时间去移动和请求。

# 什么是 angularjs?什么时候使用?

AngularJS简介

AngularJS 诞生于2009年，由Misko Hevery 等人创建，后为Google所收购。是一款优秀的前端JS框架，已经被用于Google的多款产品当中。AngularJS有着诸多特性，最为核心的是：MVC、模块化、自动化双向数据绑定、依赖注入等等。

MVC 模式

Angular遵循软件工程的MVC模式,好处是实现程序松耦合.通过依赖注入（dependency injection），Angular为客户端的Web应用带来了传统服务端的服务，例如独立于视图的控制。 因此，后端减少了许多负担，产生了更轻的Web应用。

双向绑定

AngularJS是建立在这样的信念上的：即声明式编程应该用于构建用户界面以及编写软件构建，而指令式编程非常适合来表示业务逻辑。框架采用并扩展了传统HTML，通过双向的数据绑定来适应动态内容，双向的数据绑定允许模型和视图之间的自动同步。因此，AngularJS使得对DOM的操作不再重要并提升了可测试性。

model模型数据 view视图数据 模型数据发生变化，视图数据随之变化 视图数据发生变化，模型数据随之变化

依赖注入

依赖注入(Dependency Injection,简称DI)是一种设计模式, 指某个对象依赖的其他对象无需手工创建，只需要“吼一嗓子”，则此对象在创建时，其依赖的对象由框架来自动创建并注入进来,其实就是最少知识法则;模块中所有的service和provider两类对象，都可以根据形参名称实现DI. 好处是实现程序的解耦合,提高代码复用性

模块化开发

高内聚低耦合法则

​ 高内聚：模块内部是一个统一整体，只负责当前功能相关业务 低耦合：模块与模块之间没有关联 (比如user用户信息的增删改查, order订单信息的增删改查. 它们之间没有关系. 除非你给它们依赖注入.)

​ 作用 : 定义angularjs作用范围.

## Angularjs如何使用的？（在品优购-day02）

1. 在前端页面引入angularjs相关的.js文件。
2. 定义模块名，控制器名，方法名。
3. 初始化模块和控制器，绑定方法。
4. 填写请求方式，处理响应结果。
5. 在前端代码中，修改相关变量。就可以查看输出结果。

# 单点登录系统CAS

### 是什么

CAS是耶鲁大学开发的单点登录产品。

SSO 仅仅是一种架构，一种设计，而 CAS 则是实现 SSO 的一种手段。

### 介绍\_单点登录系统介绍/什么是单点登录

单点登录是相互信任的系统模块登录一个模块后，其他模块不用重复登录即认证通过。

**原理:**

原理是不再使用tomcat提供的Session，而是使用redis来模拟Session，一旦用户登录后会生成一个token，类似于jsessionid的功能。将用户信息访问redis，key就是token，value就是用户信息。然后把token信息写入cookie中，并且实现cookie可以跨域访问。当需要对用户身份进行验证时，需要先从cookie中把token取出，然后根据token查询redis，来判断用户的登录状态。

### 优点和缺点（为什么使用单点登录？）

**优点:**

1）提高用户的效率。

用户不再被多次登录困扰，也不需要记住多个 ID 和密码。另外，用户忘记密码并求助于支持人员的情况也会减少。

2）提高开发人员的效率。

SSO 为开发人员提供了一个通用的身份验证框架。实际上，如果 SSO 机制是独立的，那么开发人员就完全不需要为身份验证操心。他们可以假设，只要对应用程序的请求附带一个用户名，身份验证就已经完成了。

3）简化管理。

如果应用程序加入了单点登录协议，管理用户帐号的负担就会减轻。简化的程度取决于应用程序，因为 SSO 只处理身份验证。所以，应用程序可能仍然需要设置用户的属性（比如访问特权）。

**缺点:**

1）不利于重构

因为涉及到的系统很多，要重构必须要兼容所有的系统，可能很耗时。

2） 无人看守桌面

因为只需要登录一次，所有的授权的应用系统都可以访问，可能导致一些很重要的信息泄露。

### 单点登录怎么做的？

当会员需要登录的时候，首先进入的是“登录页面”，输入用户名、密码、验证码进行登录，进入单点登录系统，用户信息被提交到单点登录系统进行校验。

如果登陆信息不正确，校验错误，将错误的登陆信息进行回显返回给用户，重新登录。验证通过，就将用户信息保存到 redis 中，并且生成 token 作为一个标识，并将 token 保存到 cookie 里面，发送给用户浏览器，下次请求的时候就会带回来，获取 cookie 里面的 token，根据 token 再去 redis 中查询对应的用户信息。登录成功后将重定向到登录之前的访问页面。

### Cookie\_如果cookie被禁用了怎么办

目前很多互联网公司都在使用cookie ,一旦cookie被禁用了，很多功能是用了不的。

回答：

1. 经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。
2. 还有一种技术叫做**表单隐藏字段**。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器

### Cookie\_如果拿到cookie中ticket是不是就能登录了

不可以。

因为ticket即使拿到了也不一定说明用户已经登录，也有可能此ticket已经过期。

（备注：这里问的ticket和token是一个意思）

### 登录的有效期有设置的吗?

设置为7天.

因为如果不设置期限，只要不清浏览器缓存，永远不用再验证，很不安全.

[备注: 使用redis来模拟session. 用户登录成功产生token（也就是上边的ticket） 设置在cookie中，并在redis中设置该用户的过期时间。]

### 如果一个用户在两台电脑登录，会有什么问题？怎么解决

回答：

出现的问题: 会存在同一个用户登录两次，存在两个token的情况。

解决: 应当在用户以同一个账号进行登录时，可以登录成功，并把上一次登录的信息清除掉，即强制退出。也就是说，数据库中只存放最后登录的账号信息。

### 单点登录系统是用什么框架做的？没有用框架怎么做

答：

本项目的单点登录系统用 来自 Yale University 的 CAS框架来做的.

不用框架怎么做:

1.最简单的单点登录实现方式，是使用cookie作为媒介，存放用户凭证。

但是存在问题:

1. Cookie不安全
2. 不能跨域实现免登

2.是通过父应用和子应用来回重定向中进行通信，实现信息的安全传递。

存在问题:不方便.

# 消息中间件解决方案JMS(ActiveMQ)

JMS，是一个Java平台中关于面向消息中间件的api，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信 。

## 什么是消息中间件

答: 关注于数据的发送和接收，利用高效可靠的异步消息传递机制集成分布式系统。

## 消息中间件作用？

模块之间的依赖称之为耦合。而耦合越多，之后的维护工作就越困难。我们通过引入消息中间件activeMQ,使得运营商系统与搜索服务、页面生成服务解除了耦合。

## JMS消息传递类型

对于消息的传递有两种类型：

一种是点对点的，即一个生产者和一个消费者一一对应；

点对点的模式主要建立在一个队列上面，当连接一个列队的时候，发送端不需要知道接收端是否正在接收，可以直接向ActiveMQ发送消息，发送的消息，将会先进入队列中，如果有接收端在监听，则会发向接收端，如果没有接收端接收，则会保存在activemq服务器，直到接收端接收消息，点对点的消息模式可以有多个发送端，多个接收端，但是一条消息，只会被一个接收端给接收到，哪个接收端先连上ActiveMQ，则会先接收到，而后来的接收端则接收不到那条消息。

(导入Solr索引库,用的是点对点,因为导入,只需做一次就够了.)

另一种是发布/ 订阅模式，即一个生产者产生消息并进行发送后，可以由多个消费者进

行接收。

(网页静态化,用的是发布订阅模式,需要每个服务器都执行.)

## MQ、Dubbo 区别以及使用场景

MQ：消息队列。生产者消费者模式，可用于对消息实时性要求不高的场景。多程序之间间接调用关系

Dubbo：RPC实现。多程序之间直接调用关系

[RPC:](http://dict.youdao.com/search?q=RPC&keyfrom=chrome.extension&le=eng" \t "_blank) 远程过程调用(Remote Procedure Call)

## ActiveMQ的优缺点

优点:

1. 消息不丢失：服务器挂掉,消息会保存到mq中间件中，当消费者服务器恢复后就会重新发过去，消息不会丢失.  
   2.异步处理：比如一个商城用户购买产品后,后台会去更新数据库,然后响应给客户端.如果在高并发的情况下，这样更新数据库响应客户端会变慢，可以使用mq消息队列的消费者进程中获取数据来进行异步写数据，由于消息对垒的服务处理速度远快于数据库，因此响应延迟能得到有效改善.

## **哪些情况用到 activeMq？**

商品上架后更新 solr 索引库、更新静态页、发送短信

## 消息中间件acitveMQ的作用、原理

Activemq的作用就是系统之间进行通信。当然可以使用其他方式进行系统间通信，如果使用Activemq的话可以对系统之间的调用进行解耦，实现系统间的异步通信。

原理就是生产者生产消息，把消息发送给activemq。Activemq接收到消息，然后查看有多少个消费者，然后把消息转发给消费者，此过程中生产者无需参与。消费者接收到消息后做相应的处理和生产者没有任何关系.

## **ActiveMq** 消息**被重复消费，丢失，或者不消费怎么办**

回答：

重复消费：如何防止消息重复发送？

解决方法很简单：增加消息状态表。通俗来说就是一个账本，用来记录消息的处理状态，每次处理消息之前，都去状态表中查询一次，如果已经有相同的消息存在，那么不处理，可以防止重复发送。

丢消息：用持久化消息，或者非持久化消息及时处理不要堆积，或者启动事务，启动事务后，commit()

方法会负责任的等待服务器的返回，也就不会关闭连接导致消息丢失了。

不消费：去 ActiveMQ.DLQ 里找找

## **activeMQ 的消息持久化问题？**

A：持久化为文件

这个你装 ActiveMQ 时默认就是这种，只要你设置消息为持久化就可以了。

B：持久化为 MySql

加载驱动 jar，为数据中创建三个数据库表，存储 activemq 的消息信息

## MQ发送消息失败怎么办（90%）

可以设置重试机制，如果重试多次会出现重复消息，只需要保证数据的最终一致性即可，

## activemq如何确认消息发送成功

activemq通过应答模式来确认是否发送或者消息消费成功。

## ActiveMQ 如果消息发送失败怎么办？

Activemq 有两种通信方式，点到点形式和发布订阅模式。

如果是点到点模式的话，如果消息发送不成功此 消息默认会保存到 activemq 服务端知道有消费者将其消费，所以此时消息是不会丢失的。

如果是发布订阅模式的通信方式，默认情况下只通知一次，如果接收不到此消息就没有了。这种场景只适 用于对消息送达率要求不高的情况。如果要求消息必须送达不可以丢失的话，需要配置持久订阅。每个订阅端定义一个 id，在订阅是向 activemq 注册。发布消息和接收消息时需要配置发送模式为持久化。此时 如果客户端接收不到消息，消息会持久化到服务端，直到客户端正常接收后为止。

## activemq怎么操作的？

<http://192.168.25.128:8161> 即可进入 ActiveMQ 管理页面

1. 下载并安装好avtivemq。输入启动指令启动它。
2. 创建工程架构，引入SpringJms，activeMQ的jar包。
3. 编写生产者类，消费者类，监听类。在配置文件中要配置消息监听容器。
4. 查看控制器，发现消息创建了，也消费了。

## Hystrix 解释一下怎么处理服务雪崩问题?

分布式系统中经常会出现某个基础服务不可用造成整个系统不可用的情况, 这种现象被称为服务雪崩效应.

使用Hystrix预防服务雪崩

Hystrix [hɪst'rɪks]的中文含义是豪猪, 因其背上长满了刺,而拥有自我保护能力. Netflix的 Hystrix 是一个帮助解决分布式系统交互时超时处理和容错的类库, 它同样拥有保护系统的能力.

资源隔离:

Hystrix通过将每个依赖服务分配独立的线程池进行资源隔离, 从而避免服务雪崩.  

熔断器:

熔断器的开关能保证服务调用者在调用异常服务时, 快速返回结果, 避免大量的同步等待. 并且熔断器能在一段时间后继续侦测请求执行结果, 提供恢复服务调用的可能.

命令模式:

Hystrix使用命令模式(继承HystrixCommand类)来包裹具体的服务调用逻辑(run方法), 并在命令模式中添加了服务调用失败后的降级逻辑(getFallback).

## SpringMVC的视图解析流程

1、调用目标方法，SpringMVC将目标方法返回的String、View、ModelMap或是ModelAndView都转换为一个ModelAndView对象；

2、然后通过视图解析器（ViewResolver）对ModelAndView对象中的View对象进行解析，将该逻辑视图View对象解析为一个物理视图View对象；

3、最后调用物理视图View对象的render()方法进行视图渲染，得到响应结果。

## SpringMVC中Json的解析？

在springMVC-servlet中添加<mvc:annotation-driven />注解使JSON字符串自动转换成实体类。

## 如何解决 get 和 post 乱码问题？

解决 post 请求乱码:我们可以在 web.xml 里边配置一个 CharacterEncodingFilter 过滤器。 设置为 utf-8.

对于 get 请求中文参数出现乱码解决方法有两个:

1. 修改 tomcat 配置文件添加编码与工程编码一致。

2. 另 外 一 种 方 法 对 参 数 进 行 重 新 编 码 String userName = New

String(Request.getParameter(“userName”).getBytes(“ISO8859-1”), “utf-8”);

## SpringMVC和struts2比较

| **对比项目** | **SrpingMVC** | **Struts2** | **优势** |
| --- | --- | --- | --- |
| 国内市场情况 | 有大量用户，一般新项目启动都会选用springmvc | 有部分老用户，老项目组，由于习惯了，一直在使用。 | 国内情况，springmvc的使用率已经超过Struts2 |
| 框架入口 | 基于servlet | 基于filter | 本质上没太大优势之分，只是配置方式不一样 |
| 框架设计思想 | 控制器基于方法级别的拦截，处理器设计为单实例 | 控制器基于类级别的拦截， 处理器设计为多实例 | 由于设计本身原因，造成了Struts2，通常来讲只能设计为多实例模式，相比于springmvc设计为单实例模式，Struts2会消耗更多的服务器内存。 |
| 参数传递 | 参数通过方法入参传递 | 参数通过类的成员变量传递 | Struts2通过成员变量传递参数，导致了参数线程不安全，有可能引发并发的问题。 |
| 与spring整合 | 与spring同一家公司，可以与spring无缝整合 | 需要整合包 | Springmvc可以更轻松与spring整合 |

## **Springclound 主要组件以及说明？**

Eureka和Ribbon，是最基础的组件，一个注册服务，一个消费服务。

Hystrix为了优化Ribbon、防止整个微服务架构因为某个服务节点的问题导致崩溃，是个保险丝的作用。

Dashboard给Hystrix统计和展示用的，而且监控服务节点的整体压力和健康情况。

Turbine是集群收集器，服务于Dashboard的。

Feign是方便我们程序员写更优美的代码的。

Zuul是加在整个微服务最前沿的防火墙和代理器，隐藏微服务结点IP端口信息，加强安全保护的。

Config是为了解决所有微服务各自维护各自的配置，设置一个统一的配置中心，方便修改配置的。

Bus是因为config修改完配置后各个结点都要refresh才能生效实在太麻烦，所以交给bus来通知服务节点刷新配置的。

Stream是为了简化研发人员对MQ使用的复杂度，弱化MQ的差异性，达到程序和MQ松耦合。

Sleuth是因为单次请求在微服务节点中跳转无法追溯，解决任务链日志追踪问题的。

## 流程\_dubbo服务开发流程(运行流程/使用方法)

使用流程：

第一步：要在系统中使用dubbo应该先搭建一个注册中心，一般推荐使用zookeeper。

第二步：有了注册中心然后是发布服务，发布服务需要使用spring容器和dubbo标签来发布服务。并且发布服务时需要指定注册中心的链接地址。

第三步：服务发布之后就是调用服务。一般调用服务也是使用spring容器和dubbo标签来引用服务，这样就可以在客户端的容器中生成一个服务的代理对象，在Controller中直接调用service的方法即可。

## **多线程下，如何优雅的保证线程安全和性能？**

对以上各个锁进行1000000次的加锁解锁的空操作时间如下：

OSSpinLock: 46.15 ms  
dispatch\_semaphore: 56.50 ms  
pthread\_mutex: 178.28 ms  
NSCondition: 193.38 ms  
NSLock: 175.02 ms  
pthread\_mutex(recursive): 172.56 ms  
NSRecursiveLock: 157.44 ms  
NSConditionLock: 490.04 ms  
@synchronized: 371.17 ms

总的来说：

OSSpinLock和dispatch\_semaphore的效率远远高于其他。

@synchronized和NSConditionLock效率较差。

鉴于OSSpinLock的不安全，所以我们在开发中如果考虑性能的话，建议使用dispatch\_semaphore。

如果不考虑性能，只是图个方便的话，那就使用@synchronized

# 第二项目相关面试题

## 人员分配(14人)

* + - * 产品经理：1 人，确定需求以及给出产品原型图。
      * 项目经理：1 人，项目管理。
      * 前端团队：2 人，根据产品经理给出的原型制作静态页面。
      * 后端团队：6 人，实现产品功能。
      * 测试团队：2 人，测试所有的功能。
      * 运维团队：2 人，项目的发布以及维护。

## 项目描述(介绍)

**答:**

**我们做的是叫品书易的购物网站, 采用前后端分离的技术架构， 以json数据格式进行交互.后端框架采用SpringMVC+Spring +mybatis +Dubbox 。前端采用angularJS + Bootstrap。**

**采用分布式的架构设计,包括后台管理、前台系统、搜索系统、订单系统、单点登录系统、会员系统等。**

**我主要负责 :后台管理、前台系统、搜索系统，单点登录系统。**

**搜索用solr, freemarker 做静态化页面，用 nginx 做负载均衡.用 redis 做缓存，使用 Activemq 做消息中间件,后台数据库使用mysql 数据库. CAS做单点登录.**

##### **1.5 品书易业务简介**

​ 品书易购物网站是一个综合性 平台，类似京东商城、天猫商城。网站采用商家入驻的模式，商家入驻平台提交申请，有平台进行资质审核，审核通过后，商家拥有独立的管理后台录入商品信息。商品经过平台审核后即可发布。

​ 品书易购物网站主要分为网站前台、运营商后台、商家管理后台三个子系统。

​

3.1.1 网站前台

主要包括网站首页、商家首页、商品详细页、、搜索页、会员中心、订单与支付相关页面、秒杀频道等。

3.1.2 运营商后台

是运营商的运营人员的管理后台。 主要包括商家审核、品牌管理、规格管理、模板管理、商品分类管理、商品审核、广告类型管理、广告管理、订单查询、商家结算等。

3.1.3 商家管理后台

入驻的商家进行管理的后台，主要功能是对商品的管理以及订单查询统计、资金结算等功能。

