JAVAEE面试题精选101

1. JAVA基础
2. **==和equals方法究竟有什么区别？**

|  |
| --- |
| 1). equals方法比较两个对象的内容  2). ==比较两个对象是否是同一对象，或者说对象在内存中的首地址是否相同  应用：(1) 对于基本数据类型，“==”比较的是两者的值是否相等  (2) 对于引用数据类型，==”比较的是引用的地址是否相同，对象默认的equals方法和==一样  (3) 对于String,Integer等包装类都重写了equals方法，比较的是内容是否相同 |

1. **静态变量和实例变量的区别？**

|  |
| --- |
| 1). 静态变量属于类，该类不生产对象，通过类名就可以调用静态变量。 2). 实例变量属于该类的对象，必须产生该类对象，才能调用实例变量。 |

1. **Overload和Override的区别？**

|  |
| --- |
| java中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。与此相对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型。覆盖者可能不会限制它所覆盖的方法的访问。 |

1. **abstract class和interface有什么区别？**

|  |
| --- |
| java提供和支持创建抽象类和接口。它们的实现有共同点，不同点在于：  接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。  类可以实现很多个接口，但是只能继承一个抽象类  类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法， 当然，在这种情况下，类也必须得声明成是抽象的。  抽象类可以在不提供接口方法实现的情况下实现接口。  Java接口中声明的变量默认都是final的。抽象类可以包含非final的变量。  Java接口中的成员函数默认是public的。抽象类的成员函数可以是private，protected或者是public。  接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含main方法的话是可以被调用的 |

1. **String,StringBuffer与StringBuilder的区别？**

|  |
| --- |
| String是字符串常量,final修饰;StringBuffer字符串变量(线程安全); StringBuilder 字符串变量(线程不安全).  String和StringBuffer  String和StringBuffer主要区别是性能:String是不可变对象,每次对String类型进行操作都等同于产生了一个新的String对象,然后指向新的String对象.所以尽量不在对String进行大量的拼接操作,否则会产生很多临时对象,导致GC开始工作,影响系统性能.  StringBuffer是对对象本身操作,而不是产生新的对象,因此在通常在有大量拼接的情况下我们建议使用StringBuffer.  但是需要注意现在JVM会对String拼接做一定的优化: String s=“This is only ”+”simple”+”test”会被虚拟机直接优化成String s=“This is only simple test”,此时就不存在拼接过程.  StringBuffer和StringBuilder  StringBuffer是线程安全的可变字符串,其内部实现是可变数组.StringBuilder是java 5.0新增的,其功能和StringBuffer类似,但是非线程安全.因此,在没有多线程问题的前提下,使用StringBuilder会取得更好的性能. |

1. **运行时异常与一般异常有何异同？**

|  |
| --- |
| 异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常 |

1. **深拷贝和浅拷贝的区别是什么？**

|  |
| --- |
| 浅拷贝：被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。换言之，浅拷贝仅仅复制所考虑的对象，而不复制它所引用的对象。  深拷贝：被复制对象的所有变量都含有与原来的对象相同的值，而那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。换言之，深拷贝把要复制的对象所引用的对象都复制了一遍。 |

1. **equals()和hashcode()的联系？**

|  |
| --- |
| hashCode()是Object类的一个方法,返回一个哈希值.如果两个对象根据equal()方法比较相等,那么调用这两个对象中任意一个对象的hashCode()方法必须产生相同的哈希值. 如果两个对象根据eqaul()方法比较不相等,那么产生的哈希值不一定相等(碰撞的情况下还是会相等的.) |

1. **conllection和conllections的区别？**

|  |
| --- |
| Collection是集合类的上级接口，继承与他的接口主要有Set 和List.  Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。 |

1. **list和set的区别？**

|  |
| --- |
| (1). List,Set都是继承自Collection接口  (2). List特点：元素有放入顺序，元素可重复 ，Set特点：元素无放入顺序，元素不可重复（注意：元素虽然无放入顺序，但是元素在set中的位置是有该元素的HashCode决定的，其位置其实是固定的）  (3). List接口有三个实现类：LinkedList，ArrayList，Vector ，Set接口有两个实现类：HashSet(底层由HashMap实现)，LinkedHashSet |

1. **ArrayList和Vector的区别？**

|  |
| --- |
| 1). ArrayList在内存不够时默认是扩展50% + 1个，Vector是默认扩展1倍。 2). Vector提供indexOf(obj, start)接口，ArrayList没有。 3). Vector属于线程安全级别的，但是大多数情况下不使用Vector，因为线程安全需要更大的系统开销。 |

1. **ArrayList,Vector, LinkedList的存储性能和特性？**

|  |
| --- |
| ArrayList 和Vector是采用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接序号索引元素，但是插入数据要设计到数组元素移动等内存操作，所以索引数据快插入数据慢，Vector由于使用了synchronized方法（线程安全）所以性能上比ArrayList要差，LinkedList使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但是插入数据时只需要记录本项的前后项即可，所以插入数度较快！ |

1. **HashMap和Hashtable的区别？**

|  |
| --- |
| 1).hashTable同步的，而HashMap是非同步的，效率上比hashTable要高。 2).hashMap允许空[键值](https://www.baidu.com/s?wd=%E9%94%AE%E5%80%BC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnHf1nhcdmvDYuWc3Pvm10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnH6vnHTYPHbLn1bsnjbdPW0vPs)，而hashTable不允许。  3).hashMap去掉了HashTable 的contains方法，但是加上了containsValue（）和containsKey（）方法。 |

1. **set集合如何保证不重复？**

|  |
| --- |
| 主要是要判断set集合中是否有重复的元素，规则如下：  1)、判断两个对象的hashCode是否相等 。       如果不相等，认为两个对象也不相等，完毕        如果相等，转入2) （这一点只是为了提高存储效率而要求的，其实理论上没有也可以，但如果没有，实际使用时效率会大大降低，所以我们这里将其做为必需的。后面会重点讲到这个问题。）  2)、判断两个对象用equals运算是否相等 。       如果不相等，认为两个对象也不相等        如果相等，认为两个对象相等（equals()是判断两个对象是否相等的关键） |

1. **HashMap的实现原理？**

|  |
| --- |
| HashMap基于hash原理，我们通过put()和get()方法储存和获取对象。当我们将键值对传递给put()方法时，它调用键对象的hashCode()方法来计算hashcode，让后找到bucket位置来储存值对象。当获取对象时，通过键对象的equals()方法找到正确的键值对，然后返回值对象。HashMap使用LinkedList来解决碰撞问题，当发生碰撞了，对象将会储存在LinkedList的下一个节点中。 HashMap在每个LinkedList节点中储存键值对对象。  当两个不同的键对象的hashcode相同时会发生什么？ 它们会储存在同一个bucket位置的LinkedList中。键对象的equals()方法用来找到键值对 |

1. **TreeSet的原理以及使用？**

|  |
| --- |
| 1). TreeSet()是使用二叉树的原理对新add()的对象按照指定的顺序排序（升序、降序），每增加一个对象都会进行排序，将对象插入的二叉树指定的位置。  2). Integer和String对象都可以进行默认的TreeSet排序，而自定义类的对象是不可以的，自己定义的类必须实现Comparable接口，并且覆写相应的compareTo()函数，才可以正常使用。 |

1. **进程和线程的区别？**

|  |
| --- |
| 一个进程是一个独立(self contained)的运行环境，它可以被看作一个程序或者一个应用。而线程是在进程中执行的一个任务。Java运行环境是一个包含了不同的类和程序的单一进程。线程可以被称为轻量级进程。线程需要较少的资源来创建和驻留在进程中，并且可以共享进程中的资源 |

1. **创建线程的方式？**

|  |
| --- |
| 进程是执行着的应用程序，而线程是进程内部的一个执行序列。一个进程可以有多个线程。线程又叫做轻量级进程 |

1. **线程有哪些状态？**

|  |
| --- |
| 线程在执行过程中，可以处于下面几种状态：  就绪(Runnable):线程准备运行，不一定立马就能开始执行。  运行中(Running)：进程正在执行线程的代码。  等待中(Waiting):线程处于阻塞的状态，等待外部的处理结束。  睡眠中(Sleeping)：线程被强制睡眠。  I/O阻塞(Blocked on I/O)：等待I/O操作完成。  同步阻塞(Blocked on Synchronization)：等待获取锁。  死亡(Dead)：线程完成了执行。 |

1. **线程同步的方式有哪些？**

|  |
| --- |
| 1). synchronized关键字修饰的方法  2). synchronized关键字修饰的语句块  3). 使用重入锁Lock类实现线程同步  4). 使用ThreadLocal管理变量实现线程同步  5.) 使用阻塞队列实现线程同步  6). 使用原子变量实现线程同步 |

1. **如何保证几个线程线程有序执行？**

|  |
| --- |
| 1). 共享对象锁，可以保证每个方法只能同时有一个线程进入，配合wait和notifyall方法，可以启动或者唤醒线程  2). 通过主线程Join() |

1. **java中wait和sleep方法有何不同？**

|  |
| --- |
| 1). 这两个方法来自不同的类分别是Thread和Object  2). 最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法  3). wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用  4). sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常 |

1. **线程之间是如何通信的？**

|  |
| --- |
| 当线程间是可以共享资源时，线程间通信是协调它们的重要的手段。Object类中wait()\notify()\notifyAll()方法可以用于线程间通信关于资源的锁的状态 |

1. **同步方法和同步块，哪个是更好的选择？**

|  |
| --- |
| 同步块是更好的选择，因为它不会锁住整个对象（当然也可以让它锁住整个对象）。同步方法会锁住整个对象，哪怕这个类中有多个不相关联的同步块，这通常会导致他们停止执行并需要等待获得这个对象上的锁 |

1. **线程中同步和异步的区别？**

|  |
| --- |
| 同步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为同步机制存在，A线程请求不到，怎么办，A线程只能等待下去  异步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为没有同步机制存在，A线程仍然请求的到，A线程无需等待 |

1. **简述Volatile变量？**

|  |
| --- |
| 用来确保将变量的更新操作通知到其他线程,保证了新值能立即同步到主内存,以及每次使用前立即从主内存刷新。换句话说:volatile 变量在各个线程中是一致的 |

1. **简述线程局部变量ThreadLocal？**

|  |
| --- |
| ThreadLocal变量线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享，是一种实现线程安全的方式 |

1. **简述Java多线程中死锁、活锁、饥饿？**

|  |
| --- |
| 死锁：死锁发生在一个线程需要获取多个资源的时候，这时由于两个线程互相等待对方的资源而被阻塞  活锁：的是线程不断重复执行相同的操作，但每次操作的结果都是失败的。尽管这个问题不会阻塞线程，但是程序也无法继续执行  饥饿:指的线程无法访问到它需要的资源而不能继续执行时，引发饥饿最常见资源就是CPU时钟周期 |

1. **简述Java多线程的悲观锁与乐观锁？**

|  |
| --- |
| 悲观锁:就是不管是否发生多线程冲突，只要存在这种可能，就每次访问都加锁，加锁就会导致锁之间的争夺，有争夺就会有输赢，输者等待。   乐观锁:获得锁后一直持有锁以防本线程再次申请该锁造成无谓的解锁再加锁开销 |

1. **简述java垃圾回收机制？**

|  |
| --- |
| Java 垃圾回收机制最基本的做法是分代回收。内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中。一般的实现是划分成3个世代：年轻、年老和永久。内存的分配是发生在年轻世代中的。当一个对象存活时间足够长的时候，它就会被复制到年老世代中。对于不同的世代可以使用不同的垃圾回收算法。进行世代划分的出发点是对应用中对象存活时间进行研究之后得出的统计规律。一般来说，一个应用中的大部分对象的存活时间都很短。比如局部变量的存活时间就只在方法的执行过程中。基于这一点，对于年轻世代的垃圾回收算法就可以很有针对性. |

1. **如何判断一个对象是否应该被回收？**

|  |
| --- |
| 这就是所谓的对象存活性判断,常用的方法有两种:1.引用计数法;2:对象可达性分析.由于引用计数法存在互相引用导致无法进行GC的问题,所以目前JVM虚拟机多使用对象可达性分析算法. |

1. **简述JVM内存分配？**

|  |
| --- |
| (1). 基本数据类型比变量和对象的引用都是在栈分配的  (2). 堆内存用来存放由new创建的对象和数组  (3). 类变量（static修饰的变量），程序在一加载的时候就在堆中为类变量分配内存，堆中的内存地址存放在栈中  (4). 实例变量：当你使用java关键字new的时候，系统在堆中开辟并不一定是连续的空间分配给变量，是根据零散的堆内存地址，通过哈希算法换算为一长串数字以表征这个变量在堆中的"物理位置”,实例变量的生命周期--当实例变量的引用丢失后，将被GC（垃圾回收器）列入可回收“名单”中，但并不是马上就释放堆中内存  (5). 局部变量: 由声明在某方法，或某代码段里（比如for循环），执行到它的时候在栈中开辟内存，当局部变量一但脱离作用域，内存立即释放 |

1. **Java中常用的垃圾收集算法？**

|  |
| --- |
| (1). 标记-清除  (2). 标记-复制  (3). 标记-整理  (4). 分代回收  参考网站：<http://www.cnblogs.com/cielosun/p/6674431.html>  最好能简单的描述一下，当然描述2种即可（自行百度） |

1. **System.gc()和Runtime.gc()的作用？**

|  |
| --- |
| 这两个方法用来提示JVM要进行垃圾回收。但是，立即开始还是延迟进行垃圾回收是取决于JVM的 |

1. **finalize()方法什么时候被调用以及作用？**

|  |
| --- |
| 在释放对象占用的内存之前，垃圾收集器会调用对象的finalize()方法。一般建议在该方法中释放对象持有的资源。 |

1. **简述java类加载机制以及加载过程？**

|  |
| --- |
| 类从被加载到虚拟机内存中开始，直到卸载出内存为止，它的整个生命周期包括了： 加载、验证、准备、解析、初始化、使用和卸载 这7个阶段。其中， 验证、准备和解析这三个部分统称为连接（linking）  加载：加载阶段由类加载器在完成，主要作用如下：   1. 获取定义此类的二进制字节流 2. 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构 3. 在Java堆中生成一个代表这个类的java.lang.Class对象   验证： 验证是连接阶段的第一步，这一阶段的目的是为了确保Class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全  准备：准备阶段是为类的静态变量分配内存并将其初始化为默认值，这些内存都将在方法区中进行分配  解析： 解析阶段是虚拟机将常量池内的符号引用替换为直接引用的过程  初始化：初始化阶段是执行类构造器<clinit>()方法的过程 |

1. **什么是类加载器，类加载器有哪些？**

|  |
| --- |
| 类加载器用于完成类的加载。类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader的子类） |

1. **能说出至少3到5种设计模式？**

|  |
| --- |
| 参考网站：<http://www.cnblogs.com/maowang1991/archive/2013/04/15/3023236.html> |

1. **能说出3种排序算法的思想？**

|  |
| --- |
| 冒泡排序：<http://www.cnblogs.com/jingmoxukong/p/4302718.html>  快速排序：<http://www.cnblogs.com/jingmoxukong/p/4302891.html>  直接插入排序：<http://www.cnblogs.com/jingmoxukong/p/4303270.html>  希尔排序：<http://www.cnblogs.com/jingmoxukong/p/4303279.html>  简单选择排序：<http://www.cnblogs.com/jingmoxukong/p/4303289.html> |

1. Java Web
2. **简述http协议？**

|  |
| --- |
| HTTP（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式。http请求由三部分组成，分别是：请求行、消息报头、请求正文，响应也包含三部分：状态行（包含HTTP版本、状态码、状态码的原因短语）、响应首部字段、响应内容实体 |

1. **Get和Post请求方式的区别？**

|  |
| --- |
| (1). get地址栏有参数显示 post不会再地址栏显示参数(参数是放在了请求体）  (2). get不安全 post相对安全  (3). get限制大小 post理论上不限制 |

1. **Servlet的生命周期？**

|  |
| --- |
| 1). 第一次请求的时候servlet被初始化，而且只初始化一次，所以tomcat容器中每一个servlet只有一个对象存在  2). 初始化后先调用init方法，只执行一遍  3). 每个请求，调用一遍service -> service -> doGet/doPost。以多线程的方式运行  4). 卸载前调用destroy方法 |

1. **JSP和Servlet有哪些相同点和不同点，他们之间的联系是什么？**

|  |
| --- |
| JSP是Servlet技术的扩展，本质上就是Servlet的简易方式。JSP编译后是“类servlet”。Servlet和JSP最主要的不同点在于，Servlet的应用逻辑是在Java文件中，并且完全从表示层中的HTML里分离开来。而JSP的情况是Java和HTML可以组合成一个扩展名为.jsp的文件。JSP侧重于视图，Servlet主要用于控制逻辑 |

1. **重定向和转发的区别？**

|  |
| --- |
| 1). 重定向是客户端行为，转发是服务器端行为  2). 重定向两次请求两次响应,转发一次请求一次响应  3). 重定向路径需要加工程名,转发的路径不需要加工程名.  4). 重定向可以跳转到任意网站,转发只能在服务器内部进行转发.  5). 重定向会导致request对象信息丢失。转发则不会 |

1. **述JSP九大内置对象？**

|  |
| --- |
| request：负责得到客户端请求的信息，对应类型：javax.servlet.http.HttpServletRequest  response:负责向客户端发出响应，对应类型：javax.servlet.http.HttpServletResponse  session:负责保存同一客户端一次会话过程中的一些信息，对应类型：javax.servlet.http.httpsession  out：负责管理对客户端的输出，对应类型：javax.serlvet.jsp.jspwriter  application:表示整个应用环境的信息，对应类型：javax.servlet.servletcontext  config:表示ServletConfig，对应类型：javax.servlet.servletconfig  exception:表示页面中发生的异常，可以通过它获得页面异常信息，对应类型：java.lang.exception  pagecontext:表示这个JSP页面上下文，对应类型：javax.servlet.jsp.pagecontext  page:表示当前JSP页面本身 |

1. **JSP的四种作用域？**

|  |
| --- |
| 1). page是代表一个页面相关的对象和属性。  2). request一个请求可能跨越多个页面。  3). session一个web回话也可以经常跨域多个请求。  4). application是代表与整个web应用程序相关的对象和属性。 |

1. **JSP中静态include与动态include的区别？**

|  |
| --- |
| 1). 静态include：语法：<%@ include file="文件名" %>，相当于复制，编辑时将对应的文件包含进来，当内容变化时，不会再一次对其编译，不易维护。  2). 动态include：语法：<jsp:include page="文件名">,能够自动检查被包含文件，当客户端对JSP文件进行请求时，会重新将对应的文件包含进来，进行实时的更新。 |

1. **cookie和session的作用、区别？**

|  |
| --- |
| 1). cookie数据保存在客户端，session数据保存在服务端。  2). cookie不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗，相当重要的数据，应该使用session保存到服务端。  3). session会在一定时间内保持在服务器上，但是会占用内存资源，当访问的用户过多，会加重服务器的负载，考虑到减轻服务器的压力，可以将不重要的数据放在cookie中持久的保存。  4). 单个cookie保存的数据不能超过4k，很多浏览器都限制站点最多保存20个cookie。 |

1. **tomcat的优化经验？**

|  |
| --- |
| 1). 去掉对web.xml的监视  2). 把JSP提前编辑成Servlet  3). 有富余物理内存的情况下  4). 加大Tomcat使用的JVM内存。 |

1. **如何防止表单重复提交？**

|  |
| --- |
| 1). JavaScript控制  2). 使用session实现令牌机制 |

1. **过滤器有哪些作用，以及过滤器的生命周期？**

|  |
| --- |
| 生命周期：每个Filter在tomcat启动时进行初始化，每个Filter只有一个实例对象  (1). Init:在服务器启动时会创建Filter实例  (2). doFilto:这个方法会在用户每次访问“目标资源”时执行  (3). destroy():服务器关闭时销毁Filter对象  作用：  (1). 验证客户是否来自可信网络  (2). 对客户提交的数据进行重新编码  (3). 过滤掉客户的某些不应该出现的词汇  (4). 验证用户是否可以登录  (5). 验证客户的浏览器是否支持当前的应用  (6). 记录系统日志 |

1. **详细描述MVC？**

|  |
| --- |
| 基于java的web应用系统采用MVC设计模型，即用Model（模型）、View（视图）和Controller（控制）分离设计，这是目前web应用服务系统的主流设置方向。        Model：处理业务逻辑的模块。        View：负责页面显示，显示Model的处理结果给用户，主要实现数据到页面的转换过程。        Controller：负责每个请求的分发，把Form数据传递给Model进行处理，处理完成后，把处理结果返回给相应的View显示给用户 |

1. 数据库
2. **sql语句查询练习？**

|  |
| --- |
| 自行参照之前发的笔记好好把sql语句的增删查改进行复习 |

1. **项目中优化sql语句执行效率的方法？**

|  |
| --- |
| （1）尽量选择较小的列  （2）将where中用的比较频繁的字段建立索引  （3）select子句中避免使用‘\*’  （4）避免在索引列上使用计算，not，in和<>等操作  （5）当只需要一行数据的时候使用limit 1  （6）保证表单数据不超过200w，适时分割表  （7）针对查询较慢的语句，可以使用explain来分析该语句具体的执行情况  （8）避免查询时判断null,否则可能会导致全表扫描,无法使用索引;  （9）避免like查询,否则可能导致全表扫描,可以考虑使用全文索引  （10）能用union all的时候就不用union,union过滤重复数据要耗费更多的CPU资源 |

1. **truncate和delete的区别？**

|  |
| --- |
| （1）delete from 表名： 删除所有记录，属于DML语句，一条记录一条记录删除。事务可以作用在DML语句上的  （2）truncate table 表名：删除所有记录，属于DDL语句，将表删除，然后重新创建一个结构一样的表。事务不能控制DDL的，推荐使用 |

1. **数据库索引的作用？缺点是什么？**

|  |
| --- |
| 创建索引可以大大提高系统的性能。  1). 通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。  2). 可以大大加快 数据的检索速度，这也是创建索引的最主要的原因。  3). 可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。  4). 在使用分组和排序 子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。  5). 通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。  缺点：  1). 创建索引和维护索引要耗费时间，这种时间随着数据 量的增加而增加。  2). 索引需要占物理空间。  3). 当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度 |

1. **MySQL数据库有哪些类型的索引？**

|  |
| --- |
| 索引类型: B-TREE索引，哈希索引  B-TREE索引加速了数据访问，因为存储引擎不会扫描整个表得到需要的数据  哈希索引建立在哈希表的基础上，它只对使用了索引中的每一列的精确查找有用 |

1. **Mysql存储引擎——MyISAM 和InnoDB区别？**

|  |
| --- |
| 1.MySQL默认采用的是MyISAM。  2.MyISAM不支持事务，而InnoDB支持。  3.InnoDB支持数据行锁定，MyISAM不支持行锁定，只支持锁定整个表。  4.InnoDB支持外键，MyISAM不支持。  5.InnoDB的主键范围更大，最大是MyISAM的2倍。  6.InnoDB不支持全文索引，而MyISAM支持。全文索引是指对char、 varchar和text中的每个词（停用词除外）建立倒排序索引。M  7.没有where的count(\*)使用MyISAM要比InnoDB快得多。 |

1. **mysql分库分表？**

|  |
| --- |
| mysql分库分表的的重点就是表的切分。表的切分可分为纵向和横向两种切分方法。  (1). 纵向切分：将不同模块的表分别存储到不同的数据库中，按模块分出来的表在代码中最好不要与其他数据库中的表有关联操作。  (2). 横向切分：横向切分是根据表内数据的逻辑关系，将同一个表按某种规则拆分到多个数据库或多个表中。 |

1. **Mysql集群实现读写分离？**

|  |
| --- |
| MySQL读写分离基本原理是让master数据库处理写操作，slave数据库处理读操作。master将写操作的变更同步到各个slave节点。  MySQLProxy是mysql集群的中间件，可以实现mysql主从集群，并且实现mysql的读写分离与负载均衡，  MySQLProxy实际上是在客户端请求与MySQL节点之间建立了一个连接池。所有客户端请求都是发向MySQLProxy，然后经由MySQLProxy进行相应的分析，判断出是读操作还是写操作，分发至对应的MySQL节点上。对于多节点Slave的集群，也可以起做到负载均衡的效果。 |

1. **mysql和oracle分页的区别？**

|  |
| --- |
| 1). MySQL 分页 (利用LIMIT关键字）  2). Oracle 分页 (利用自带的rownum) |

1. **oracle中存储过程和函数的区别？**

|  |
| --- |
|  1).函数（function）总是向调用者返回数据，并且一般只返回一个值；   2).存储过程（procedure）不直接返回数据，但可以改变输出参数的值，这可以近似看作能返回值，且存储过程输出参数的值个数没有限制  3).调用方式不同 |

1. JavaEE
2. **简述Struts的工作流程？**

|  |
| --- |
| 客户端发送请求--》请求经过一系列过滤器--》FilterDispatcher通过ActionMapper来决定这个Request需要调用哪个Action --》FilterDispatcher把请求的处理交给ActionProxy--》通过ConfigurationManager询问Struts配置文件（Struts.xml），找到需要调用的Action类--》ActionProxy创建一个ActionInvocation的实例 --》调用Action--》执行完毕，返回结果 |

1. **Strust2中值栈ValueStack的原理？**

|  |
| --- |
| ValueStack贯穿整个 Action 的生命周期，保存在request域中，所以ValueStack和request的生命周期一样。当Struts2接受一个请求时，会迅速创建ActionContext， ValueStack，action。然后把action存放进ValueStack，所以action的实例变量可以被OGNL访问。 请求开始的时候，action、ValueStack的生命开始，请求结束，action、ValueStack的生命结束 |

1. **简述struts2拦截器的工作原理？**

|  |
| --- |
| 在Struts2中，可以实现Interceptor接口或继承AbstractInterceptor类，来自定义拦截器。  ①接口中的init()方法，在拦截器被创建后立即被调用，它在拦截器的生命周期内只被调用一次，可以在该方法中对相关资源进行必要的初始化；  ②每拦截一个请求，intercept()方法就会被调用一次；  ③destory()方法将在拦截器销毁之前被调用 |

1. **谈一下Struts2的拦截器和过滤器的区别？**

|  |
| --- |
| 拦截器和过滤器都可以用来实现横切关注功能，其区别主要在于：  ①拦截器是基于JAVA反射机制的，而过滤器是基于函数回调的  ②过滤器依赖于Servlet容器，而拦截器不依赖于Servlet容器  ③拦截器只能对Action请求起作用（Action中的方法），而过滤器可以对几乎所有的请求起作用（CSS JSP JS） |

1. **Hibernate的SessionFactory是线程安全的吗？Session是线程安全的吗？两个线程能共享一个Session吗？**

|  |
| --- |
| ①SessionFactory对应Hibernate的一个数据存储的概念，它是线程安全的，可以被多个线程并发访问。SessionFactory一般只会在启动的时候构建。对于应用程序，最好将SessionFactory通过单例的模式进行封装以便于访问。  ②Session是一个轻量级非线程安全的对象（线程间不能共享Session），它表示与数据库进行交互的一个工作单元。Session是由SessionFactory创建的，在任务完成之后会被关闭。Session是持久层服务对外提供的主要接口。Session会延迟获取数据库连接（也就是在需要的时候才会获取）。为了避免创建太多的session，可以使用TreadLocal来获取当前的session，无论你调用多少次getCurrentSession（）方法，返回的都是同一个session |

1. **HibernateSession的load和get方法区别是什么？**

|  |
| --- |
| ①如果没有找到符合条件的记录，get方法返回null值，而load方法会抛出异常；  ②get方法直接返回实体类对象，load方法返回实体类对象的代理；  ③在Hibernate3之前，get方法只在一级缓存（内部缓存）中进行数据查找，如果没有找到对应的数据则越过二级缓存，直接发出SQL语句完成数据的读取；load方法则可以充分利用二级缓存中现有数据，进行延迟加载。当然从Hibernate3开始，get方法不再是对二级缓存只写不读，它也是可以访问二级缓存的；  --------------------------------------------------------------------------------------------------------------------------------------------------------  简单的说，对于load()方法，hibernate认为该数据在数据库中一定存在，可以放心的使用代理来实现延迟加载，如果没有数据，就会抛出异常，而通过get()方法去取数据，是可以不存在的 |

1. **HibernateQuery接口的list方法和iterate方法有什么区别？**

|  |
| --- |
| ①list()方法返回的每个对象都是完整的（对象中的每个属性都被表中的字段填充上了），list方法无法利用缓存，它对一级缓存只写不读；  ②iterate方法可以充分利用一级缓存，它所返回的对象中仅包含了主键值（标识符），只有当你对iterator中的对象进行操作时，Hibernate才会向数据库再次发送SQL语句来获取该对象的属性值；  ③list方法不会引起N+1查询问题，而iterate方法会引起N+1查询问题 |

1. **简述Hibernate中常见的优化策略？**

|  |
| --- |
| ①指定合理的缓存策略；  ②采用合理的Session管理机制；  ③尽量使用延迟加载特性；  ④设定合理的批量处理参数；  ⑤如果可以，选用UUID作为主键生成器；  ⑥如果可以, 选用基于version的乐观锁替代悲观锁；  ⑦开发过程中, 开启hibernate.show\_sql选项查看生成的SQL,从而了解底层的状况;开发完成后关闭此选项；  ⑧数据库本身的优化(合理的索引,缓存,数据分区策略等)也会对持久层的性能带来可观的提升,这些需要专业的DBA提供支持； |

1. **谈一谈Hibernate的一级缓存、二级缓存和查询缓存？**

|  |
| --- |
| ①一级缓存:Hibernate的Session提供了一级缓存的功能，默认总是有效的，当应用程序保存持久化实体、修改持久化实体时，Session并不会立即把这种改变提交到数据库，而是缓存在当前的Session中，除非显示调用了Session的flush()方法或通过close()方法关闭Session。通过一级缓存，可以减少程序与数据库的交互，从而提高数据库访问性能。  ②二级缓存:SessionFactory级别的二级缓存是全局性的，所有的Session可以共享这个二级缓存。不过二级缓存默认是关闭的，需要显示开启并指定需要使用哪种二级缓存实现类(可以使用第三方提供的实现)。一旦开启了二级缓存并设置了需要使用二级缓存的实体类，SessionFactory就会缓存访问过的该实体类的每个对象，除非缓存的数据超出了指定的缓存空间。  ③查询缓存:一级缓存和二级缓存都是对整个实体进行缓存，不会缓存普通属性，如果希望对普通属性进行缓存，可以使用查询缓存。查询缓存是依赖二级缓存的，注意先打开二级缓存。Iterator()不使用查询缓存,查询缓存只对query.list()有效 开启条件： 1) hibernate.cache.use\_query\_cache=true 2)Query或者Criteria使用时，设置setCacheable(true) |

1. **阐述Hibernate实体对象的三种状态？**

|  |
| --- |
| ①瞬时态（transient）：没有持久化标识OID,没有被纳入到Session对象的管理；  ②持久态（persistent）：有持久化标识OID,已经被纳入到Session对象的管理；  ③游离态（detached）有持久化标识OID,没有被纳入到Session对象的管理； |

1. **简述Hibernate的延迟加载？**

|  |
| --- |
| 1).延迟加载，也叫懒加载，它是Hibernate为提高程序执行效率而提供的一种机制，即只有真正使用该对象的数据时才会创建。  2).Hibernate中主要是通过代理（proxy）机制来实现延迟加载。它的具体过程：Hibernate丛数据库获取某一个对象数据时、获取某一个对象的集合属性值时，或获取某一个对象所关联的另一个对象时，由于没有使用该对象的数据，hibernate并不是数据库加载真正的数据，而只是为该对象创建一个代理对象来代表这个对象，这个对象上的所有属性都是默认值；只有在真正需要使用该对象的数据时才创建这个真实对象，真正从数据库中加载它的数据，这样在某些情况下，就可以提高查询效率 |

1. **ibatis(mybatis)、Hibernate有什么不同？**

|  |
| --- |
| 1. ibatis(mybatis)它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句，这样我们可能自行编写高效的sql语句。但是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。 2. Hibernate对象/关系映射能力强，数据库无关性好，因为hibernate自动生成sql语句，但是我们无法控制该语句，我们就无法去写特定的高效率的sql。 |

1. **谈一谈你对Spring的理解？**

|  |
| --- |
| ①Spring是一个开源的业务层框架，分模块，一站式框架，它能够整合各种其他主流框架；  ②Spring的实质就是一个实现了工厂模式的工厂类，在其配置文件中，通过添加<bean>标签，来创建实例对象；  ③Spring的核心——IoC/DI;  a)IoC(Inverse of Control）控制反转，将对象的创建全交给Spring去管理，然 后Spring容易通过依赖注入的方式，注入给调用者。这样做的好处是，让bean与bean之间以配置文件的形式组织在一起，而不是以硬编码的方式耦合在一起。  b)依赖注入的方式有三种：接口注入、Setter方法注入（使用<property name="" value或者ref="">）、构造注入；  ④Spring的核心——AOP;  a)AOP(面向切面编程)，可以在不修改源代码的前提下，对程序进行增强，例如，加入日志，加入权限判断，加入异常处理等；  b)AOP底层使用的是代理技术，分为JDK动态代理（面向接口）和CGLIB动态代理（没接口的，产生子类代理）； |

1. **SpringMVC的执行过程？**

|  |
| --- |
| 1、用户发送请求至前端控制器DispatcherServlet  2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。  3、处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。  4、DispatcherServlet通过HandlerAdapter处理器适配器调用处理器  5、HandlerAdapter执行处理器(handler，也叫后端控制器)。  6、Controller执行完成返回ModelAndView  7、HandlerAdapter将handler执行结果ModelAndView返回给DispatcherServlet  8、DispatcherServlet将ModelAndView传给ViewReslover视图解析器  9、ViewReslover解析后返回具体View对象  10、DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。  11、DispatcherServlet响应用户 |

1. **SpringMVC与Strus2的区别？**

|  |
| --- |
| 1). springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过虑器。  2). springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。  3). Struts采用值栈存储请求和响应的数据，通过OGNL存取数据， springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面 |

1. **Spring中自动装配的方式有哪些？**

|  |
| --- |
| >no：不进行自动装配，手动设置Bean的依赖关系。  >byName：根据Bean的名字进行自动装配。  >byType：根据Bean的类型进行自动装配。  >constructor：类似于byType，不过是应用于构造器的参数，如果正好有一个Bean与构造器的参数类型相同则可以自动装配，否则会导致错误。  >autodetect：如果有默认的构造器，则通过constructor的方式进行自动装配，否则使用byType的方式进行自动装配。 |

1. **阐述Spring框架中Bean的生命周期？**

|  |
| --- |
| 1). 实例化一个Bean－－也就是我们常说的new；  2). 按照Spring上下文对实例化的Bean进行配置－－也就是IOC注入；  3). Spring调用Bean实现BeanNameAware、BeanFactoryAware的方法  4). Spring对Bean调用的方法进行增强  5). 当Bean不再需要时，销毁 |

1. **Shiro 的优点？**

|  |
| --- |
| > 简单的身份认证, 支持多种数据源  > 对角色的简单的授权, 支持细粒度的授权(方法级)  > 支持一级缓存，以提升应用程序的性能；  > 内置的基于 POJO 企业会话管理, 适用于 Web 以及非 Web 的环境  > 非常简单的加密 API  > 不跟任何的框架或者容器捆绑, 可以独立运行 |

1. **简述 Shiro 的核心组件？**

|  |
| --- |
| > Subject: 正与系统进行交互的人, 或某一个第三方服务.  > SecurityManager: Shiro 架构的心脏, 用来协调内部各安全组件, 管理内部组件实例  > Realms: 本质上是一个特定安全的 DAO. 当配置 Shiro 时, 必须指定至少一个 Realm 用来进行身份验证和/或授权. |

1. **如何配置在 Spring 中配置使用 Shiro？**

|  |
| --- |
| ①. 在 web.xml 中配置 Shiro 的 Filter  ②. 在 Spring 的配置文件中配置 Shiro:  > 配置自定义 Realm：实现自定义认证和授权  > 配置 Shiro 实体类使用的缓存策略  > 配置 SecurityManager  > 配置保证 Shiro 内部 Bean 声明周期都得到执行的 Lifecycle Bean 后置处理器  > 配置AOP 式方法级权限检查  > 配置 Shiro Filter |

1. 事务
2. **事务的ACID？**

|  |
| --- |
| 1). 原子性（Atomicity）  原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生  2). 一致性（Consistency）  事务必须使数据库从一个一致性状态变换到另外一个一致性状态  3). 隔离性（Isolation）  事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。  4). 持久性（Durability）  持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。 |

1. **事务隔离级别解决？**

|  |
| --- |
| 事务隔离级别解决  1).未提交读read uncommitted  会发生 脏读、不可重复读、虚读  2).已提交读read committed  解决脏读,但是不可重复读和虚读有可能发生  3).重复读repeatable read  解决脏读和不可重复读,但是虚读有可能发生.  4).串行化serializable  避免脏读，不可重复读，虚读的发生 |

1. **Spring事务的传播机制？**

|  |
| --- |
| Spring事务的传播机制  1).PROPAGATION\_REQUIRED  如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是最常见的选择。  2).PROPAGATION\_SUPPORTS  支持当前事务，如果当前没有事务，就以非事务方式执行。  3).PROPAGATION\_MANDATORY  使用当前的事务，如果当前没有事务，就抛出异常。  4).PROPAGATION\_REQUIRES\_NEW  新建事务，如果当前存在事务，把当前事务挂起。  5).PROPAGATION\_NOT\_SUPPORTED  以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。  6).PROPAGATION\_NEVER  以非事务方式执行，如果当前存在事务，则抛出异常。  7).PROPAGATION\_NESTED  如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION\_REQUIRED类似的操作。 |

1. **分布式事务解决方案？**

|  |
| --- |
| 分布式事务：当一个应用操作的不只是一个数据时，那么事务的连接也就不一样。这个时候如果一个方法同上要解决两个数据库的事务的原子性。那么就需要使用分布式事务。  方案一：如果使用同一个Spring容器管理了多个数据库，那么就可以使用Spring JTA解决分布式事务，这个只是数据库层面的一个分布式，服务并没有分布式。  方案二：如果使用的不同的Spring容器，也就是项目时分布式的。也就是说现在serviceA要操作数据库A，serviceB操作数据库B，并且serviceA和serviceB在不同的子系统中，但是现在需要在serviceA中调用serviceB完成业务，那么如何解决serviceA和serviceB的分布式事务的提交呢？  答： 首先这种分布式事务到目前为止没有完美的解决方法，目前比较流行的解决方法是结合MQ消息中间件实现的可靠消息最终一致性。  案例说明：现在小明给小强转账1000，那么小明就应该扣1000，而小强要加1000，但是扣款的服务在系统A的serviceA中，而加款的服务在系统B的serviceB中，这个时候在serviceA调用serviceB完成转账过程。事务执行过程  a). serviceA开始  b). 在执行扣款之前，先向redis缓存中添加一条消息，消息采用hash方式，redis的key写死，比如是MESSAGE\_TX,而里面具体的是—— uuid(由serviceA生成):{mid:uuid(同前面一样),status:”1”,…}  c). 执行扣款操作  d). 使用MQ消息(RocketMQ支持消息事务)给serviceB所在的系统发送消息，发送消息包含上面生成的uuid和具体的加款操作。并且对MQ的消息事务进行超时捕获，如果出现消息事务超时，那么就在超时的异常代码块中读取刚刚存储在redis的消息，如果消息的状态被修改，那么就代表serviceB事务正确提交，那么serviceA也提交，如果消息状态没有改变，那么serviceB事务没有正确提交，所以serviceA事务回滚。  e). 在serviceB中接收到MQ消息之后，执行serviceB的加款操作  f). 如果serviceB加款操作执行成功，那么修改uuid对应的redis缓存中的status状态，为0  g). 事务结束。  其他分布式解决方案：<http://www.cnblogs.com/taiyonghai/p/6094350.html> |

1. 项目面试实战
2. **商城项目介绍案例**

|  |
| --- |
| \*\*\*网上商城是一个综合性的B2C平台，类似京东商城。该项目采用Spring+SpirngMVC+Mybatis框架搭建的maven工程，并采用分布式架构按功能将系统分为不同的子系统，将不同的子系统部署到不同的节点服务器独立运行。在各个系统之间采用dubbo+zookeeper进行通信，实现了基于SOA面向服务的架构，使得服务层与表现层分离。该项目划分为六大子系统：后台管理系统、前台系统、搜索系统、会员系统、订单系统、、单点登录系统。我主要参与了后台管理系统中cms模块、商品管理模块，搜索系统，前台系统中首页以及单点登陆系统的实现。后台cms模块主要是对首页展示内容的管理，而商品管理就是对商品信息以及相关基础信息的维护，对于商品管理除去基本的增删查改功能以外，还需要在增删改的时候采用activemq消息队列实现搜索系统的索引同步以及商品详情页面的静态化处理，这样可以做到不同子系统之间的业务分离和消息通信。搜索系统主要对商品信息的查询，这里不是简单的查询数据库，这里采用solrcluod提供的分布式搜索方案搭建搜索服务器，solrcluod是基于solr+zookeeper集群的，solr的索引由商品管理时同步维护。首页信息展示数据由cms系统维护，但是由于首页是每个网站的入口，访问量比较大，这里采用了redis缓存技术减少对数据库的访问。单点登陆系统采用redis模拟Session，实现在不同系统中Session的统一管理 |

1. **商城门户首页描述**

|  |
| --- |
| 商城门户首页作为系统的入口，所以首页的访问量也是非常大的，并且数据是由后台cms内容管理系统动态维护，如果每一次访问都从数据库查询，那么数据库的压力就会特别大，为了减轻数据库压力，这里我们采用redis缓存技术存储首页的数据。Redis 是一个高性能的key-value数据库，提供多种数据类型的支持，并且提供了数据持久化解决方案——RDB快照和AOF操作日志追加方式。在本系统中数据类型采用的是string类型，在第一次访问首页的时候去查询数据库得到数据之后，我们将得到的数据转换为json字符串保存在redis缓存中，后面的查询都会从redis缓存中获取json字符串再返回给页面渲染。同时持久化方案采用的是RDB快照方式，因为首页的数据相对而言还是非常大的，RDB快照持久化是由子进程完成这些工作，这样就可以极大的避免服务进程执行IO操作，同时如果redis服务器宕机，RDB快照持久化的启动更快。由于redis服务器不仅仅只保存首页的数据，而且在SSO单点登陆系统中还会使用redi模拟session。这个时候如果有100万用户登陆，那么redis的缓存数据就特别大了，如果使用一个Redis实例那么速度就会大大折扣，这里就可以使用redis集群将原来的Redis的所有的keys分发到各个服务器的Redis上，每个redis服务器之间是可以相互通信的，但是多个redis服务器通信时当其中一个节点发生故障，那么Redis集群就不能继续正常工作，所以需要一种复制机制（Master-Slave）机制，也就是需要为每一个redis服务器创建一个备用服务器。一般来讲我们会搭建3个redis集群服务器并且为这3个redis服务器都创建一个备用服务器，那么就需要6个redis服务器。再redis集群中采用哈希槽来存储客户端的keys。在Redis中，目前存在16384个哈希槽，它们将会被分配给集群中的每一个redis服务器。 |

1. **搜索子系统描述**

|  |
| --- |
| 搜索子系统提供查询接口，查询接口的业务实现并不是采用查询数据库的方式进行数据检索而是使用solrj查询solr服务器中的数据，因为对于电商项目而言搜索功能的使用频率是非常高的，倘若使用数据库检索的方式，那么会给数据库带来巨大的访问压力，为了减轻数据库的压力，这里就考虑采用solr搭建一个搜索服务器并在搜索子系统中采用solrj连接solr搜索服务器并对外提供搜索功能。在solr搜索服务器的solrcode中使用schema.xml配置了ikanalyzer 中文分词的fileType以及与商品表对应的file域，solr索引的维护由后台管理系统在维护商品信息时进行同步，同步是以activemq消息队列的方式通知搜索子系统进行索引的相关操作，采用activemq消息队列的方式既可以完成索引同步也可以使得商品维护和索引维护的业务分离。那么随着商城系统中维护的商品越来越多，那么对应的索引也就越来越大，而搜索的并发量随着用户的增加也会随之越大，那么这个时候单机版的solr服务器可能难以提供高性能、高容错的数据检索能力，那么这个时候系统将单机版的solr搜索服务器升级为分布式搜索方案——SolrCloud。solrCloud是基于solr集群和zookeeper集群的，zookeeper作为solr集群的配置中心统一管理solr集群的配置文件。由于zookeeper集群采用一种称为Leader election的选举算法来实现高可用，也就是说在整个集群运行过程中，集群中只有一个Leader，其他的都是Follower，如果ZooKeeper集群在运行过程中Leader出了问题，系统会采用该算法重新选出一个Leader，所以在zookeeper集群中至少需要搭建3台zookeeper服务器来完成完成选举算法。在搭建solr集群中，集群至少应该包含两个solr实例，同时为了实现solr集群的高可用，那么对每一个solr实例搭建一个备用服务器，所以至少应该搭建4台solr服务器，由于solr实例需要运行在tomcat容器中，所以还要搭建4个tomcat节点。solr集群搭建完毕需要将任意一个solr实例的conf目录上传到zookeeper集群进行配置管理，这样solrcloud分布式搜索方案就搭建完成。 |

1. **单点登录系统描述**

|  |
| --- |
| 由于系统采用的是分布式架构，不同的子系统要实现统一的登录权限验证。如果采用传统的登录方式，那么登录成功之后将用户信息存储在session中，但是不同子系统中如果实现session共享呢？这里有一种解决方法是使用tomcat的广播机制以session复制的方式完成session共享，但是由于本系统用户信息需要在除开后台管理系统以外的其他5个子系统中实现session共享，那么在这里至少需要在5个tomcat中进行广播机制的session复制，这时每个tomcat直接都会两两相连，这样会导致tomcat的性能较低，并不能完美的解决分布式架构下统一登录权限校验，那么我们采用目前企业比较流行的业务整合方案——SSO单点登录系统。单点登录系统是采用redis缓存来模拟session机制，即用户登录成功后，我们将用户信息以json字符串的形式存储在redis缓存中，并生成一个唯一token作为redis存储用户信息的key。然后将该token写入cookie并返回给浏览器客户端。由于各个子系统的域名不一样，所以我们必须在返回cookie之前设置有效域名，使得不同子系统在发送请求时，都可以携带该cookie到后台进行登录拦截。 |

1. **项目问题1：zookeeper集群的工作原理？**

|  |
| --- |
| Zookeeper的核心是原子广播，这个机制保证了各个server之间的同步。实现这个机制的协议叫做Zab协议。Zab协议有两种模式，它们分别是恢复模式和广播模式。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数server的完成了和leader的状态同步以后，恢复模式就结束了。一旦leader已经和多数的follower进行了状态同步后，他就可以开始广播消息了，即进入广播状态。这时候当一个server加入zookeeper服务中，它会在恢复模式下启动，发现leader，并和leader进行状态同步。待到同步结束，它也参与消息广播。 |

1. **项目问题2：为什么zookeeper集群的数目一般为奇数个？**

|  |
| --- |
| Leader选举算法采用了Paxos协议，Paxos核心思想是当多数Server写成功，则任务数据写成功。如果有3个Server，则两个写成功即可；如果有4或5个Server，则三个写成功即可。Server数目一般为奇数（3、5、7）如果有3个Server，则最多允许1个Server挂掉；如果有4个Server，则同样最多允许1个Server挂掉由此， 我们看出3台服务器和4台服务器的的容灾能力是一样的，所以为了节省服务器资源，一般我们采用奇数个数，作为服务器部署个数。 |

1. **项目问题3：zookeeper的Leader选举机制？**

|  |
| --- |
| 要想成为leader,必须有过半的节点投票给该节点，所以节点格式要求为奇数个！默认选举算法FastLeaderElectio  (1) 每个Server启动以后都询问其它的Server它要投票给谁。  (2)对于其他server的询问，server每次根据自己的状态都回复自己推荐的leader的id和上一次处理事务的zxid（系统启动时每个server都会推荐自己）  (3)收到所有Server回复以后，就计算出zxid最大的哪个Server，并将这个Server相关信息设置成下一次要投票的Server。  (4)计算这过程中获得票数最多的的sever为获胜者，如果获胜者的票数超过半数，则改server被选为leader。否则，继续这个过程，直到leader被选举出来 |

1. **项目问题4：使用Redis有哪些好处？**

|  |
| --- |
| 1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)  2) 支持丰富数据类型，支持string，list，set，sorted set，hash  3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行  4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除 |

1. **项目问题5：Redis 集群常见的性能问题都有哪些？如何解决？**

|  |
| --- |
| (1) Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久化.因为快照比较大时对性能影响是非常大的，会间断性暂停服务  (2)Redis主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内 |

1. **项目问题6：redis内存数据到达上限，该怎么处理？**

|  |
| --- |
| redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。redis 提供几种种数据淘汰策略：  (1) voltile-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  (2) 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  (3) 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 |

1. **项目问题7：redis持久化策略比较？**

|  |
| --- |
| (1). Snapshotting快照：当redis需要做持久化时，redis会fork一个子进程；子进程将数据写到磁盘上一个临时RDB文件中；当子进程完成写临时文件后，将原来的RDB替换掉，缺点：如果redis服务死掉，可能造成最近数据丢失  (2). AOF(Append-only file)：AOF就可以做到更精细的持久化，只需要在配置文件中开启（默认是no），appendonly yes开启AOF之后，redis每执行一个修改数据的命令，都会把它添加到aof文件中，当redis重启时，将会读取AOF文件进行“重放”以恢复到 redis关闭前的最后时刻 |

1. **项目问题8：SolrCloud描述？**

|  |
| --- |
| solrcloud是solr提供的集群方案,基于solr和zookeeper的分布式搜索，一个solrcloud集群可以看成是一个collection,一个collection分为多个分片Shard,一个分片又由多个solrcore组成,其中这几个solrcore是一主多从。solrcloud使用zookeeper来统一管理solr集群的配置信息 ,比如solrConfig.xml和schema.xml;这样所有的solr节点都从zookeeper集群中获取相关配置信息，并且访问solrcloud的时候是通过zookeeper负载均衡来访问不同的solr节点。  solrcloud的几个特色功能:1)集中式的配置信息,2)自动容错3)近实时搜索4)查询时自动负载均衡 |

1. **项目问题9：activeMQ的Queue和Topic消息的区别？**

|  |  |  |
| --- | --- | --- |
| 类型 | Topic | Queue |
| 概要 | Publish Subscribe messaging 发布订阅消息 | Point-to-Point 点对点 |
| 有无状态 | topic数据默认不落地，是无状态的。 | Queue数据默认会在mq服务器上以文件形式保存，比如Active MQ一般保存在$AMQ\_HOME\data\kr-store\data下面。也可以配置成DB存储。 |
| 完整性保障 | 并不保证publisher发布的每条数据，Subscriber都能接受到。 | Queue保证每条数据都能被receiver接收。 |
| 消息是否会丢失 | 一般来说publisher发布消息到某一个topic时，只有正在监听该topic地址的sub能够接收到消息；如果没有sub在监听，该topic就丢失了。 | Sender发送消息到目标Queue，receiver可以异步接收这个Queue上的消息。Queue上的消息如果暂时没有receiver来取，也不会丢失。 |
| 消息发布接收策略 | 一对多的消息发布接收策略，监听同一个topic地址的多个sub都能收到publisher发送的消息。Sub接收完通知mq服务器 | 一对一的消息发布接收策略，一个sender发送的消息，只能有一个receiver接收。receiver接收完后，通知mq服务器已接收，mq服务器对queue里的消息采取删除或其他操作。 |

1. **项目问题10：ActiveMQ丢消息问题解决**

|  |
| --- |
| 用持久化消息或者启动事务后，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。 |

1. **项目问题11：单点登录禁用cookie解决方案**

|  |
| --- |
| 采用url重写技术：  （1）登录成功之后返回浏览器一个token,该token对应redis缓存中的key  （2）登录成功之后，每次浏览器与服务器进行跳转页面交互，浏览器需要发送token参数，服务器需要接受token参数然后再返回token参数到所跳转的页面。而ajax请求只需要从本页面取得token，然后发送即可 |