Java 基础

题目 1：遍历 map 的几种方法？【了解】

（1） 问题分析：
考官主要考核 map 集合的四种遍历方式
（2） 核心答案讲解：
第一种，通过 key 取值。(Map.keyset()遍历 key 和 value)。
第二种，通过迭代器取值。（Map.enteySet()使用 iterator 遍历 key 和 value）。
第三中，通过 entryset()。（通过 Map.enteySe()遍历 key 和 value)。
第四中，通过 map 的 value 方法。（Map.values()遍历所有的 value）。
（3） 问题扩展：
无
（4） 结合项目中使用：
在商城中，通过 map 记录保存相关商品信息。

题目 2：hashmap 的特性？

（1） 问题分析：
考官主要考核 map 中 hashmap 自身独有的特点。
（2） 核心答案讲解：
1. 允许空键和空值（但空键只有一个，且放在第一位）
2. 元素是无序的，而且顺序会不定时改变
3. key 用 Set 存放，所以想做到 key 不允许重复，key 对应的类需要重写 hashCode 和 equals 方法。
4. 底层实现是 链表数组，JDK 8 后又加了红黑树。
5. 实现了 Map 全部的方法
（3） 问题扩展：
红黑树本质上是一种二叉查找树，但它在二叉查找树的基础上额外添加了一个标记（颜色），同时具有一定的规则。
*每个节点要么是红色，要么是黑色；
*根节点永远是黑色的；
*所有的叶节点都是是黑色的（注意这里说叶子节点其实是上图中的 NIL 节点）；
*每个红色节点的两个子节点一定都是黑色；
*从任一节点到其子树中每个叶子节点的路径都包含相同数量的黑色节点
（4） 结合项目中使用：
有一个 people 类，HashMap 的 key 想通过 name 和 age 判断 people 是否相等，而不是通过 people 对象的存储地址.从而根据 HashMap 原理，需同时重写 equals() 和

hashCode()；

题目 3：Java 虚拟机中的内存模型？

（1） 问题分析：
考官主要是对 Java 程序中各个变量的访问规则的考核。Java 虚拟节内存空间分为方法区，Java 堆，Java 栈，本地方法栈。

（2） 核心答案讲解：
Java 虚拟机运行时内存所有的类的实例（不包括局部变量与方法参数）都存储在 Java 堆中，每条线程有自己的工作内存（Java 栈），不同线程之间无法直接访问对方工作内存中的变量。方法区用于存储被虚拟机加载的类信息、常量、static 变量等数据，堆用于存储对象实例，比如通过 new 创建的对象实例就保存在堆中，堆中的对象的由垃圾回收器负责回收。Java 栈用于实现方法调用，每次方法调用就对应栈中的一个栈帧，栈帧包含局部变量表、操作数栈、方法接口等于方法相关的信息，栈中的数据当没有引用指向数据时，这个数据就会消失。本地方法栈保存的是本地方法的调用。

（3） 问题扩展：
线程安全问题就是，多个线程的工作内存同时对堆中同一个数据的修改，使用 Java 锁避免线程安全问题。

（4） 结合项目中使用：
多线程消费同一个商品

题目 4：简单说说 Java 中的异常处理机制的简单原理和应用。【理解】

（1） 问题分析：
考官是对异常的考核。异常的分类，非检查异常和检查异常，try chtch finally 的使用。

（2） 核心答案讲解：
所有异常的根类为 Java.lang.Throwable.Throwable 派生了 2 个子类：Error 和 Exception。
Error 代表了 JVM 本身的错误，不能被程序员通过代码处理，如内存溢出。
Exception 分为 IoException 和 RuntimeException 。
Error 和 RuntimeException 以及他们的子类。Javac 在编译时，不会提示和发现这样的异常，不要求在程序处理这些异常称之为非检查异常，比如下标越界。编译器强制必须 try.catch 处理或 throws 声明继续抛给上层调用方法处理的异常称之为检查异常，比如使用 jdbc 连接数据库的 SQLException。try 块中放可能发生异常的代码。每一个 catch 块用于捕获并处理一个特定的异常，或者这异常类型的子类，顺序为从小到大。finally 无论异常是否发生，异常是否匹配被处理，finally 都会执行。

（3） 问题扩展：

Spring 框架的事务默认是 RuntimeException 才进行回滚，修改 Transactional 注解中的 rollbackFor 属性可以指定为 exception 异常回滚

（4） 结合项目中使用：

编写自定义异常，利用 throw 抛出自定义异常

题目 5：创建线程的几种方式？

(1) 问题分析：

考官主要想对线程方面的考核，如线程的生命周期、线程安全问题等。

(2) 核心答案讲解：

通过继承 Thread 类实现，多个线程之间无法共享该线程类的实例变量。

实现 Runnable 接口，较继承 Thread 类，避免继承的局限性，适合资源共享。

使用 Callable，方法中可以有返回值，并且抛出异常。

创建线程池实现，线程池提供了一个线程队列，队列中保存所有等待状态的线程，避免创建与销毁额外开销，提高了响应速度。

(3) 问题扩展：

线程的生命周期:线程要经历新建、就绪、运行(活动)、阻塞和死亡五种不同的状态。这五种状态都可以通过 Thread 类中的方法进行控制。① 新建状态：使用 new 操作符创建一个线程后，该线程仅仅是一个空对象，这时的线程处于创建状态。② 就绪状态：使用 start()方法启动一个线程后，系统为该线程分配了除 CPU 外的所需资源，使该线程处于就绪状态。③ 运行状态：系统真正执行线程的 run()方法。④ 阻塞和唤醒线程阻塞状态：使用 sleep()，wait()方法进行操作。⑤ 死亡状态：线程执行了 interrupt()或 stop()方法，那么它也会以异常退出的方式进入死亡状态。

线程安全问题:使用 synchronized 声明同步或使用锁 lock，Lock 使用起来比较灵活，但需要手动释放和开启，采用 synchronized 不需要用户去手动释放锁。

(4) 结合项目中使用：

模拟实现银行业务调度系统逻辑，具体需求如下：

1. 银行内有 6 个业务窗口，1 - 4 号窗口为普通窗口，5 号窗口为快速窗口，6 号窗口为 VIP 窗口。
2. 有三种对应类型的客户：VIP 客户，普通客户，快速客户（办理如交水电费、电话费之类业务的客户）。
3. 异步随机生成各种类型的客户，生成各类型用户的概率比例为：VIP 客户 ：普通客户 ：快速客户 = 1 ：6 ：3。
4. 客户办理业务所需时间有最大值和最小值，在该范围内随机设定每个 VIP 客户以及普通客户办理业务所需的时间，快速客户办理业务所需时间为最小值（提示：办理业务的过程可通过线程 Sleep 的方式模拟）。
5. 各类型客户在其对应窗口按顺序依次办理业务。
6. 当 VIP（6 号）窗口和快速业务（5 号）窗口没有客户等待办理业务的时候，这两个窗口可以处理普通客户的业务，而一旦有对应的客户等待办理业务的时候，则优先处理对应客户的业务。
7. 随机生成客户时间间隔以及业务办理时间最大值和最小值自定，可以设置。
8. 不要求实现 GUI，只考虑系统逻辑实现，可通过 Log 方式展现程序运行结果。

题目 6：谈谈你对垃圾回收机制的了解？

（1） 问题解析：

考官主要针对你对 GC 方面的考核，比如：什么是垃圾回收机制、JVM 回收特点等。

（2） 核心答案解析

什么是垃圾回收机制：在系统运行过程中，会产生一些无用的对象，这些对象占据着一定的内存，如果不对这些对象清理回收无用对象的内存，可能会导致内存的耗尽，所以垃圾回收机制回收的是内存。同时 GC 回收的是堆区和方法区的内存。

JVM 回收特点：(stop-the-world)当要进行垃圾回收时候，不管何种 GC 算法，除了垃圾回收的线程之外其他任何线程都将停止运行。被中断的任务将会在垃圾回收完成后恢复进行。GC 不同算法或是 GC 调优就是减少 stop-the-world 的时间。à(为何非要 stop-the-world)，就像是一个同学的聚会，地上有很多垃圾，你去打扫，边打扫边丢垃圾怎么都不可能打扫干净的哈。当在垃圾回收时候不暂停所有的程序，在垃圾回收时候有 new 一个新的对象 B，此时对象 A 是可达 B 的，但是没有来及标记就把 B 当成无用的对象给清理掉了，这就会导致程序的运行会出现错误。

（3） 问题扩展：

如何判断哪些对象需要回收呢：

1.引用计数算法(java 中不是使用此方法)：每个对象中添加一个引用计数器，当有别人引用它的时候，计数器就会加 1，当别人不引用它的时候，计数器就会减 1，当计数器为 0 的时候对象就可以当成垃圾。算法简单，但是最大问题就是在循环引用的时候不能够正确把对象当成垃圾。

2.根搜索方法（这是后面垃圾搜集算法的基础）：这是 JVM 一般使用的算法，设立若干了根对象，当上述若干个跟对象对某一个对象都不可达的时候，这个对象就是无用的对象。对象所占的内存可以回收。

根搜索算法的基础上，现代虚拟机的实现当中，垃圾搜集的算法主要有三种，分别是标记-清除算法、复制算法、标记-整理算法。

（4） 结合项目中使用：

尽量不要 new 很大的 object，大对象（>=85000Byte）直接归为 G2 代，GC 回收算法从来不对大对象堆（LOH）进行内存压缩整理，因为在堆中下移 85000 字节或更大的内存块会浪费太多 CPU 时间；不要频繁的 new 生命周期很短 object，这样频繁垃圾回收频繁压缩有可能会导致很多内存碎片，可以使用设计良好稳定运行的对象池（ObjectPool）技术来规避这种问题；之前在维护一个系统的时候，发现有很多大数据量的处理逻辑，但竟然都没有批量和分页处理，随着数据量的不断膨胀，隐藏的问题会不断暴露。然后我在重写的时候，都按照批量多次的思路设计实现，有了多线程、多进程和分布式集群技术，再大的数据量也能很好处理，而且性能不会下降，系统也会变得更加稳定可靠。

题目 7：说说 HashCode()、equals()的区别？

（1） 问题分析：

考官主要想对 hashCode()方法和 equal()方法作用和效率上进行比较。

（2） 核心答案讲解：

equal()相等的两个对象他们的 hashCode()肯定相等，也就是用 equal()对比是绝

对可靠的。
hashCode()相等的两个对象他们的 equal()不一定相等，也就是 hashCode()不是绝对可靠的。
对于需要大量并且快速的对比的话如果都用 equal()去做显然效率太低，所以解决方式是，每当需要对比的时候，首先用 hashCode()去对比，如果 hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用 equal()去再对比了）,如果 hashCode()相同，此时再对比他们的 equal()，如果 equal()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

（3） 问题扩展:
hashCode 的重写:
hashCode()和 equal()一样都是基本类 Object 里的方法，而和 equal()一样，Object 里 hashCode()里面只是返回当前对象的地址，如果是这样的话，那么我们相同的一个类，new 两个对象，由于他们在内存里的地址不同，则他们的 hashCode（）不同，所以这显然不是我们想要的，所以我们必须重写我们类的 hashCode()方法，即一个类，在 hashCode()里面返回唯一的一个 hash 值。
equals 方法的作用:
默认情况（没有覆盖 equals 方法）下 equals 方法都是调用 Object 类的 equals 方法，而 Object 的 equals 方法主要用于判断对象的内存地址引用是不是同一个地址（是不是同一个对象）;
要是类中覆盖了 equals 方法，那么就要根据具体的代码来确定 equals 方法的作用了，覆盖后一般都是通过对象的内容是否相等来判断对象是否相等。

（4） 结合项目中使用:
equals 方法是默认的判断 2 个对象是否相等的方法，在 Object 类里有实现，判断的是 2 个对象的内存地址。在 hibernate 中，不允许存在同类对象中有 2 个一样的实例。hibernate 通过 equals 方法做判断。如:

```
User u1 = new User(“张三”);
User u2 = new User(“李四”);
User u3 = new User(“张三”);
```

按照项目需求，用户只要名字相同，就表示同一个用户，所以我们认为，u1 和 u3 是同一个人，同一个对象。但是因为 u1,u2,u3 三者的内存地址都各不相同，所以 hibernate 会认为这是 3 个不同的对象。这与我们假设的出了矛盾。 因此，我们将覆盖 Object 类中的 equals 方法。

```
public class User{
private String userName;
….//get ， set 方法省
//覆盖 Object 里的 equals 方法
public boolean equals(Object arg0){
if (!(arg0 instanceof User)){
return false;
}
User user = (User)arg0;
//如果名字相同，则表示属于同一个对象。
if(user.getName().equals(this.getName)){
return true;
}else{
return false; }
}
```

这样 hibernate 在插入数据的时候，如果传过来一个叫”张三”的用户,hibernate 会先判断有没有叫“张三”的用户，如果没有，就允许插入，如果有，就不允许插入。这样做可以保证数据的高度一致性，不同的项目有不同的需求，所以要根据自己的需求来覆盖 equals 方法。

题目 8：LinkedList 和 ArrayList 区别？

（1） 问题分析：
考官主要想对集合进行考核，如集合的特点及原理方面的了解。
（2） 核心答案讲解：
ArrayList 是基于动态数组的数据结构，LinkedList 基于链表的数据结构。
ArrayList 查询操作快，LinedList 增删操作快。
（3） 问题扩展：
为了使得突破动态长度数组而衍生的 ArrayList 初始容量为 10，每次扩容会固定为之前的 1.5 倍，每次扩容的过程是内部复制数组到新数组；
LinkedList 的每一个元素都需要消耗一定的空间。
（4） 结合项目中使用：
对于需要快速插入，删除元素，应该使用 LinkedList。
对于需要快速随机访问元素，应该使用 ArrayList。
对于“单线程环境” 或者 “多线程环境，但 List 仅仅只会被单个线程操作”，此时应该使用非同步的类(如 ArrayList)。
对于“多线程环境，且 List 可能同时被多个线程操作”，此时，应该使用同步的类(如 Vector)。

题目 9：String，StringBuilder，StringBuffer 三者的区别？

（1） 问题分析：
考官主要想对 final 修饰符的作用，同步锁，以及数据类型的考察。在工作中为什么业务层频繁的拼接 sql 不用 string 而用 StringBuilder,为什么 Stringbuilder 比 StringBuffer 效率高
（2） 核心答案讲解：
String 是引用类型，底层是被 final 修饰的字符数组，所以 String 相当于一个常量，是不可改变的，每拼接一次就会产生一个新的对象，而由于垃圾回收机制的原理，原有的对象不会立马被回收，这是对内存极大的消耗；而 StringBuilder 和 StringBuffer 是可变长度的，可以利用 append 方法向原有对象拼接，然后用 toString 方法将其转化为 String 类型；这两个相比起来 StringBuilder 的效率更高，因为他是非线程安全的，不需要花费资源去维护同步锁。
（3） 问题扩展：
Final 修饰符的作用是什么？
在工作中你们如果在业务层去拼接 sql，使用 String 类型去接收的吗？
（4） 结合项目中使用：
在项目中如果频繁的拼接字符串需要用什么类型对象去接收

题目 10：是否可以从一个 static 方法内部发出对非 static 方法的调用？

（1） 问题分析：
考官主要相对 static 方法的考察，涉及到 static 关键词考核，如抽象的（abstract）方法是否可同时是静态的（static）;static 可否用来修饰局部变量；内部类与静态内部类的区别；java 中是否可以覆盖(override) 一个 private 或者是 static 的方法

（2） 核心答案讲解：
不可以。static 方法是静态方法，是属于类的方法，非 static 方法是属于对象的方法。因为非 static 方法是要与对象关联在一起的，必须在创建出一个对象后，才可以通过这个对象调用非 static 方法；而 static 方法可以直接通过类名来调用，不需要创建对象。也就是说，在一个 static 方法被调用时，还可能没有创建任何实例对象，此时如果从 static 内部发出对非 static 方法的调用，非 static 方法是无法关联到对象的。

（3） 问题扩展：
static 表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态 static 代码块。
static 修饰的变量习惯称为静态变量，static 修饰的方法称为静态方法，static 修饰的代码块叫做静态代码块。
static 的意义在于方便在没有创建对象的情况下来进行调用（方法/变量）。
“static 方法就是没有 this 的方法。在 static 方法内部不能调用非静态方法，反过来是可以的。而且可以在没有创建任何对象的前提下，仅仅通过类本身来调用 static 方法。这实际上正是 static 方法的主要用途。

（4） 结合项目中使用：
1.在项目中，很多工具类会使用 static 定义方法，达到不用 new 对象直接类名.方法名直接调用，使用工具更方便，减少重复代码的作用。例如：项目中的 UUIDUtils
2 常见的单例模式
单例模式方法定义为静态方法：达到不能用该类在其他地方创建对象，而是通过该类自身提供的方法访问类中的那个自定义对象的目的。

题目 11：java 中 sleep 和 wait 的区别？

（1） 问题分析：
面试官考核的线程方面的问题，线程的生命周期与过程中的阻塞状态。

（2） 核心答案讲解：
对于 sleep()方法，我们首先要知道该方法是属于 Thread 类中的。而 wait()方法，则是属于 Object 类中的。sleep()方法导致了程序暂停执行指定的时间，让出 cpu 该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。在调用 sleep()方法的过程中，线程不会释放对象锁。而当调用 wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用 notify()方法后本线程才进入对象锁定池准备获取对象锁进入运行状态。

（3） 问题扩展：

线程生命周期及线程同步安全问题。

（4） 结合项目中使用：

生产者消费者 生产完成 仓库最多放一百个，必须等卖出五十个之后再通知生产

题目 12：实现一个线程有哪几种方式,各有什么优缺点,比较常用的是那种？

（1） 问题分析：

面试官考核的是线程创建方式及

（2） 核心答案讲解：

1.继承 Thread 类 2.实现 Runnable 接口

3.实现 Callable 接口 4.线程池方式

优缺点：

1.继承 Thread 类

优点：代码简单 。 缺点：该类无法集成别的类。

2.实现 Runnable 接口

优点：继承其他类。 同一实现该接口的实例可以共享资源。

缺点：代码复杂

3.实现 Callable

优点：可以获得异步任务的返回值

4.线程池：实现自动化装配，易于管理，循环利用资源。

（3） 问题扩展：

在 Java 中 Lock 接口比 synchronized 块的优势是什么？你需要实现一个高效的缓存，它允许多个用户读，但只允许一个用户写，以此来保持它的完整性，你会怎样去实现它？

整体上来说 Lock 是 synchronized 的扩展版，Lock 提供了无条件的、可轮询的(tryLock 方法)、定时的(tryLock 带参方法)、可中断的(lockInterruptibly)、可多条件队列的(newCondition 方法)锁操作。另外 Lock 的实现类基本都支持非公平锁(默认)和公平锁，synchronized 只支持非公平锁，当然，在大部分情况下，非公平锁是高效的选择。

（4） 结合项目中使用：

用户登录成功之后需要记录用户连续登录天数，给用户奖励积分。可以创建一个线程，单独调用积分接口。

题目 13：String s = new String("xyz");创建了几个 StringObject？是否可以继承 String 类？

(1) 问题分析：

考官主要是对 jvm 方面问题考察，以及 String 类的考察。例如 tring str = "aaa" + new String("bbb")创建了几个 String 对象？String 是否是基本类型，String 是否有

length（）这个方法

(2) 核心答案讲解：

String s=new String("xyz")究竟创建 String Object 分为两种情况：

1.如果 String 常理池中，已经创建"xyz"，则不会继续创建，此时只创建了一个对象 new String("xyz")

2.如果 String 常理池中，没有创建"xyz"，则会创建两个对象，一个对象的值是"xyz"，一个对象 new String("xyz")。

String 类不能被继承，因为它是被 final 修饰的。

(3) 问题扩展：

1、栈区（stack）—由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

2、堆区—一般由程序员分配释放,若程序员不释放，程序结束时可能由 OS 回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表

3、全局区（静态区）（static）全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域,未初始化的全局变量和未初始化的静态变量在相邻的另一块区域程序结束后由系统释放。

4、文字常量区—常量字符串就是放在这里的。程序结束后由系统释放

5、程序代码区—存放函数体的二进制代码。

(4) 结合项目中使用：

String 它被用于裁剪，拼接。搜索字符串，比较字符串，截取字符串，转换大小写等。在项目中不经常发生变化的业务场景中，优先使用 String。

题目 14：String s = "Hello";s = s + "world!";这两行代码执行后，原始的 String 对象中的内容到底变了没有？

（1）问题分析：

考官主要是对 String 类的考察。例如：String 是否是基本数据类型？String 是否能被继承？String 如何进行字符串截取？String、StringBuffer、StringBudder 区别？

(2) 核心答案讲解：

因为 String 被设计为不可变（immutable）类，所以它的所有对象都是不可变的对象。在这段代码中，s 原先指向一个 String 对象，内容是 hello，然后我们对 s 进行+操作，那么 s 所指向的那个对象是否发生了变化呢？答案是没有，这时，s 不再指向原来的那个对象了，而指向另一个 String 对象，内容为 hello world ，原来的那个对象还存在于内存中，只是 s 这个引用变量不再指向它了。

(3) 问题扩展：

如果经常对字符串进行各种各样的操作，或者说不可预见的修改，那么使用 String 对象来代表字符串的话会引起很大的内存的额开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这是应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。

String、StringBuffer、StringBudder 区别？

1. 可变不可变

String 类中使用字符数组保存字符串，如下就是，因为有“final”修饰符，所以可以知道 string 对象是不可变的。private final char value[];
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，如下就是，可知这两种对象都是可变的。char[] value;
2. 是否线程安全
String 中的对象是不可变的，也就可以理解为常量，显然线程安全。
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。
3.StringBuilder 与 StringBuffer 共同点
StringBuilder 与 StringBuffer 有公共父类 AbstractStringBuilder(抽象类)。
抽象类与接口的其中一个区别是：抽象类中可以定义一些子类的公共方法，子类只需要增加新的功能，不需要重复写已经存在的方法；而接口中只是对方法的申明和常量的定义。StringBuilder、StringBuffer 的方法都会调用 AbstractStringBuilder 中的公共方法，如 super.append(...)。只是 StringBuffer 会在方法上加 synchronized 关键字，进行同步。

(4) 结合项目中使用：
String：适用于少量的字符串操作的情况
StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况
StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况

题目 15：多线程解决同步问题的方式？

（1） 问题分析：
考官主要相对多线程方面的考核，被多个线程同时访问的，使用线程同步技术，确保数据在任何时刻最多只有一个线程访问

（2） 核心答案讲解：
同步代码块：使用 synchronized() 对需要完整执行的语句进行“包裹”，synchronized(Obj obj) 构造方法里是可以传入任何类的对象
同步方法：
在方法的申明里申明 synchronized

（3） 问题扩展
死锁
当线程需要同时持有多个锁时，有可能产生死锁。考虑如下情形：
线程 A 当前持有互斥所锁 lock1，线程 B 当前持有互斥锁 lock2。
接下来，当线程 A 仍然持有 lock1 时，它试图获取 lock2，因为线程 B 正持有 lock2，因此线程 A 会阻塞等待线程 B 对 lock2 的释放。
如果此时线程 B 在持有 lock2 的时候，也在试图获取 lock1，因为线程 A 正持有 lock1，因此线程 B 会阻塞等待 A 对 lock1 的释放。
二者都在等待对方所持有锁的释放，而二者却又都没释放自己所持有的锁，这时二者便会一直阻塞下去。这种情形称为死锁。

（4） 应用场景
XX 去银行开个银行账户，银行给 me 一张银行卡和一张存折，XX 用银行卡和存折来搞事情：银行卡疯狂存钱，存完一次就看一下余额；同时用存折子不停地取钱，取一次钱就看一下余额。

题目 16：Hashtable 与 HashMap 有什么不同之处？

（1） 问题分析

考官主要考核对于两个 map 的区别。

（2） 核心答案讲解：

相同点：

HashMap 和 Hashtable 都是存储“键值对(key-value)”的散列表，而且都是采用 拉链法实现的。

存储的思想都是：通过 table 数组存储，数组的每一个元素都是一个 Entry；而一个 Entry 就是一个单向链表，Entry 链表中的每一个节点就保存了 key- value 键值对数据

不同点：

1 继承和实现方式不同

2 线程安全不同

3 对 null 值的处理不同

4 支持的遍历种类不同

5 通过 Iterator 迭代器遍历时，遍历的顺序不同

6 容量的初始值 和 增加方式都不一样

7 添加 key-value 时的 hash 值算法不同

（3） 问题扩展

HashTable 中 hash 数组默认大小是 11，增加的方式是 old*2+1；HashMap 中 hash 数组的默认大小是 16，而且一定是 2 的指数；

扩容的临界点是加载因子 loadFactor>0.75，其中 loadFactor=size/capaticy

（4） 使用场景

在多线程中，我们可以自己对 HashMap 进行同步，也可以选择 ConcurrentHashMap。当 HashMap 和 Hashtable 都不能满足自己的需求时，还可以考虑新定义一个类，继承或重新实现散列表；当然，一般情况下是不需要的了。

题目 17：单例中的懒汉和饿汉模式的区别？

（1） 问题分析：

主要考察懒汉和饿汉模式在创建时的区别以及分别在什么情况下使用懒汉模式，什么情况下使用饿汉模式。

懒汉模式:在类加载的时候不被初始化。

饿汉模式:在类加载时就完成了初始化，但是加载比较慢，获取对象比较快。

饿汉模式是线程安全的，在类创建好一个静态对象提供给系统使用，懒汉模式在创建对象时不加上 synchronized，会导致对象的访问不是线程安全的。

（2） 核心答案讲解：

饿汉式:

```
public class Singleton{
    private static Singleton singleton = new Singleton ();
    private Singleton (){}
    public static Singleton getInstance(){return singletion;}
}
```

懒汉式:

```
public class Singleton{
    private static Singleton singleton = null;
    public static synchronized synchronized getInstance(){
        if(singleton==null){
            singleton = new Singleton();
        }
        return singleton;
    }
}
```

饿汉式是线程安全的,在类创建的同时就已经创建好一个静态的对象供系统使用,以后不在改变

懒汉式如果在创建实例对象时不加上 synchronized 则会导致对对象的访问不是线程安全的。

从实现方式来讲他们最大的区别就是懒汉式是延时加载，是在需要的时候才创建对象,而饿汉式在虚拟机启动的时候就会创建,

（3） 问题扩展

懒汉式不会预先创建对象,只在第一次调用时才创建对象，但是多线程并发执行的时候就很容易出现安全隐患，比如说第一个线程在判断 newInstance == null 时，还没有 new 出实例时，第二个线程也进来，判断的 newInstance 也是 null，然后也会 new 出实例，这就不符合单例模式了， 所以需要加锁。使用 synchronized 关键字加锁能解决安全问题，但是加锁同时会出现一个问题，那就是每次都需要判断锁，这样性能就会降低，所以为了提高性能，我们应该尽量减少锁判断的次数，加上双重判断。

```
//静态工厂方法、单锁
public synchronized static SingletonTest2 getInstance1() {
    if (single2==null) {
        single2 = new SingletonTest2();
    }
    return single2;
}

//静态工厂方法、双重锁
public static SingletonTest2 getInstance2() {
    if (single2==null) {
        synchronized (SingletonTest2.class) {
            if (single2==null) {
                single2 = new SingletonTest2();
            }
        }
    }
    return single2;
}
```

（4） 结合项目中的使用

懒汉式的特点是延迟加载，比如配置文件，采用懒汉式的方法。

题目 18：类加载机制有了解嘛？

（1） 问题分析：

Class 文件由类装载器装载后，在 JVM 中将形成一份描述 Class 结构的元信息对象，通过该元信息对象可以获知 Class 的结构信息：如构造函数，属性和方法等，Java 允许用户借由这个 Class 相关的元信息对象间接调用 Class 对象的功能。

虚拟机把描述类的数据从 class 文件加载到内存，并对数据进行校验，转换解析和初始化，最终形成可以被虚拟机直接使用的 Java 类型，这就是虚拟机的类加载机制。

（2） 核心答案讲解：

类装载器就是寻找类的字节码文件，并构造出类在 JVM 内部表示的对象组件。在 Java 中，类装载器把一个类装入 JVM 中，要经过以下步骤：

(1) 装载：查找和导入 Class 文件；

(2) 链接：把类的二进制数据合并到 JRE 中；

(a)校验：检查载入 Class 文件数据的正确性；

(b)准备：给类的静态变量分配存储空间；

(c)解析：将符号引用转成直接引用；

(3) 初始化：对类的静态变量，静态代码块执行初始化操作

Java 程序可以动态扩展是由运行期动态加载和动态链接实现的；比如：如果编写一个使用接口的应用程序，可以等到运行时再指定其实际的实现(多态)，解析过程有时候还可以在初始化之后执行；比如：动态绑定(多态)。

（3） 问题扩展

由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后 JVM 对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，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）。

（4） 结合项目中的使用

无。

题目 19：什么是事务？事务常见的并发问题及含义

(1) 问题分析：

考官主要相对于事务方面的考核，如什么是事务，事务都有哪些特性；事务并发问题如何解决；事务的隔离级别分别可以解决什么问题；jdbc，spring 如何实现事务控制；事务的应用场景等

(2) 核心答案讲解：

事务是指逻辑上的一组操作，组成这组操作的一系列操作要么全部成功，要么一个都不做。因此，事务的结束有两种，当事务中的所有操作全部成功执行时，事务提交。如果其中一个操作失败，将发生回滚操作，撤消到事务开始时的状态。

事务常见并发问题：

丢失更新：撤消一个事务时，把其它事务已提交的更新的数据覆盖了。

脏读：一个事务读到另一个事务未提交的更新数据。

幻读：一个事务执行两次查询，但第二次查询比第一次查询多出了一些数据行。

不可重复读：一个事务执行相同的查询两次或两次以上，但是每次都得到不同的数据。

(3) 问题扩展：

事务的特性：

原子性（Atomicity）：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行。

一致性（Consistency）：事务应确保数据库的状态从一个一致状态转变为另一个一致状态。一致状态的含义是数据库中的数据应满足完整性约束。

隔离性（Isolation）：多个事务并发执行时，一个事务的执行不应影响其他事务的执行。

持久性（Durability）：已被提交的事务对数据库的修改应该永久保存

事务的隔离级别及可以解决什么问题：

Serializable（串行化）8　可避免脏读、不可重复读、幻读的发生。但是效率最低

Repeatable read（可重复读）4 可避免脏读、不可重复读的发生。（最常用的 mysql 默认的级别就是 4）

Read committed（读已提交）2　可避免脏读的发生（oracle 默认）。

Read uncommitted（读未提交）　1　最低级别，任何情况都无法保证，但效率最高，最不安全。

锁

怎样实现事务的隔离呢？

隔离机制的实现必须使用锁。

锁的基本原理

a.当一个事务访问某个数据库资源时，如果执行的是 select 语句，必须为资源加上共享锁，如果执行的是 insert,update,delete 语句，必须为资源加上排他锁，这些锁锁定正在被操作的资源。

b.当第二个事务也要反问相同的资源时，如果执行的 select 语句，那么也必须为资源加上共享锁；如果执行的是 insert,update,或 delete 语句，也必须为资源加上排他锁。但此时第二个事务并非就立即能为资源加上锁，当第一个事务为资源加的是共享锁时，第二个事务能够为资源加上共享锁，但当第一个事务为资源加的是排他锁时，第二个事务必须等待第一个事务结束，才能为资源加上排他锁。

1.共享锁（s 锁）

共享锁用于读取数据操作，它允许其他事务同时读取锁定的资源，但不允许其他事务更新它。

2.排他锁（X 锁）

排他锁用于修改数据的场合，他锁定的资源，其他事务部能读取也不能修改。

3.更新锁（U 锁）

更新锁在更新操作初始化截断用来锁定可能要被修改的资源，从而避免使用共享锁造成的死锁现象。

表锁（table lock）
表锁是 MySQL 中最基本的锁策略，并且是开销最小的策略。表锁会锁定整张表。一个用户在对表进行写操作（插入、删除、更新等）前，需要先获得写锁，这会阻塞其他用户对该表的所有读写操作。当没有写锁时，其他读取的用户才能获得读锁，读锁之间是不相互阻塞的。

行级锁（row lock）
行级锁可以最大程度地支持并发处理（同时也带来了最大的锁开销）。在 InnoDB 和 XtraDB，以及其他一些存储引擎中实现了行级锁。行级锁只在存储引擎层实现，而 MySQL 服务器层没有实现。服务器层完全不了解存储引擎中的锁实现。

一次封锁 or 两段锁?
因为有大量的并发访问，为了预防死锁，一般应用中推荐使用一次封锁法，就是在方法的开始阶段，已经预先知道会用到哪些数据，然后全部锁住，在方法运行之后，再全部解锁。这种方式可以有效的避免循环死锁，但在数据库中却不适用，因为在事务开始阶段，数据库并不知道会用到哪些数据。
数据库遵循的是两段锁协议，将事务分成两个阶段，加锁阶段和解锁阶段（所以叫两段锁）

加锁阶段：在该阶段可以进行加锁操作。在对任何数据进行读操作之前要申请并获得 S 锁（共享锁，其它事务可以继续加共享锁，但不能加排它锁），在进行写操作之前要申请并获得 X 锁（排它锁，其它事务不能再获得任何锁）。加锁不成功，则事务进入等待状态，直到加锁成功才继续执行。

解锁阶段：当事务释放了一个封锁以后，事务进入解锁阶段，在该阶段只能进行解锁操作不能再进行加锁操作。

悲观锁
悲观锁是指假设并发更新冲突会发生，所以不管冲突是否真的发生，都会使用锁机制。悲观锁会完成以下功能：锁住读取的记录，防止其它事务读取和更新这些记录。其它事务会一直阻塞，直到这个事务结束.悲观锁是在使用了数据库的事务隔离功能的基础上，独享占用的资源，以此保证读取数据一致性，避免修改丢失。悲观锁可以使用 Repeatable Read 事务，它完全满足悲观锁的要求。

乐观锁
乐观锁不会锁住任何东西，也就是说，它不依赖数据库的事务机制，乐观锁完全是应用系统层面的东西。如果使用乐观锁，大多是基于数据版本（ Version ）记录机制实现。何谓数据版本？即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个 “version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。

(4) 结合项目中使用：
spring 的申明式事务：使得我们再也无需要去处理获得连接、关闭连接、事务提交和回滚等这些操作。再也无需要我们在与事务相关的方法中处理大量的 try…catch…finally 代码。

题目 20：你所了解的数据库优化都有哪些？

（1） 问题分析

考官主要是对数据库优化方面的考核，一般数据库优化分为性能和应用方面的，如你了解 sql 优化吗；百万数据怎么优化等

（2） 核心答案讲解

（1）、根据服务层面：配置 mysql 性能优化参数；

（2）、从系统层面增强 mysql 的性能：优化数据表结构、字段类型、字段索引、分表，分库、读写分离等等。

（3）、从数据库层面增强性能：优化 SQL 语句，合理使用字段索引。

（4）、从代码层面增强性能：使用缓存和 NoSQL 数据库方式存储，如 MongoDB/Memcached/Redis 来缓解高并发下数据库查询的压力。

（5）、减少数据库操作次数，尽量使用数据库访问驱动的批处理方法。

（6）、不常使用的数据迁移备份，避免每次都在海量数据中去检索。

（7）、提升数据库服务器硬件配置，或者搭建数据库集群。

（8）、编程手段防止 SQL 注入：使用 JDBC PreparedStatement 按位插入或查询；正则表达式过滤（非法字符串过滤）；

（3） 问题扩展

Sql 优化（4-5 条即可）：

1）应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

2）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

select id from t where num is null

可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：

select id from t where num=0

3）很多时候用 exists 代替 in 是一个好的选择

4）用 Where 子句替换 HAVING 子句 因为 HAVING 只会在检索出所有记录之后才对结果集进行过滤

5）select count(*) from table；这样不带任何条件的 count 会引起全表扫描，并且没有任何业务意义，是一定要杜绝的

索引

索引概念：对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。如果我们把一个表的内容认为是一本字典，那索引就相当于字典的目录

索引类型：

Oracle:

逻辑上：Single column 单行索引

Concatenated 多行索引

Unique 唯一索引

NonUnique 非唯一索引

Function-based 函数索引

Domain 域索引

物理上：

Partitioned 分区索引

NonPartitioned 非分区索引

B-tree：

Normal 正常型 B 树
Rever Key 反转型 B 树
Bitmap 位图索引

MySQL 索引分为普通索引、唯一索引、主键索引、组合索引、全文索引

何时使用索引

（1）主键，unique 字段；
（2）和其他表做连接的字段需要加索引；
（3）在 where 里使用＞，≥，＝，＜，≤，is null 和 between 等字段；
（4）使用不以通配符开始的 like，where A like 'China%'；
（5）聚集函数 MIN()，MAX()中的字段；
（6）order by 和 group by 字段；

索引何时失效

（1）组合索引未使用最左前缀，例如组合索引（A，B），where B=b 不会使用索引；
（2）like 未使用最左前缀，where A like '%China'；
（3）搜索一个索引而在另一个索引上做 order by，where A=a order by B，只使用 A 上的索引，因为查询只使用一个索引 ；
（4）or 会使索引失效。如果查询字段相同，也可以使用索引。例如 where A=a1 or A=a2（生效），where A=a or B=b（失效）
（5）如果列类型是字符串，要使用引号。例如 where A='China'，否则索引失效（会进行类型转换）；
（6）在索引列上的操作，函数（upper()等）、or、!=(<>)、not in 等；

（4） 结合项目中使用

1.常用但不经常修改的字段建索引(譬如商品表的商品名称等字段)，达到检索速度增快，用户体验度增高的目的

2.用 mycat 进行分库分表

垂直拆分是基于数据库中的"列"进行，某个表字段较多，可以新建一张扩展表，将不经常用或字段长度较大的字段拆分出去到扩展表中。例如用户表，在字段很多的情况下（例如一个大表有 100 多个字段），通过"大表拆小表"，更便于开发与维护，也能避免跨页问题

水平分表

水平切分分为库内分表和分库分表，是根据表内数据内在的逻辑关系，将同一个表按不同的条件分散到多个数据库或多个表中，每个表中只包含一部分数据，从而使得单个表的数据量变小，达到分布式的效果（如订单表）

参考文档：

https://blog.csdn.net/yzllz001/article/details/54848513/
https://blog.csdn.net/weixin_39420024/article/details/80040549
https://blog.csdn.net/boonya/article/details/60962774
https://www.cnblogs.com/sunny3096/p/8595058.html
https://www.cnblogs.com/butterfly100/p/9034281.html

题目 21：mysql 和 oracle 的区别？（oracle 是在 ssm 之后讲的）【了解】

（1） 问题分析：

考官主要是想考察对数据库的理解以及熟练程度，什么情况下使用 mysql，什么情况下使用 Oracle，各有什么优缺点

（2） 核心答案讲解：

1. Oracle 是大型数据库而 Mysql 是中小型数据库，Oracle 市场占有率达 40%，Mysql 只有 20%左右，同时 Mysql 是开源的而 Oracle 价格非常高。
2. Oracle 支持大并发，大访问量，是 OLTP 最好的工具。
3. 安装所用的空间差别也是很大的，Mysql 安装完后才 152M 而 Oracle 有 3G 左右，且使用的时候 Oracle 占用特别大的内存空间和其他机器性能。
4. Oracle 也 Mysql 操作上的区别

①主键

Mysql 一般使用自动增长类型，在创建表时只要指定表的主键为 auto increment,插入记录时，不需要再指定该记录的主键值，Mysql 将自动增长；Oracle 没有自动增长类型，主键一般使用的序列，插入记录时将序列号的下一个值付给该字段即可；只是 ORM 框架是只要是 native 主键生成策略即可。

②单引号的处理

MYSQL 里可以用双引号包起字符串，ORACLE 里只可以用单引号包起字符串。在插入和修改字符串前必须做单引号的替换：把所有出现的一个单引号替换成两个单引号。

③翻页的 SQL 语句的处理

MYSQL 处理翻页的 SQL 语句比较简单，用 LIMIT 开始位置, 记录个数；ORACLE 处理翻页的 SQL 语句就比较繁琐了。每个结果集只有一个 ROWNUM 字段标明它的位置, 并且只能用 ROWNUM<100, 不能用 ROWNUM>80

④ 长字符串的处理

长字符串的处理 ORACLE 也有它特殊的地方。INSERT 和 UPDATE 时最大可操作的字符串长度小于等于 4000 个单字节, 如果要插入更长的字符串, 请考虑字段用 CLOB 类型，方法借用 ORACLE 里自带的 DBMS_LOB 程序包。插入修改记录前一定要做进行非空和长度判断，不能为空的字段值和超出长度字段值都应该提出警告, 返回上次操作。

⑤空字符的处理

MYSQL 的非空字段也有空的内容，ORACLE 里定义了非空字段就不容许有空的内容。按 MYSQL 的 NOT NULL 来定义 ORACLE 表结构, 导数据的时候会产生错误。因此导数据时要对空字符进行判断，如果为 NULL 或空字符，需要把它改成一个空格的字符串。

⑥字符串的模糊比较

MYSQL 里用 字段名 like '%字符串%',ORACLE 里也可以用 字段名 like '%字符串%' 但这种方法不能使用索引, 速度不快。

⑦Oracle 实现了 ANSII SQL 中大部分功能，如，事务的隔离级别、传播特性等而 Mysql 在这方面还是比较的弱。

（3） 问题扩展

数据库优化技术以及 sql 优化知道那些？

会写存储过程吗？

结合项目中使用

题目 22：左链接和右链接的语法及区别？

（1） 问题分析：
考官主要是对常用 sql 语法的考察
（2） 核心答案讲解：
1.左连接
数据表 A 中的记录为主循环体，依次匹配数据表 B 中的记录，如果数据表 A 中连接字段 Aid 的值，在数据表 B 中没有 Bnameid 值与之对应，则右侧以 null 代替。结果集：公共部分记录集 C＋表 A 记录集 A1。语句如下：select * from A Left JOIN B ON A.Aid=B.Bnameid
2. 右链接
数据表 B 中的记录为主循环体，依次匹配数据表 A 中的记录，如果数据表 B 中连接字段 Bnameid 的值，在数据表 A 中没有 Aid 值与之对应，则左侧以 null 代替。结果集：公共部分记录集 C＋表 B 记录集 B1。语句如下：select * from A Right JOIN B ON A.Aid=B.Bnameid
（3） 问题扩展
内连接，全连接各是什么意思？
（4） 结合项目中使用
在 sql 编写的时候什么对语法的选择

JavaWeb 阶段

题目 23：cookie 和 session 的区别与联系

(1) 问题分析
考官主要是针对你对 javaweb 会话跟踪技术的考核， 比如：cookie 被用户禁用怎么办？
(2) 核心答案解析
1、cookie 数据存放在客户的浏览器上，session 数据放在服务器上。
2、cookie 并不是很安全，别人可以分析存放在本地的 cookie 并进行 cookie 欺骗，考虑到安全应当使用 session。
3、session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用 cookie。
4、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。
5、可以考虑将登陆信息等重要信息存放为 session，其他信息如果需要保留，可以放在 cookie 中。
6、session 会在浏览器关闭或者一段时间内销毁而 cookie 将持久化的存放在客户端

一般情况下，session 生成的 sessionid 都是保存在 cookie 中。

(3) 问题扩展

cookie 被用户禁用怎么办？可以使用 URL 地址重写是对客户端不支持 Cookie 的解决方案。URL 地址重写的原理是将该用户 Session 的 id 信息重写到 URL 地址中。服务器能够解析重写后的 URL 获取 Session 的 id。这样即使客户端不支持 Cookie，也可以使用 Session 来记录用户状态。

(4) 结合项目使用

1.判断用户是否登陆过网站，以便下次登录时能够直接登录。如果我们删除 cookie，则每次登录必须从新填写登录的相关信息。

2.另一个重要的应用是“购物车”中类的处理和设计。用户可能在一段时间内在同一家网站的不同页面选择不同的商品，可以将这些信息都写入 cookie，在最后付款时从 cookie 中提取这些信息，当然这里面有了安全和性能问题需要我们考虑了。

题目 24：在 HTTP 请求中，什么情况下我们会选择 post 方式而非 get？反之也是如何考虑的？

(1) 问题分析

考官主要是针对 http 协议请求方式的考核，比如：http 协议其他的请求方式！GET 方法和 POST 方法本质上的区别？等

(2) 核心答案解析

GET 一般用于获取和查询资源信息，指定的资源经服务器端解析后返回响应内容，必要时，可以将查询字符串参数追加到 URL 末尾，以便将信息发送给服务器。POST 一般用于更新资源信息，通常会用来传输实体的本体，用 GET 方法也可以传输实体的主体，但一般不用 GET 方法进行传输，而是用 POST 方法，虽然 GET 方法和 POST 方法很相似，但是 POST 的主要目的并不是获取响应的主体内容。

(3) 问题扩展

http 协议其他的请求方式?

HEAD：获得报文首部，GET 方法有实体，HEAD 方法无实体。

PUT：传输文件，就像 FTP 协议的文件上传一样，要求在请求报文的主体中包含文件内容，然后保存在请求 URI 指定的位置，存在安全问题，故一般不用。

DELETE：删除文件或资源，与 PUT 方法相反，按 URI 删除指定资源

OPTIONS：询问支持的方法，客户端询问服务器可以提交哪些请求方法

TRACE：追踪路径，让 Web 服务器端将之前的请求通信还给客户端的方法

CONNECT：要求用隧道协议连接代理，实现用隧道协议进行 TCP 通信。

GET 方法和 POST 方法本质上的区别？

1、GET 方法用于信息获取，它是安全的（安全：指非修改信息，如数据库方面的信息），而 POST 方法是用于修改服务器上资源的请求；

2、GET 请求的数据会附在 URL 之后，而 POST 方法提交的数据则放置在 HTTP 报文实体的主体里，所以 POST 方法的安全性比 GET 方法要高；

3、GET 方法传输的数据量一般限制在 2KB，而 Chrome，FireFox 浏览器理论上对于 URL 是没有限制的，它真正的限制取决于操作系统本身；POST 方法对于数据大小是无限制的，真正影响到数据大小的是服务器处理程序的能力。

(4) 结合项目使用

在项目使用 RESTful 架构风格进行开发，GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源，这样就统一了数据操作的接口，仅通过 HTTP 方法，就可以完成对数据的所有增删查改工作。

题目 25：JSP 的九大内置对象及作用分别是什么？【了解】

（1） 问题分析：

考官主要想考核一下你前端知识的基本功，与之同类的问题可能还会问一下四大作用域以及 OSI 七层物理模型以及常用标签、作用等基础知识。

（2） 核心答案讲解：

1、request 对象

request 对象是 javax.servlet.httpServletRequest 类型的对象。 该对象代表了客户端的请求信息，主要用于接受通过 HTTP 协议传送到服务器的数据。（包括头信息、系统信息、请求方式以及请求参数等）。request 对象的作用域为一次请求。

2、response 对象

response 代表的是对客户端的响应，主要是将 JSP 容器处理过的对象传回到客户端。response 对象也具有作用域，它只在 JSP 页面内有效。

3、session 对象

session 对象是由服务器自动创建的与用户请求相关的对象。服务器为每个用户都生成一个 session 对象，用于保存该用户的信息，跟踪用户的操作状态。session 对象内部使用 Map 类来保存数据，因此保存数据的格式为“Key/value”。 session 对象的 value 可以使复杂的对象类型，而不仅仅局限于字符串类型。

4、application 对象

application 对象可将信息保存在服务器中，直到服务器关闭，否则 application 对象中保存的信息会在整个应用中都有效。与 session 对象相比，application 对象生命周期更长，类似于系统的“全局变量”。

5、out 对象

out 对象用于在 Web 浏览器内输出信息，并且管理应用服务器上的输出缓冲区。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据，为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

6、pageContext 对象

pageContext 对象的作用是取得任何范围的参数，通过它可以获取 JSP 页面的 out、request、reponse、session、application 等对象。pageContext 对象的创建和初始化都是由容器来完成的，在 JSP 页面中可以直接使用 pageContext 对象。

7、config 对象

config 对象的主要作用是取得服务器的配置信息。通过 pageConext 对象的 getServletConfig() 方法可以获取一个 config 对象。当一个 Servlet 初始化时，容器把某些信息通过 config 对象传递给这个 Servlet。 开发者可以在 web.xml 文件中为应用程序环境中的 Servlet 程序和 JSP 页面提供初始化参数。

8、page 对象

page 对象代表 JSP 本身，只有在 JSP 页面内才是合法的。 page 隐含对象本质上包含当前 Servlet 接口引用的变量，类似于 Java 编程中的 this 指针。

9、exception 对象

exception 对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的 JSP 页面中使用该对象将无法编译 JSP 文件。excepation 对象和 Java 的所有对象一样，都具有系统提供的继承结构。exception 对象几乎定义了所有异常情况。在 Java 程序中，可以使用 try/catch 关键字来处理异常情况； 如果在 JSP 页面中出现没有捕获到的异常，就会生成 exception 对象，并把 exception 对象传送到在 page 指令中设定的错误页面中，然后在错误页面中处理相应的 exception 对象。

（3） 问题扩展

Jsp 四大作用域及其作用：

四个作用域从大到小：appliaction>session>request>page

application：全局作用范围，整个应用程序共享.生命周期为：应用程序启动到停止；

session：会话作用域，当用户首次访问时，产生一个新的会话，以后服务器就可以记住这个会话状态；

request：请求作用域，就是客户端的一次请求；

page：一个 JSP 页面。

以上作用范围使越来越小， request 和 page 的生命周期都是短暂的，他们之间的区别就是：一个 request 可以包含多个 page 页(include，forward)。

（4） 结合项目中使用

举例：request 对象

1、获取数据

getParameter；；接收请求参数的，

2、对全部数据进行再编码

public byte[] getBytes(“encoding”)
例如以下实例将 byte 数组编码转换

```
<%@ page contentType="text/html";charset=gbk"%>
<html>
	<body>
		<%
			//接收内容
			String name=request.getParameter("uname");
			byte[] b=name.getBytes("ISO8859-1");
			name=new String(b);
			String name= new String(request.getParameter("uname").get-
Bytes("ISO8859-1"));
		%>
		<h1>输入内容为:<%=name%></h1>
	</body>
</html>
```

3、设置统一的请求编码

public void setCharacterEncoding(String env) throws UnsunpportedEncodingExcep-
tion
例如以下设置

```
<%@ page contentType="text/html";charset=gbk"%>
<html>
	<body>
		<%
			//接收内容
			request.setCharacterEncoding("GBK");
			String name= request.getParameter("uname");
		%>
		<h1>输入内容为：<%=name%></h1>
	</body>
</html>
```

其他举例 Demo：https://download.csdn.net/download/a_blackmoon/10940085

题目 26：servlet 的生命周期及常用方法？

（1） 问题分析：
考官主要想考核 Java 基本功的掌握和应用。一方面，是 servlet 的生命周期；另一方面，则是其单例非线程安全对于安全这块的一个处理。

（2） 核心答案讲解：

1. init()方法

在 Servlet 的生命周期中，仅执行一次 init()方法，它是在服务器装入 Servlet 时执行的，可以配置服务器，以在启动服务器或客户机首次访问 Servlet 时装入 Servlet。无论有多少客户机访问 Servlet，都不会重复执行 init()；

2. service()方法

它是 Servlet 的核心，每当一个客户请求一个 HttpServlet 对象，该对象的 Service()方法就要调用，而且传递给这个方法一个“请求”（ServletRequest）对象和一个“响应”（ServletResponse）对象作为参数。在 HttpServlet 中已存在 Service()方法。默认的服务功能是调用与 HTTP 请求的方法相应的 do 功能。

3. destroy()方法

仅执行一次，在服务器端停止且卸载 Servlet 时执行该方法，有点类似于 C++的 delete 方法。一个 Servlet 在运行 service()方法时可能会产生其他的线程，因此需要确认在调用 destroy()方法时，这些线程已经终止或完成。

（3） 问题扩展

如何解决 servlet 线程安全：

第一种，继承 SingleThreadModel 但是这样每次都会创建一个新的 servlet 实例，但这样消耗服务器的内存，降低了性能，并且这个接口现在已经过时了，不推荐使用。

第二种：我们尽量避免使用全局变量，就我个人而言，我比较喜欢使用这种方法。

第三种，我们可以通过使用 ThreadLocal， 内部结构是一个 Map 结构，用当前线程作为 key,他会创建多个副本。get,set 方法

第四种，我们当然还可以来加锁，进行解决线程问题。

而且我还知道，向我们这种常用的 MVC 框架，struts1，spring 这些 MVC 框架，都是基于 servlet 发展而来的，就比如 struts1 的核心总控制器是 ActionServlet， 而 springMVC 的前端总控制器是 dispatchServlet，在项目我们曾经用 serlet 来生成 图片验证码的，防止用户进行暴力破解。

（4） 结合项目中使用

servlet 的配置文件 web.xml

```
<servlet>
    <servlet-name>ImageCodeServlet</servlet-name>       <servlet-
class>org.leopard.code.ImageCodeServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
        <servlet-name>ImageCodeServlet</servlet-name>
        <url-pattern>/d</url-pattern>
</servlet-mapping>
```

描述：

在 web.xml 中，首先需要写一个 servlet 标签，servlet 标签中有两个子标签，一个叫 servlet-name，这个 name 可以随便起，但是要保证唯一性，除此之外，在这个 servlet-name 下有一个 servlet-class，这个 servlet-class 对应的就是我后台提高服务的 servlet，除此之外还有一个 servlet-mapping，这个里边首先有一个 servl-name。，这个 servl-name 首先要保证和上边的 servlet-name 保持一致，除此之外还有一个 url-pattern，这是一个虚拟路径，是用来发送请求的 url 地址。

Servlet 的生命周期是由 Servlet 容器来控制的，它始于装入 Web 服务器的内存时，并在终止或重新装入 Servlet 时结束。

在代码中，Servlet 生命周期由接口 javax.servlet.Servlet 定义。所有的 Java Servlet 必须直接或间接地实现 javax.servlet.Servlet 接口，这样才能在 Servlet Engine 上运行。

题目 27：转发和重定向区别？

(1) 问题分析

考官主要是针对你对 javaweb 页面跳转技术的考核，如：获取 servlet 转发和响应重定向的方式？

(2) 核心答案解析

1、重定向是浏览器发送请求并受到响应以后再次向一个新地址发请求；转发是服务器受到请求后为了完成响应转到一个新的地址。

2、重定向中有两次请求对象，不共享数据；转发只产生一次请求对象且在组件间共享数据。

3、重定向后地址栏地址改变；而转发则不会。

4、重定向的新地址可以是任意地址；转发的新地址必须是同一个应用内的某地址。

(3) 问题扩展

获取 servlet 的转发和响应重定向的方式？

转发的方法：

1) 通过 HttpServletRequest 的 getRequestDispatcher()方法获得

2) 通过 ServletContext 的 getRequestDispatcher()方法获得

重定向的方法：HttpServletResponse 的 sendRedirect()方法。

(4) 结合项目使用

重定向可以实现图片的异步上传。

一般访问 web-inf 下面文件，只能通过转发来实现。

题目 28：ajax 书写方式及内部主要参数都有哪些

(1) 问题分析

考官主要是针对你对创建交互式网页应用技术进行考核，比如：ajax 的优缺点？

(2) 核心答案解析

使用$.ajax(),$.getJson()。

主要参数：

1. url：要求为 string 类型的参数，发送请求的地址。
2. Type:要求为 Stirng 类型，请求方式 post 或 get。
3. Timeout:要求为 number 类型，设置请求超时时间（毫秒）。
4. Async:要求为 boolean 类型，异步为 true，同步为 false。
5. Cache：要求为 boolean 类型，从游览器缓存中是否加载信息。
6. Data:要求为 object 或 stirng 类型，发送到服务器的数据。
7. DataType:要求为 String 类型，预期服务器返回的数据类型。
8. BeforeSend：要求为 function 类型的参数。

(3) 问题扩展

ajax 的优缺点？

优点：减轻服务器的负担，按需取数据，最大程度的减少冗余请求，局部刷新页面，减少用户心理和实际的等待时间，带来更好的用户体验。

缺点：ajax 大量的使用了 javascript 和 ajax 引擎,这些取决于浏览器的支持,在编写的时候考虑对浏览器的兼容性。AJAX 只是局部刷新,所以页面的后退按钮是没有用的。

(4) 结合项目使用

城市三级联动是使用 ajax，动态实现页面展示省、市、区等。

题目 29：Jquery 常用选择器都有哪些？

（1）问题分析

（2）核心问题讲解

（3）问题扩展

（4）结合项目中使用

题目 30：JSP 与 Servlet 的区别？【了解】

/jsp 和 servlet 有哪些相同点和不同点，他们 之间的联系是什么？

（1）问题分析

（2）核心问题讲解

（3）问题扩展

（4）结合项目中使用

题目 31：拦截器和过滤器区别

（1） 问题分析：
考官主要考察拦截器和过滤器在应用场景和基本原理方面的差别

（2） 核心答案讲解：
1. 拦截器是基于 java 的反射机制的，而过滤器是基于函数回调。
2. 拦截器不依赖与 servlet 容器，过滤器依赖与 servlet 容器。
3. Filter 不能够使用 Spring 容器资源，拦截器是一个 Spring 的组件，归 Spring 管理，配置在 Spring 文件中，因此能使用 Spring 里的任何资源
4. Filter 定义在 web.xml 中

（3） 问题扩展
Spring 的 Interceptor(拦截器)与 Servlet 的 Filter 有相似之处，比如二者是 AOP 编程思想的体现，都能实现权限检查、日志记录等。

（4） 结合项目中使用
Servlet 中的过滤器 Filter 是实现了 javax.servlet.Filter 接口的服务器端程序，主要的用途是设置字符集、控制权限、控制转向。拦截器可以加载用户信息，判断用户的访问权限。

题目 32：一次完整的 http 请求是什么样的?

（1）问题分析：

考官主要考察 HTTP 通信机制，在一次完整的 HTTP 通信过程中，Web 浏览器与 Web 服务器之间有哪些步骤。

（2）核心答案讲解：

一次完整的 HTTP 请求需要的 7 个步骤：

1：建立 TCP 连接

2：web 浏览器想 web 服务器发送请求命令

3：web 浏览器发送请求头信息

4：Web 服务器应答

5：Web 服务器发送应答头信息

6：Web 服务器向浏览器发送数据

7：Web 服务器关闭 TCP 连接

（3）问题扩展

在浏览器的地址栏输入 www.itcast.com ，然后回车，回车这一瞬间到看到页面到底发生了什么呢？

域名解析 --> 发起 TCP 的 3 次握手 --> 建立 TCP 连接后发起 http 请求 --> 服务器响应 http 请求，浏览器得到 html 代码 --> 浏览器解析 html 代码，并请求 html 代码中的资源（如 js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

（4）结合项目中使用

在很多场景下都需要用到 java 代码来发送 http 请求：如和短信后台接口的数据发送，发数据到微信后台接口中，使用 httpClient 模拟 http 请求

题目 33：ajax 提交请求 默认是 异步还是同步 怎么改成同步?

（1）问题分析：

考官主要考察 ajax 请求参数，async 属性的了解，和同步请求异步请求的差别。

（2）核心答案讲解：

默认设置为 true，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。注意，同步请求将锁住浏览器，用户其他操作必须等待请求完成才可以执行

（3）问题扩展

同步请求和异步请求的差别

一.什么是同步请求：(false)同步请求即是当前发出请求后，浏览器什么都不能做，必须得等到请求完成返回数 据之后，才会执行后续的代码，相当于是排队，前一个人办理完自己的事务，下一个人才能接着办。也就是说，当 JS 代码加载到当前 AJAX 的时候会把页面里所有的代码停止加载，页面处于一个假死状态，当这个 AJAX 执行完毕后才会继续运行其他代码页面解除假死状态。

二. 什么是异步请求：(true) 异步请求就当发出请求的同时，浏览器可以继续做任何事，Ajax 发送请求并不会影响页面的加载与用户的操作，相当于是在两条线上，各走各的，互不影响。

（4）结合项目中使用

```
$.ajax({
    url:"url",
    type:"post",
    async:false,
    success:function(){代码}});
```

题目 34：你的项目中使用过那些 JSTL 标签？【了解】

（1）问题分析：

考官主要考察在 jsp 中对标签库的使用能力，有没有实际开发经验。

（2）核心答案讲解：

常见的有 Core 标签库里面的，c 标签的使用。有输出标签<c:out>，判读标签<c:if>和迭代标签<c:forEach>，多重判断标签<c:choose>

（4）问题扩展

jstl 标签是为了解决 EL 表达式不能循环取出集合对象的问题，是建立在 EL 表达式基础上的语言，两者之间是互通的，属于 jsp 外部的一个标准标签，库导入标签格式为：<%@ taglib uri=”http://java.sun.com/jsp/jstl/core” prefix=”c” %>

（4）结合项目中使用

如使用<c:forEach>,循环打印用户的姓名和年龄

```
<c:forEach items="${userList}" var="user" varStatus="status"
begin="0" end="${userList.size}" step="1" >
    //循环体
    <c:out value="${ user.name }"></c:out>
    <c:out value="${ user.age }"></c:out>
</c:forEach>
```

题目 35：JSP 常用的动作标签及作用【了解】

（1）问题分析：

考官主要考察对 jsp 的熟练程度，对常用动作标签的熟悉程度，部分同学肯定会遗忘动作标签的概念。

（2）核心答案讲解：

常用标签举例：

1.<jsp:forward>用于请求转发

2.<jsp:param> 用于页面传递数据

3.<jsp:include>这个标签的作用与 jsp 中 include 指令功能是一样的，区别是 include 标签为动态包含。

（3）问题扩展

Jsp 中静态包含(include 指令)与动态包含(include 标签)区别?
静态包含包含的是内容，而动态包含包含的是结果。
静态包含不可以传递参数 ,而动态包含可以传递参数

（4）结合项目中使用
<jsp:forward page="login.jsp"></jsp:forward>，将请求转发到 login.jsp 页面

题目 36：JSP 四大作用域及请求范围 【了解】

（1）问题分析：
考官主要考察在 jsp，有没有实际开发经验。

（2）核心答案讲解：
JSP 四大作用域分别为：page， request ，session， application 。
第一个作用域是 page，他只在当前页面有效，也就是用户请求的页面有效
第二个作用域是 request，他在当前请求中有效
第三个作用域是 session，他在当前回话中有效
第四个作用域是 application，他在所有的应用程序中都有效

（3）问题扩展
page，他只在当前页面有效，也就是用户请求的页面有效，当当前页面关闭或转到其他页面时，page 对象将在响应回馈给客户端后释放。
request 可以通过 setAttribute()方法实现页面中的信息传递，也可以通过 forward()方法进行页面间的跳转，需要注意的是 request 是转发不是重定向，转发相对于浏览器来说是透明的，也就是无论页面如何跳转，地址栏上显示的依旧是最初的地址。
session，他在当前回话中有效。当一个台电脑上的同一浏览器对服务器进行多次访问时，在这多次访问之间传递的信息就是 session 作用域的范围。它从浏览器发出第一个 HTTP 请求即可认为会话开始，但是会话结束的时间是不确定的，因为在浏览器关闭时并不会通知服务器，一般 Tomcat 设置的默认时间为 120 分钟，也可以通过 setMaxInactiveInterval(int)方法进行设置，或是通过 invalidate()方法强制结束当前 会话。
application，他在所有的应用程序中都有效，也就是当服务器开始到服务器结束这段时间，application 作用域中存储的数据都是有效的，同样可以通过 setAttribute 赋值和 getAttribute 取值。

题目 37：如何防止表单重复提交问题

（1）问题分析：
考察表单重复提交的场景与解决方式。

（2）核心答案讲解：
网络延迟时，重复点击提交按钮，有可能发生重复提交表单问题。
解决方案：
1.数据库主键唯一。
2.提交成功后重定向。
3.使用 JavaScript 解决，使用标记位，提交后隐藏或不可用提交按钮。

（3）问题扩展

使用 Session 解决：

生成唯一的 Token 给客户端，客户端第一次提交时带着这个 TOken,后台与 Session 中的进行对比。一样则提交成功并清除 Session 中的 Token。不一样则提交失败。

（4）结合项目中使用

```
<script type="text/javascript">
// 标志位
    var isCommitted = false;
    function dosubmit(){
        if(isCommitted==false){
          //提交表单后，将表单是否已经提交标识设置为 true;
            isCommitted = true;
            return true;
        }else{
              return false;// 返回 false 那么表单将不提交;
        }
    }
</script>
```

题目 38：分别说出 http,https,ftp,talnet 的默认端口

（1）问题分析：

考察面试人对常见协议和端口号的熟悉程度。

（2）核心答案讲解：

http：80

https：443

ftp：21

telnet：23

（3）问题扩展

分布式多服务项目时往往需要修改服务的端口号。

（4）结合项目中使用

```
<Connector port="9001" protocol="HTTP/1.1" connectionTimeout="20000" redi-
rectPort="8443" />
```

题目 39：常见的 http 返回状态码（200,301，302,400）

(1)问题分析：

当客户端通过 http 协议向服务端发起请求时，都会返回一个状态码表示请求结果。本题主要考核的内容是对 http 请求结果的了解

(2)核心答案讲解

100：这个状态码是告诉客户端应该继续发送请求，这个临时响应是用来通知客户端的，部分的请求服务器已经接受，但是客户端应继续发送求请求的剩余部分，如果请求已经完成，就忽略这个响应，而且服务器会在请求完成后向客户发送一个最终的结

果。

200：返回 200 表示请求响应成功

202：返回 202 表示请求已经被受理还未做出响应。

400：表示请求无效，常见的情况是请求参数有误，http 头构建错误等 404：访问不到资源。

500:服务器后端错误。

我们 java 项目开发中常用到的状态码有 200、404、和 500。

(3)问题扩张

完整的 http 状态码注册表由互联网号码分配局负责维护的。状态码共三位，一共分为 5 种，从 1...5 开始。1 开头的状态码是消息类型的。2 开头的状态码表示成功。3 开头的状态码表示需要重定向，也就是需要用户进一步操作。4 开头的状态码表示请求错误。5 开头的状态吗表示服务器错误。

(4)结合项目使用

http 协议是 javaweb 开发的基石，只要开发 javaWEB 项目，就无法避免和 http 打交道，通过获取 http 的状态码可以知道请求的结果。

题目 40：TCP 和 UDP 区别，你对 HTTP 协议的理解

(1)问题分析：

题目的意思很直接明确，就是考你 TCP 和 UDP 协议以及 HTTP 协议。Http 协议和 tcp 以及 UDP 不同，所以单独分析。

(2)核心答案讲解

TCP 协议提供安全可靠的网络传输服务，它是一种面向连接的服务，类似于打电话，必须先拨号， 通道内传输。

而 UDP 协议是一种数据报协议，它传输的数据是分组报文，它是无连接的，不需要和目标通信方建立连接，类似于写信，所以它的传输不保证安全可靠。但适合大数据量的传输。

Http 协议是超文本传输协议，是一种相对于 TCP 来说更细致的协议，TCP 以及 UDP 协议规范的是网络设备之间的通信规范，HTTP 是在 TCP 协议的基础上针对用户服务的协议，用户服务具体体现在应用程序之间的交互，比如我们的 javaweb 中客户端服务端体系就要用 http 协议来规范通信。

(3)问题扩张

计算机网络中有这样一个术语，TIP/IP 网络参考模型，整个计算机网络系统被分为 4 层，从底层到顶层分别为：网络接口层，网际层，传输层，应用层，每一层的通信都有专门的协议，底层是为上一层提供服务的。我们的 TCP 以及 UDP 是传输层的协议，而 HTTP 协议是处在应用层的协议。

(4)结合项目使用

TCP 和 UDP 在开发中我们很少见到，但是网络底层都有它们的影子，正常的会话级别的服务：如客户端服务器体系底层就说基于 TCP 协议的。而邮件发送，短信发送等底层使用的是 UDP 协议。

HTTP 协议，客户端/服务器体系的程序都使用 HTTP 协议来规范通信。

题目 41：json 数据的格式是什么?

(1)问题分析：

本题考查的是 json 数据格式写法

(2)核心答案讲解

Json 数据格式结构和数组相似，是这样的：（1）数据在名称/值对中；（2）数据由逗号分隔；（3）花括号保存对象；（4）方括号保存数组，如：

[{“属性名”：“值”,“属性名”:“值”}，{“属性名”：“值”,“属性名”:“值”}]

(3)问题扩张

Json 的最初出现是专门为 javaScript 准备的，它可以把 JS 对象和字符串之间来回转换，来应对对象数据的传输，需要注意的是 json 中的值是有限制的，对于复合类型来说，只能放数组或者对象，不能是正则、函数或者日期；对于简单类型来说，只能是字符串、数值（必须是十进制）、布尔值和 null。

(4)结合项目使用

Json 在网络开发中有非常广泛的用途，但可以归纳为一句：可以用于接口开发及调用中使用的数据格式。一来用于服务端和 javascript 之间的数据交互，二来可以用于跨域传输数据的数据格式。

在项目中，前后端交互、接口开发中很多都使用 json 来作为数据传输格式。

题目 42：xml 的解析方式有哪些？

(1)问题分析：

本题考查的是对 xml 数据格式的解析方式。

(2)核心答案讲解

1）DOM(JAXP Crimson 解析器)

DOM 是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准。DOM 是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。DOM 以及广义的基于树的处理具有几个优点。首先，由于树在内存中是持久的，因此可以修改它以便应用程序能对数据和结构作出更改。

2）SAX

SAX 处理的优点非常类似于流媒体的优点。分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中。这对于大型文档来说是个巨大的优点。事实上，应用程序甚至不必解析整个文档；

它可以在某个条件得到满足时停止解析。一般来说，SAX 还比它的替代者 DOM 快许多。

3）JDOM

JDOM 与 DOM 主要有两方面不同。首先，JDOM 仅使用具体类而不使用接口。这在某些方面简化了 API，但是也限制了灵活性。第二，API 大量使用了 Collections 类，简化了那些已经熟悉这些类的 Java 开发者的使用。

4）DOMJ

DOM4J 是一个非常非常优秀的 Java XML API，具有性能优异、功能强大和极端易用使用

的特点，同时它也是一个开放源代码的软件，是目前 XML 主流的解析方式。

(3)问题扩张

SAX 是基于事件驱动的解析方式的，DOM，JDOM,DOmJ4 是基于文档结构解析的，4 中方式性能对比如下：

DOM4J 性能最好，连 Sun 的 JAXM 也在用 DOM4J。目前许多开源项目中大量采用 DOM4J，例如大名鼎鼎的 Hibernate 也用 DOM4J 来读取 XML 配置文件。如果不考虑可移植性，那就采用 DOM4J。

JDOM 和 DOM 在性能测试时表现不佳，在测试 10M 文档时内存溢出。在小文档情况下还值得考虑使用 DOM 和 JDOM。

SAX 表现较好，这要依赖于它特定的解析方式－事件驱动。一个 SAX 检测即将到来的 XML 流，但并没有载入到内存(当然当 XML 流被读入时，会有部分文档暂时隐藏在内存中)。

(4)结合项目使用

目前 java 开发中主要用的解析方式是 DOM4J,来解析基于 XML 的配置文件，很多常用的中间件通过 spring 集成时都通过 DOM4J 解析 XML，我们常用的 SSM 框架，那些配置文件都是基于 DOM 解析的。另外在 webservice 接口开发中，数据传输使用 XML 格式，这时，我们可以自己选择这几种解析方式来解析 XML 数据。

SSM 框架阶段

题目 43：Spring 是如何管理事务的，事务管理机制？以及隔离级别？

（1）问题分析

主要考察事务的 4 种隔离级别，如何使用 spring 进行事务管理，声明式事务管理

（2）核心问题讲解

数据隔离级别分为不同的四种：

1、Serializable ：最严格的级别，事务串行执行，资源消耗最大；

2、REPEATABLE READ ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了“脏读取”和“不可重复读取”的情况，但是带来了更多的性能损失。

3、READ COMMITTED ：大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了“脏读取”。该级别适用于大多数系统。

4、Read Uncommitted ：保证了读取过程中不会读取到非法数据。

Spring 的事务机制包括声明式事务和编程式事务。

编程式事务管理：Spring 推荐使用 TransactionTemplate，实际开发中使用声明式事务较多。

声明式事务管理：将我们从复杂的事务处理中解脱出来，获取连接，关闭连接、事务提交、回滚、异常处理等这些操作都不用我们处理了，Spring 都会帮我们处理。

声明式事务管理使用了 AOP 面向切面编程实现的，本质就是在目标方法执行前后进行拦截。在目标方法执行前加入或创建一个事务，在执行方法执行后，根据实际情况选择提交或是回滚事务。

（3）问题扩展

Spring 的事务管理器

Spring 并不直接管理事务，而是提供了多种事务管理器，它们将事务管理的职责委托给 JTA 或其他持久化机制所提供的平台相关的事务实现。每个事务管理器都会充当某一特定平台的事务实现的门面，这使得用户在 Spring 中使用事务时，几乎不用关注实际的事务实现是什么。

Spring 事务的只读

“只读事务”并不是一个强制选项，它只是一个“暗示”，提示数据库驱动程序和数据库系统，这个事务并不包含更改数据的操作，那么 JDBC 驱动程序和数据库就有可能根据这种情况对该事务进行一些特定的优化，比方说不安排相应的数据库锁，以减轻事务对数据库的压力，毕竟事务也是要消耗数据库的资源的。“只读事务”仅仅是一个性能优化的推荐配置而已，并非强制你要这样做不可。

Spring 事务的事务超时

为了使应用程序更好的运行，事务不能运行太长的时间。因此，声明式事务的第四个特性就是超时。

Spring 事务的回滚规则

默认情况下，事务只有在遇到运行期异常时才会回滚，而在遇到检查型异常时不会回滚，但是也可以声明事务在遇到特定的检查型异常时像遇到运行期异常那样回滚。同样，你还可以声明事务遇到特定的异常不回滚，即使这些异常是运行期异常。

（4）结合项目中使用

如何管理的：

Spring 事务管理主要包括 3 个接口，Spring 的事务主要是由他们三个共同完成的。

1）PlatformTransactionManager：事务管理器--主要用于平台相关事务的管理主要有三个方法：

commit 事务提交；
rollback 事务回滚；
getTransaction 获取事务状态。

2）TransactionDefinition：事务定义信息--用来定义事务相关的属性，给事务管理器 PlatformTransactionManager 使用

这个接口有下面四个主要方法：

getIsolationLevel：获取隔离级别；

getPropagationBehavior：获取传播行为；

getTimeout：获取超时时间；

isReadOnly：是否只读（保存、更新、删除时属性变为 false--可读写，查询时为 true--只读）

事务管理器能够根据这个返回值进行优化，这些事务的配置信息，都可以通过配置文件进行配置。

3）TransactionStatus：事务具体运行状态--事务管理过程中，每个时间点事务的状态信息。

例如它的几个方法：

hasSavepoint()：返回这个事务内部是否包含一个保存点，

isCompleted()：返回该事务是否已完成，也就是说，是否已经提交或回滚

isNewTransaction()：判断当前事务是否是一个新事务

题目 44：Spring AOP 的实现原理？

（1）问题分析

Spring AOP 的面向切面编程，是面向对象编程的一种补充，用于处理系统中分布的各个模块的横切关注点，比如说事务管理、日志、缓存等。它是使用动态代理实现的，在内存中临时为方法生成一个 AOP 对象，这个对象包含目标对象的所有方法，在特定的切点做了增强处理，并回调原来的方法。

（2）核心问题讲解

Spring AOP 的动态代理主要有两种方式实现，JDK 动态代理和 cglib 动态代理。JDK 动态代理通过反射来接收被代理的类，但是被代理的类必须实现接口，核心是 InvocationHandler 和 Proxy 类。cglib 动态代理的类一般是没有实现接口的类，cglib 是一个代码生成的类库，可以在运行时动态生成某个类的子类，所以，CGLIB 是通过继承的方式做的动态代理，因此如果某个类被标记为 final，那么它是无法使用 CGLIB 做动态代理的。

（3）问题扩展

AOP 能做什么：

1、降低模块之间的耦合度
2、使系统容易扩展
3、避免修改业务代码，避免引入重复代码，更好的代码复用

AOP 怎么用：

前置通知：某方法调用之前发出通知。

后置通知：某方法完成之后发出通知

返回后通知：方法正常返回后，调用通知。在方法调用后，正常退出发出通知

异常通知：抛出异常后通知（After throwing advice） ： 在方法抛出异常退出时执行的通知。在方法调用时，异常退出发出通知

环绕通知：通知包裹在被通知的方法的周围知。

（4）结合项目中使用

可举例：项目中日志的处理和事务的处理

题目 45：IOC 和 DI 是什么？

（1）问题分析

主要考察 IOC 和 DI 的理解，使用的设计思想，IOC 和 DI 的关系

（2）核心问题讲解

IOC（控制反转）：全称为：Inverse of Control。从字面上理解就是控制反转了，将对在自身对象中的一个内置对象的控制反转，反转后不再由自己本身的对象进行控制这个内置对象的创建，而是由第三方系统去控制这个内置对象的创建。简单来说就是把本来在类内部控制的对象，反转到类外部进行创建后注入，不再由类本身进行控制，这就是 IOC 的本质。

DI（依赖注入）：全称为 Dependency Injection，意思自身对象中的内置对象是通过注入的方式进行创建。

IOC 和 DI 的关系：oc 就是容器，di 就是注入这一行为，那么 di 确实就是 ioc 的具体功能的实现。而 ioc 则是 di 发挥的平台和空间。所以说，ioc 和 di 即是相辅相成的搭档，又是殊途同归的双胞胎。最重要的是，他们都是良好的降低耦合的思想。

（3）问题扩展

DI 是如何实现的？

依赖注入可以通过 setter 方法注入（设值注入）、构造器注入和接口注入三种方式来实现，Spring 支持 setter 注入和构造器注入，通常使用构造器注入来注入必须的依赖关系，对于可选的依赖关系，则 setter 注入是更好的选择，setter 注入需要类提供无参构造器或者无参的静态工厂方法来创建对象。

（4）结合项目中使用

举个例子：一个类 A 需要用到接口 B 中的方法，那么就需要为类 A 和接口 B 建立关联或依赖关系，最原始的方法是在类 A 中创建一个接口 B 的实现类 C 的实例，但这种方法需要开发人员自行维护二者的依赖关系，也就是说当依赖关系发生变动的时候需要修改代码并重新构建整个系统。如果通过一个容器来管理这些对象以及对象的依赖关系，则只需要在类 A 中定义好用于关联接口 B 的方法（构造器或 setter 方法），将类 A 和接口 B 的实现类 C 放入容器中，通过对容器的配置来实现二者的关联。

题目 46：Spring 中用到了那些设计模式？

（1）问题分析

考察对设计模式的认识，以及 spring 中设计模式的应用

（2）核心问题讲解

1. 工厂模式，这个很明显，在各种 BeanFactory 以及 ApplicationContext 创建中都用到了；
2. 模版模式，这个也很明显，在各种 BeanFactory 以及 ApplicationContext 实现中也都用到了；
3. 代理模式，在 Aop 实现中用到了 JDK 的动态代理；
4. 单例模式，这个比如在创建 bean 的时候。
5. Tomcat 中有很多场景都使用到了外观模式，因为 Tomcat 中有很多不同的组件，每个组件需要相互通信，但又不能将自己内部数据过多地暴露给其他组件。用外观模式隔离数据是个很好的方法。
6. 策略模式在 Java 中的应用，这个太明显了，因为 Comparator 这个接口简直就是为策略模式而生的。Comparable 和 Comparator 的区别一文中，详细讲了 Comparator 的使用。比方说 Collections 里面有一个 sort 方法，因为集合里面的元素有可能是复合对象，复合对象并不像基本数据类型，可以根据大小排序，复合对象怎么排序呢？基于这个问题考虑，Java 要求如果定义的复合对象要有排序的功能，就自行实现 Comparable 接口或 Comparator 接口.
7. 原型模式：使用原型模式创建对象比直接 new 一个对象在性能上好得多，因为 Object 类的 clone()方法是一个 native 方法，它直接操作内存中的二进制流，特别是复制大对象时，性能的差别非常明显。
8. 迭代器模式：Iterable 接口和 Iterator 接口　这两个都是迭代相关的接口，可以这么认为，实现了 Iterable 接口，则表示某个对象是可被迭代的；Iterator 接口相当于是一个迭代器，实现了 Iterator 接口，等于具体定义了这个可被迭代的对象时如何进行迭代的

（3）问题扩展

讲一下项目中使用的设计模式：

项目中的事务处理，使用了代理模式，实际使用的类不是自己写的类而是生成的代理类，在调用相应的方法前开启事务处理；

（4）结合项目中使用

举个例子：

在我们的项目中遇到这样一个问题：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。我们以往在 spring 和 hibernate 框架中总是配置一个数据源，因而 sessionFactory 的 dataSource 属性总

是指向这个数据源并且恒定不变，所有 DAO 在使用 sessionFactory 的时候都是通过这个数据源访问数据库。但是现在，由于项目的需要，我们的 DAO 在访问 sessionFactory 的时候都不得不在多个数据源中不断切换，问题就出现了：如何让 sessionFactory 在执行数据持久化的时候，根据客户的需求能够动态切换不同的数据源？我们能不能在 spring 的框架下通过少量修改得到解决？是否有什么设计模式可以利用呢？

首先想到在 spring 的 applicationContext 中配置所有的 dataSource。这些 dataSource 可能是各种不同类型的，比如不同的数据库：Oracle、SQL Server、MySQL 等，也可能是不同的数据源：比如 apache 提供的 org.apache.com-mons.dbcp.BasicDataSource、spring 提供的 org.springframework.jndi.Jndi-ObjectFactoryBean 等。然后 sessionFactory 根据客户的每次请求，将 dataSource 属性设置成不同的数据源，以到达切换数据源的目的。

spring 中用到的包装器模式在类名上有两种表现：一种是类名中含有 Wrapper，另一种是类名中含有 Decorator。基本上都是动态地给一个对象添加一些额外的职责。

题目 47：Spring 中 Bean 的作用域有哪些？

（1）问题分析

Spring IOC 容器创建一个 Bean 实例时，可以为 Bean 指定实例的作用域，作用域包括 singleton（单例模式）、prototype（原型模式）、request（HTTP 请求）、session（会话）、global-session（全局会话）。

（2）核心问题讲解

作用域限定了 Spring Bean 的作用范围，在 Spring 配置文件定义 Bean 时，通过声明 scope 配置项，可以灵活定义 Bean 的作用范围。

scope 配置项有 5 个属性，用于描述不同的作用域。

① singleton：使用该属性定义 Bean 时，IOC 容器仅创建一个 Bean 实例，IOC 容器每次返回的是同一个 Bean 实例。

② prototype：使用该属性定义 Bean 时，IOC 容器可以创建多个 Bean 实例，每次返回的都是一个新的实例。

③ request：该属性仅对 HTTP 请求产生作用，使用该属性定义 Bean 时，每次 HTTP 请求都会创建一个新的 Bean，适用于 WebApplicationContext 环境。

④ session：该属性仅用于 HTTP Session，同一个 Session 共享一个 Bean 实例。不同 Session 使用不同的实例。

⑤ global-session：该属性仅用于 HTTP Session，同 session 作用域不同的是，所有的 Session 共享一个 Bean 实例。

（3）问题扩展

Bean 的生命周期：

1、实例化 bean 对象(通过构造方法或者工厂方法)

2、设置对象属性(setter 等)（依赖注入）

3、如果 Bean 实现了 BeanNameAware 接口，工厂调用 Bean 的 setBeanName()方法传递 Bean 的 ID。（和下面的一条均属于检查 Aware 接口）

4、如果 Bean 实现了 BeanFactoryAware 接口，工厂调用 setBeanFactory()方法传入工厂自身

5、将 Bean 实例传递给 Bean 的前置处理器的 postProcessBeforeInitialization(Object bean, String beanname)方法

6、调用 Bean 的初始化方法

7、将 Bean 实例传递给 Bean 的后置处理器的 postProcessAfterInitialization(Object bean, String beanname)方法

8、使用 Bean

9、容器关闭之前，调用 Bean 的销毁方法

（4）结合项目中使用

Bean 的作用域进本上使用的都是单例，在 struts2 中曾使用过多利，但目前已经很少使用了。

本题可以结合 springmvc 和 struts2 整合 spring 时的控制层的单例多例回答。

题目 48：spring 框架实现实例化和依赖注入的方式分别是什么？

（1）问题分析

主要考察对于 Spring 容器管理的 bean 的实例化方式，以及 spring 管理的 bean 如何在项目中注入使用

（2）核心问题讲解

Spring 框架实现实例化的三种方式：

第一种：**使用构造器实例化 Bean**：这是最简单的方式，Spring IoC 容器即能使用默认空构造器也能使用有参数构造器两种方式创建 Bean。

第二种：**使用静态工厂方式实例化 Bean**，使用这种方式除了指定必须的 class 属性，还要指定 factory-method 属性来指定实例化 Bean 的方法，而且使用静态工厂方法也允许指定方法参数，spring IoC 容器将调用此属性指定的方法来获取 Bean。

第三种：**使用实例工厂方法实例化 Bean**，使用这种方式不能指定 class 属性，此时必须使用 factory-bean 属性来指定工厂 Bean，factory-method 属性指定实例化 Bean 的方法，而且使用实例工厂方法允许指定方法参数，方式和使用构造器方式一样。

依赖注入是 Spring 协调不同 Bean 实例之间的合作而提供的一种工作机制，在确保 Bean 实例之间合作的同时，并能保持每个 Bean 的相对独立性。

1、 基于构造函数的依赖注入

构造函数注入就是通过 Bean 类的构造方法，将 Bean 所依赖的对象注入。构造函数的参数一般情况下就是依赖项，spring 容器会根据 bean 中指定的构造函数参数来决定调用那个构造

2、基于设置函数的依赖注入

将 Bean 所依赖的对象通过设置函数注入，Bean 需要为注入的依赖对象提供设置方法。

3、基于自动装配的依赖注入

Spring IOC 容器会基于反射查看 Bean 定义的类。当 Spring 容器发现 Bean 被设置为自动装配的 byType 模式后，它会根据参数类型在 Spring 容器中查找与参数类型相同的被依赖 Bean 对象,如果已经创建,则会把被依赖的对象自动注入到 Bean 中,如果没有创建,则不会注入。注入过程需要借助 Bean 提供的设置方法来完成，否则注入失败。

4、基于注解的依赖注入

Spring 主要提供了@Autowired 和@Resource 注解模式，下面也重点讨论这两种注解模式。

@Autowired 默认按类型装配（这个注解是属业 spring 的），默认情况下必须要求依赖对象必须存在，如果要允许 null 值，可以设置它的 required 属性为 false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier 注解进行使用

（3）问题扩展

@Resource 装配顺序：

1. 如果同时指定了 name 和 type，则从 Spring 上下文中找到唯一匹配的 bean 进行装配，找不到则抛出异常
2. 如果指定了 name，则从上下文中查找名称（id）匹配的 bean 进行装配，找不到则抛出异常
3. 如果指定了 type，则从上下文中找到类型匹配的唯一 bean 进行装配，找不到或者找到多个，都会抛出异常
4. 如果既没有指定 name，又没有指定 type，则自动按照 byName 方式进行装配；如果没有匹配，则回退为一个原始类型进行匹配，如果匹配则自动装配；

（4）结合项目中使用

题目 49：SpringMVC 的工作流程?

（1）问题分析

主要考察对 springmvc 中前端控制器和三大组件的理解，以及接收到请求后所进行的处理

（2）核心问题讲解

1. 用户向服务器发送请求，请求被 Spring 前端控制 Servelt DispatcherServlet 捕获；

2. DispatcherServlet 对请求 URL 进行解析，得到请求资源标识符（URI）。然后根据该 URI，调用 HandlerMapping 获得该 Handler 配置的所有相关的对象（包括 Handler 对象以及 Handler 对象对应的拦截器），最后以 HandlerExecutionChain 对象的形式返回；

3. DispatcherServlet 根据获得的 Handler，选择一个合适的 HandlerAdapter。（**附注**：如果成功获得 HandlerAdapter 后，此时将开始执行拦截器的 preHandler(...)方法）

4. 提取 Request 中的模型数据，填充 Handler 入参，开始执行 Handler（Controller)。 在填充 Handler 的入参过程中，根据你的配置，Spring 将帮你做一些额外的工作：

HttpMessageConveter： 将请求消息（如 Json、xml 等数据）转换成一个对象，将对象转换为指定的响应信息

数据转换：对请求消息进行数据转换。如 String 转换成 Integer、Double 等

数据根式化：对请求消息进行数据格式化。 如将字符串转换成格式化数字或格式化日期等

数据验证： 验证数据的有效性（长度、格式等），验证结果存储到 BindingResult 或 Error 中

5. Handler 执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象；

6. 根据返回的 ModelAndView，选择一个适合的 ViewResolver（必须是已经注册到 Spring 容器中的 ViewResolver)返回给 DispatcherServlet ；

7. ViewResolver 结合 Model 和 View，来渲染视图

8. 将渲染结果返回给客户端。

（3）问题扩展

Springmvc 的优点：

（1）它是基于组件技术的。全部的应用对象,无论控制器和视图,还是业务对象之类的都是 java 组件.并且和 Spring 提供的其他基础结构紧密集成.

（2）不依赖于 Servlet API(目标虽是如此,但是在实现的时候确实是依赖于 Servlet 的)

（3）可以任意使用各种视图技术,而不仅仅局限于 JSP

（4） 支持各种请求资源的映射策略

（5）它应是易于扩展的

springMVC 和 struts2 的区别有哪些?

（1）springmvc 的入口是一个 servlet 即前端控制器（DispatchServlet），而 struts2 入口是一个 filter 过虑器（StrutsPrepareAndExecuteFilter）。

（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。

（4）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。Jsp 视图解析器默认使用 jstl。

（4）结合项目中使用

1、解决 post 请求乱码问题：

在 web.xml 中加入：

```
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFil-
ter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

其余参考本体核心问题讲解

题目 50：spring 以及 springMVC 常用注解有哪些？

（1）问题分析：

考官主要对 spring 和 springMVC 注解的考核，如 Controller、Service、Repository、Autowired、RequestMapping、RequestBody、ResponseBody、Path-Variable、RequestParam、RestController。

（2）核心答案讲解：

@Controller 用于标记在一个类上，代表这个类是控制层组件。

@Service 用于标记在一个类上，代表这个类是业务层组件。

@Repository 用于标记在一个类上，代表这个类是数据访问层组件。

@RequestMapping 是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

@RequestParam 用于将指定的请求参数赋值给方法中的形参。

@PathVariable 可以获取 URL 中的动态参数。

@RequestBody 用于读取 Request 请求的 body 部分数据，通过适当的消息转换器转换为对象。

@ResponseBody 用于将 Controller 的方法返回的对象，通过适当的消息转换器转换。

@RestController=@Controller+@ResponseBody，用于标记在一个类上。

（3）问题扩展

在使用@RequestBody 和@ResponseBody 注解时，springMVC 默认使用 MappingJackson2HttpMessageConverter 转换器将 json 格式的字符串和对象相互转换。

（4）结合项目中使用

对于前后端分离的项目，我们往往会给前端直接返回 json 格式的字符串由前端接收处理。

题目 51：简单介绍下 springMVC 和 struts2 的区别有哪些？【了解】

（1）问题分析：

考官主要是对 springMVC 和 struts2 的区别的考核。

（2）核心答案讲解：

springMVC 的入口是一个 servlet 即前端控制器；struts2 入口是一个 filter 过虑器。

springMVC 是基于方法开发，传递参数是通过方法形参，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。

springMVC 通过参数绑定将 request 请求内容解析,并给方法形参赋值；struts2 采用值栈存储请求和相应的数据,通过 OGNL 存取数据。

（3）问题扩展：

springmvc 是基于方法的开发，都是用形参接收值，一个方法结束参数就销毁了，多线程访问都会有一块内存空间产生，里面的参数也是不会共用的，所有 springmvc 默认使用了单例，这时候 controller 里面不适合在类里面定义属性。如果在特殊情况需要定义属性的时候，那么就在类上面加上注解@Scope("prototype")改为多例的模式。

（4）结合项目中使用：

企业在进行技术选型的时候更倾向于使用 springMVC 框架,因为 struts2 存在很多漏洞。

题目 52：springmvc 前端控制器是什么？处理器映射器是什么？

（1）问题分析：

考官主要是对 springMVC 组件的考核。前端控制器是 DispatcherServlet，处理器映射器是 HandlerMapping。

（2）核心答案讲解：

DispatcherServlet：前端控制器。用户请求到达前端控制器，它就相当于 mvc 模式中的 c，DispatcherServlet 是整个流程控制的中心，由它调用其它组件处理用户的请求，DispatcherServlet 的存在降低了组件之间的耦合性。

HandlerMapping：处理器映射器。HandlerMapping 负责根据用户请求找到 Handler 即处理器，springmvc 提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

（3）问题扩展：

Handler：处理器。Handler 是继 DispatcherServlet 前端控制器的后端控制器，在 DispatcherServlet 的控制下 Handler 对具体的用户请求进行处理。由于 Handler 涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求开发 Handler。

HandlerAdapter：处理器适配器。通过 HandlerAdapter 对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

ViewResolver：视图解析器。ViewResolver 负责将处理结果生成 View 视图，ViewResolver 首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成 View 视图对象，最后对 View 进行渲染将处理结果通过页面展示给用户。

View：视图。SpringMVC 框架提供了很多的 View 视图类型的支持，包括：jstlView、freemarkerView、pdfView 等。我们最常用的视图就是 jsp。一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

（4）结合项目中使用：

在 web.xml 中配置前端控制器。

题目 53：springmvc 如何进行参数绑定

（1） 问题分析：

考官主要针对 springMVC 进行参数绑定的考核。

（2）核心答案讲解：

处理器适配器调用 springmvc 提供的参数绑定组件将前端传入的 key/value 数据转成 controller 方法的形参。

（3）问题扩展：

参数绑定组件：springmvc 提供了很多 converter（转换器），在特殊情况下需要自定义 converter，比如对日期数据的绑定。

（4）结合项目中使用：

SpringMVC 配置任何类型转换器 Converter（以时间类型为例）。

1. 定义时间 DateConverter 转换类实现 Converter<String, Date> 接口。

```
public class DateConverter implements Converter<String, Date> {
    @Override
    public Date convert(String dateStr) {
        Date date = null;
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
        try {
             date = dateFormat.parse(dateStr);
        } catch (ParseException e) {
            //e.printStackTrace();
            dateFormat = new SimpleDateFormat("yyyy-MM-dd");
             try {
                date = dateFormat.parse(dateStr);
            } catch (ParseException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
        return date;
    }
}
```

2.配置 spring-mvc.xml

```
<!-- 配置配置类型转换器注解驱动 -->
<mvc:annotation-driven conversion-service="conversionService" />
<!-- 配置类型转换器 -->
<bean id="conversionService" class="org.springframework.format.support.For-
mattingConversionServiceFactoryBean">
    <property name="converters">
        <set>
            <bean class="com.itcast.controller.DateConverter" />
        </set>
    </property>
```

</bean>

3. 页面把时间字符串按格式转成时间类型传往后台。

题目 54：springmvc 获取表单数据的几种方式？

（1）问题解析：

考官主要针对 Spring MVC 中 Controller 如何获取 Form 表单提交的数据的考核，比如通过 HttpServletRequest 接收等。

（2）核心答案解析

Controller 的方法，添加 HttpServletRequst 类型入参，通过 HttpServletRequst.getParameter()获取请求数据。

Controller 的方法，添加对应表单字段 name 的参数，有几个表单字段就添加多少个对应的入参。

Controller 的方法，添加自定义 Java 类型的入参，并添加@ModelAttribute 注解，由这个入参对象接收表单提交的数据。

（3）问题扩展：

当表单字段名与 Controller 的方法的形参对象名不一致时，可以在参数类型前添加@requestParam(“表单字段名”)注解来获取表单参数。

（4）结合项目中使用：

添加对应表单字段 name 的参数，有几个表单字段就添加多少个对应的入参：

```
@RequestMapping(value="/user/save", method=RequestMethod.POST)
private String doSave(@RequestParam("userName") String userName, @RequestParam("age") Integer age){}。
```

题目 55：SSM 架构的整合流程是怎样的？

（1）问题分析：

考官主要针对 SSM 架构的搭建进行考核。

（2）核心答案讲解：

创建 maven web 项目。

准备所需的 jar 包：Spring 框架 jar 包、MyBatis 框架 jar 包、MyBatis 整合 Spring 中间件 jar 包、数据库驱动 jar 包、日志包等。

准备数据库资料：建立所需的数据库表。

完成配置文件：database.properties、applicationContext.xml、springmvc-config.xml、mybatis-config.xml、web.xml、log4j.properties。

（3）问题扩展:

MyBatis-Plus（简称 MP）是一个 MyBatis 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。

（4）结合项目中使用：

企业在进行技术选型时，使用 SSM 的项目较多，在搭建过程中，注意不要丢掉该有的 jar 包、不要丢掉该有的配置。

题目 56：mybatis 和 hibernate 之间的优缺点比较？【了解】

（1）问题分析：

考官主要对 mybatis 和 hibernate 的优缺点进行考核。

（2）核心答案讲解：

Hibernate 的优点：

hibernate 是全自动，hibernate 完全可以通过对象关系模型实现对数据库的操作，拥有完整的 JavaBean 对象与数据库的映射结构来自动生成 sql。

功能强大，数据库无关性好，O/R 映射能力强，需要写的代码很少，开发速度很快。

数据库移植性良好。

Hibernate 的缺点：

学习门槛高，精通门槛更高，程序员如何设计 O/R 映射，在性能和对象模型之间如何取得平衡，以及怎样用好 Hibernate 方面需要的经验和能力都很强才行。

hibernate 的 SQL 很多都是自动生成的，无法直接维护 SQL；虽然有 hql 查询，但功能还是不及 SQL 强大，hql 查询是有局限的；hibernate 虽然也支持原生 sql 查询，但开发模式上却与 orm 不同，需要转换思维，因此使用上有些不方便。

Mybatis 的优点：

易于上手和掌握，提供了数据库查询的自动对象绑定功能，而且延续了很好的 SQL 使用经验，对于没有那么高的对象模型要求的项目来说，相当完美。

SQL 写在 xml 里，便于统一管理和优化，解除 SQL 与程序代码的耦合。

提供映射标签，支持对象与数据库的 orm 字段关系映射。

提供 xml 标签，支持编写动态 SQL。

速度相对于 Hibernate 的速度较快。

Mybatis 的缺点：

关联表多时，字段多的时候，SQL 工作量很大。

SQL 依赖于数据库，导致数据库移植性差。

（3）问题扩展：

spring data jpa 整合了 hibernate，使用起来更加方便。

（4）结合项目中使用：

在实际项目中，可以同时使用多个数据访问层框架。对于单表的增删改查建议使用 hibernate 框架使开发更加高效，对于需要复杂的 SQL 比如多表联查建议使用 mybatis 框架会更加灵活。

题目 57：MyBatis 中使用#和$书写占位符有什么区别？

（1）问题分析：

考官主要考察 mybatis 中变量传参的区别

（2）核心答案讲解：

使用#传入参数时，sql 语句解析是会加上"",当成字符串来解析，这样相比于$的好处是比较明显的，#{}传参能防止 sql 注入，如果你传入的参数为 单引号'，那么如果使用${},这种方式 是会报错的

另外一种场景时，如果要做动态的排序，比如 order by column，这个时候务必要用${}。

（3）问题扩展

无

（4）结合项目中使用

项目中经常用到的是#，因为这样能有效防止 sql 注入

题目 58：MyBatis 中的动态 SQL 是什么意思？

（1）问题分析：

看到动态 SQL 就要想到静态 SQL,知道两者的区别和优劣,知道什么是动态 SQL 和静态 SQL,两者结合起来回答比较好

（2）核心答案讲解：

1.所谓 SQL 的动态和静态，是指 SQL 语句在何时被编译和执行，二者都是用在 SQL 嵌入式编程中的，这里所说的嵌入式是指将 SQL 语句嵌入在高级语言 中，而不是针对于单片机的那种嵌入式编程。

2.SQL 语句的主体结构，在编译时尚无法确定，只有等到程序运行起来，在执行的过程中才能确定，这种 SQL 叫做动态 SQL

3.静态 SQL 语句的编译是在应用程序运行前进行的，编译的结果会存储在数据库内部。而程序运行时，数据库将直接执行编译好的 SQL 语句，降低运行时的开销

（3）问题扩展

1.MyBatis 中用于实现动态 SQL 的元素主要有：if 、choose、 when、 otherwise、 trim、where、set 、foreach。

2.另外还要注意一点，在 SQL 中如果某些参数没有确定，如“select * from user where age > ? ”这种语句是静态 SQL，不是动态 SQL，虽然个别参数的值不知道，但整个 SQL 的结构已经确定，数据库是可以将它编译的，在执行阶段只需将个别参数的值补充进来即可

（4）结合项目中使用

使用 LIKE 语句进行模糊查询进行条件匹配

题目 59：Mybatis 中 Mapper 动态代理规范是什么？

（1）问题分析：

面试官主要考核编写 mapper 映射文件的注意点

（2）核心答案讲解：

1、xml 映射文件中的 namespace 与 mapper 接口的全类名相同。

2、Mapper 接口方法名和 xml 映射文件中定义的每个 statement 的 id 相同。
3、Mapper 接口方法的输入参数类型和 xml 映射文件中定义的每个 sql 的 parameterType 的类型相同。
4、Mapper 接口方法的输出参数类型和 xml 映射文件中定义的每个 sql 的 resultType 的类型相同。

（3）问题扩展：

Mybatis 中的 mapper 动态代理是不支持方法重载的 Dao 接口里的方法，因为是全类名+方法名的保存和寻找策略。
Mapper 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK 动态代理为 mapper 接口生成代理 proxy 对象，代理对象 proxy 会拦截接口方法，转而执行 MappedStatement 所代表的 sql，然后将 sql 执行结果返回。

（4）结合项目中使用：

在项目中使用 Dao 接口开发时，编写对应的 mapper 映射文件时候需要注意 mapper 映　　射文件的规范.

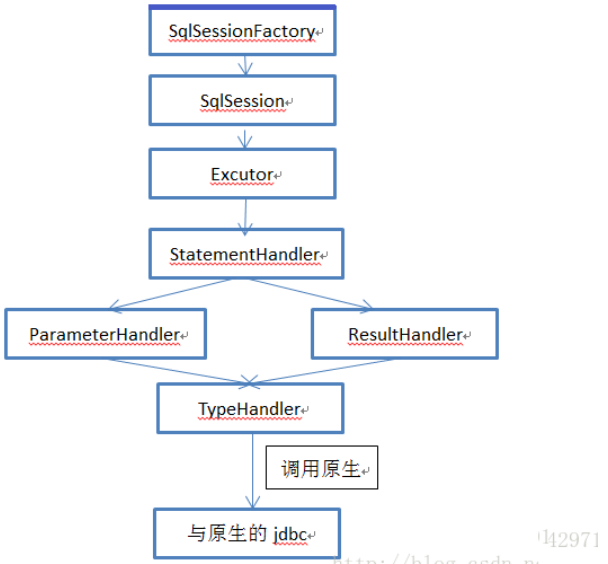
题目 60：mybatis 的执行流程是什么？

（1）　问题分析：

考官主要想考察学生对 mybatis 的理解以及熟悉程度

（2）　核心答案讲解：

mybatis 的执行流程如下：
1、new SqlSessionFactoryBuilder().build(InputStream inputStream);这个是构造 SqlSessionFactory 的方法，返回的是 DefaultSqlSessionFactory，在这个过程中已经创建了 Configuration 对象是通过 XMLConfigBuilder 的 parse()方法创建的。
2、SqlSessinoFactory.openSession() 返回的 sqlSession 是 DefaultSession 类型的，此 SqlSession 里包含一个 Configuration 的对象，和一个 Executor 对象
3、sqlSession.getMapper(Class<T> type);这个是创建 mapper 的方法，内部是通过 MapperProxyFactory(Mapper 代理工厂)来创建一个 mapper 接口的代理对象。
4、当执行 mapper 中相应的方法时，是通过反射执行的 mapper 代理对象的 invoke 方法，同时完成 JavaBean 对象到数据库参数之间的相互转换，这种映射关系就是有 TypeHandler 对象来完成的，在获取数据表对应的元数据时，会保存该表所有列的数据库类型。
（PS：下图为 mybatis 的执行流程，面试时可以给面试官画出来:）



（3） 问题扩展

mybatis 底层还是采用原生 jdbc 来对数据库进行操作的，只是通过 SqlSessionFactory，SqlSession Executor，StatementHandler，ParameterHandler，ResultHandler 和 TypeHandler 等几个处理器封装了这些过程

（4） 结合项目中使用

在秒项目开发中，mybatis 都是作为数据持久层的框架使用，并且一般都结合 spring 来使用。我们开发者只需要关注 SQL 本身，而不需要花费精力去处理例如注册驱动、创建 connection、创建 statement、手动设置参数、结果集检索等 jdbc 繁杂的过程代码。Mybatis 通过 xml 或注解的方式将要执行的各种 statement（statement、preparedStatemnt、CallableStatement）配置起来，并通过 java 对象和 statement 中的 sql 进行映射生成最终执行的 sql 语句，最后由 mybatis 框架执行 sql 并将结果映射成 java 对象并返回。

题目 61：说说你比较熟悉的设计模式及应用场景？

（1） 问题分析：

考官主要是对一些常用设计模式和代码抽取优化能力的考核，问题可能涉及 6 大设计原则及 23 种设计模式。

（2） 核心答案讲解：

比较熟悉的有：

单例模式：保证一个类仅有一个实例，并提供一个全局访问点，比如一些配置文件或者管理类可以设计为单例，我们常用的线程池也是单例的

模板方法：在定义好的算法骨架下，允许子类为一个或多个步骤提供实现，一次性实现算法的不变部分，将可变部分留给子类实现，当子类实现代码逻辑雷

同时，可以使用此设计模式
工厂方法：
创建对象需要大量的重复代码时，通过子类实现方法来创建对象。如 Spring 中通过工厂模式将创建对象的任务交给容器管理。
建造者模式：
讲复杂对象的构建和表示分离，适用于流程固定，但是顺序不一定固定的场景。如需要给一个对象多次给不同的属性赋值，可以使用链式调用穿参，最后生成对象
另外的如，foreach 中的迭代器模式，spring 中 ASM 的访问者模式，动态代理等都有一些了解。

（3） 问题扩展
ASM 可以操作 java 字节码的框架，可以读取修改 java 中的字节码。
Spring 中获取 FileSystemResource 和 ClassPathResource 等功能雷同的类时使用模板方法。
单例包含懒汉饿汉式以及不同的变种，工厂类设计模式一般也设计为单例

（4） 结合项目中使用
单例：项目中的一些配置，或者引入外部的 sdk 需要创建管理类，或者封装自己的框架时，需要用到单例
工厂方法：在编码时不能预见需要创建哪种类的实例

题目 62：动态代理 的 2 种方式以及区别？

（1）问题分析：
考官主要想对 JDK 动态代理、CGlib 动态代理这两种代理进行考核，如两种代理的定义、特点应用等。

（2）核心答案讲解：
JDK 动态代理：利用反射机制生成一个实现代理接口的匿名类，在调用具体方法前调用 InvokeHandler 来处理。
CGlib 动态代理：利用 ASM（开源的 Java 字节码编辑库，操作字节码）开源包，将代理对象类的 class 文件加载进来，通过修改其字节码生成子类来处理。

（3）问题扩展：
1.JDK 代理使用的是反射机制实现 aop 的动态代理，CGLIB 代理使用字节码处理框架 asm，通过修改字节码生成子类。所以 jdk 动态代理的方式创建代理对象效率较高，执行效率较低，cglib 创建效率较低，执行效率高；2.JDK 动态代理机制是委托机制，具体说动态实现接口类，在动态生成的实现类里面委托 hanlder 去调用原始实现类方法，CGLIB 则使用的继承机制，具体说被代理类和代理类是继承关系，所以代理类是可以赋值给被代理类的，如果被代理类有接口，那么代理类也可以赋值给接口。

（4）结合项目中使用：
JDK 动态代理回调方式是反射，目标类是接口类，例如事务
CGlib 动态代理通过 FastClass 方法索引调用，目标是非接口类，非 final 类，非 final 方法
Spring AOP 的核心实现原理就是采用的动态代理，根据被代理对象是否实现了所要被代理的接口这个条件，动态代理会选择不同的实现方案.

项目阶段

题目 63：项目中有没有用到多线程，哪些地方要用多线程？

（1） 问题分析：

考官主要想考核面试人员对项目的开发经验，或者说考核学员到底有没有真实的开发经验。

（2） 核心答案讲解：

1. 同步商品名牌
2. 同步商品信息
4. 用户静态任务
5. 地址缓存的同步
6. 熔断中有多线程
7. 拼团 锁单、解锁订单、取消订单 失败或者异常重试方法

新启动一个线程，每隔一秒重试一次，重试 5 次，如果 5 次都失败的话发送 MQ 处理

8. Mq 处理
9. 商品库存
10. 消息的处理 店铺服务

（3） 问题扩展：

是否使用多线程是看实际场景，跟架构没多大关系。比如你用 ssm 做电商网站 ，肯定要考虑多线程问题，如果用 ssm 做一般的管理应用系统，并发量不是很大，就不需要多线程。 并不是说采用 ssh 的项目，框架自动就帮你弄好多线程了，那是不可能的。

（4） 结合项目中使用：

使用多线程就是为了充分利用 cpu 的资源，提高程序执行效率，当你发现一个业务逻辑执行效率特别低，耗时特别长，就可以考虑使用多线程。

题目 64：日志文件的管理，你们是怎么做的？

（1） 问题分析：

考官主要是想对你是否有真实的工作经验进行考核，以及对 aop 的理 解。

（2） 核心答案讲解

在 Java 开发中日志的管理有很多种。我一般会使用过滤器，或者是 spring 的拦截器进行日志的处理。如果是用过滤器比较简单，只要对所有的.do 提交进行拦截，然后获取 action 的提交路径就可以获取对每个方法的调用。然后进行日志记录。使用过滤器的好处是可以自己选择性的对某一些方法进行过滤，记录日志。但是实现起来有点麻烦。

另外一种就是使用 Spring 的 AOP 了。这种方式实现起来非常简单，只要配置一下配置文件就可以了。可是这种方式会拦截下所有的对 action 的每个操作。使得效率比较低。不过想做详细日志这个方法还 是非常好的

(3)问题扩展

Spring AOP 对普通类的拦截操作：

首先我们要写一个普通类，此类作为日志记录类；

```
[java]
1. <span style="font-size:12px;">          package chen.hui.log
2.         public classs MyLog{
3.             //在类里面写方法，方法名诗可以任意的。此处我用标准的before和after
   来表示
4.             public void before(){
5.                 System.out.println("被拦截方法调用之前调用此方法，输出此语
   句");
6.             }
7.             public void after(){
8.                 System.out.println("被拦截方法调用之后调用此方法，输出此语
   句");
9.             }
0.         }
1.         其次我们在写一个类作为被拦截类（Spring的AOP就是拦截这个类里面的方
   法）
2.         package chen.hui.log
3.         public class Test{//此类中方法可以写任意多个。我只写一个
4.             public void test(){
5.                 Sytem.out.println("测试类的test方法被调用");
6.             }
```

最后进行配置文件的编写。在 Spring 的配置文件中我们需要进行几句话的配置

```
[html]
<span style="font-
size:12px;">        <bean id="testLog" class="chen.hui.log.MyLog"></bean> <!
--将日志类注入到bean中。-->

            <aop:config>
                    <aop:aspect id="b" ref="testLog"><!--调用日志类-->
                    <aop:pointcut id="log" expression="execution(* chen.hui.log.*.*
(..))"/><!--配置在log包下所有的类在调用之前都会被拦截-->
                    <aop:before pointcut-ref="log" method="before"/><!--在log包
下面所有的类的所有方法被调用之前都调用MyLog中的before方法-->
                    <aop:after pointcut-ref="log" method="after"/>><!--在log包下
面所有的类的所有方法被调用之前都调用MyLog中的after方法-->

                    </aop:aspect>
```

到此处整个程序完成，在 MyLog 类里面的 before 和 after 方法添加日志逻辑代码就可以完成日志的管理。以上是对普通类的管理，如果只想拦截某一个类。只要把倒数第二个 * 改成类名就可以了。

第二：使用 Spring AOP 对 action 做日志管理

如果是想拦截 action 对 action 做日志管理，基本和上面差不多，但是要注意。以下几点

首先还是要写一个普通类，不过此类中的方法需要传入参数。

```
[java]
01. <span style="font-size:12px;">          package chen.hui.log
02.         import org.aspectj.lang.JoinPoint;
03.          public classs MyLog{
04.                  //在类里面写方法，方法名诗可以任意的。此处我用标准的before和after
来表示
05.                   //此处的JoinPoint类可以获取，action所有的相关配置信息和request等
内置对象。
06.                 public void before(JoinPoint joinpoint){
07.                         joinpoint.getArgs();//此方法返回的是一个数组，数组中包括
request以及ActionCofig等类对象
08.                         System.out.println("被拦截方法调用之前调用此方法，输出此语
句");
09.                 }
10.                 public void after(JoinPoint joinpoint){
11.                         System.out.println("被拦截方法调用之后调用此方法，输出此语
句");
12.                 }
```

其次我们在写一个 action 类作为被拦截类（Spring 的 AOP 就是拦截这个类里面的方法）

最后进行配置文件的编写。在 Spring 的配置文件中我们需要进行几句话的配置

```
[html]
01. <span style="font-
size:12px;">          <bean id="testLog" class="chen.hui.log.MyLog"></bean> <!
--将日志类注入到bean中。-->
02.
03.          <aop:config>
04.              <aop:aspect id="b" ref="testLog"><!--调用日志类-->
05.              <aop:pointcut id="log" expression="execution(* chen.hui.log.*.*
(..))"/><!--配置在log包下所有的类在调用之前都会被拦截-->
06.              <aop:before pointcut-ref="log" method="before"/><!--在log包
下面所有的类的所有方法被调用之前都调用MyLog中的before方法-->
07.              <aop:after pointcut-ref="log" method="after"/>><!--在log包下
面所有的类的所有方法被调用之前都调用MyLog中的after方法-->
08.
09.              </aop:aspect>
10.
```

除了参数外其他地方基本和普通类相似。

需要注意的是：普通类可以监控单一的类，而 action 在配置文件中只能到包名而不能到 action 的类名。不然会报错。就是说如果要记录日志就要记录所有的 action 而不能记录其中一个

（4）结合项目使用

同（3）一

题目 65：你觉得分布式开发的缺点是什么？

（1） 问题分析

这个问题首先你得知道分布式开发是什么，再说明缺点。

（2） 核心问题讲解：

举个例子：

随着淘宝的做大，功能也日益完善，加了很多的功能，在把一个项目都让一套 tomcat 跑，tomcat 说它也很累，能不能少跑点代码，这时候分布式系统架构就产生了，我们把天猫这个大项目按功能划分为很多的模块，比如说单独一个系统处理订单，一个处理用户登录，一个处理后台等等，然后每一子系统都单独跑在一个 tomcat 中，和起来就是一个完整的天猫项目，这样对每一个 tomcat 就相对轻松一点。（如果某个子系统的压力还是很多，可以考虑对这个子系统再做集群）

总结：多台服务器合起来跑的才是一套完整代码，这就叫分布式

(3) 问题扩展

	传统单体架构	分布式服务化架构
新功能开发	需要时间	容易开发和实现
部署	不经常且容易部署	经常发布，部署复杂
隔离性	故障影响范围大	故障影响范围小
架构设计	难度小	难度级数增加
系统性能	响应时间快，吞吐量小	响应时间慢，吞吐量大
系统运维	运维简单	运维复杂
新人上手	学习曲线大（应用逻辑）	学习曲线大（架构逻辑）
技术	技术单一且封闭	技术多样且开放
测试和查错	简单	复杂
系统扩展性	扩展性很差	扩展性很好
系统管理	重点在于开发成本	重点在于服务治理和调度

1. 从上面的表格可以看到，分布式系统虽然有一些优势，但也存在一些问题
2. 架构设计变得复杂（尤其是其中的分布式事务）
3. 部署单个服务会比较快，但是如果一次部署需要多个服务，部署会变得复杂
4. 系统的吞吐量会变大，但是响应时间会变长
5. 运维复杂度会因为服务变多而变得很复杂
6. 架构复杂导致学习曲线变大
7. 测试和查错的复杂度增大
8. 技术可以很多样，这会带来维护和运维的复杂度
9. 管理分布式系统中的服务和调度变得困难和复杂

（4）结合项目使用

我们在品优购中使用的 dubbo 就是分布式框架，还是面向 SOA 架构的。

题目 66：支付接口是怎么做的？

（1） 问题分析

我们一般都是调用阿里的支付宝或者微信的接口来做

（2） 核心答案讲解：支付接口业务流程步骤：

1、生成订单信息

2、商户到银行，商户发起请求到银行

把订单信息的部分数据拼接，然后通过银行端给的 MD5 加密工具加密，然后作为参数，通过 http 的 post 请求 发送到支付提供的目标银行网站。例子：

```
var bankURL = "b2bpay.ccb.com/NCCB/NECV5B2BPayMainPlat"; // 生产环境的银行请求地址
```

```
        tmp = 'MERCHANTID=' + MERCHANTID + '&POSID=' + POSID + '&BRANCHID='
                + BRANCHID + '&ORDERID=' + ORDERID + '&PAYMENT=' + PAYMENT
                + '&CURCODE=' + CURCODE + '&TXCODE=' + TXCODE + '&REMARK1='
                + REMARK1 + '&REMARK2=' + REMARK2 + '&PROJECTNO =' + PRO-
JECTNO
                + '&PAYACCNO=' + PAYACCNO + '&ACCTYPE=' + ACCTYPE + '&END-
TIME ='
                + ENDTIME + '& TYPE =' + TYPE
        '& PUB =' + PUB + '& REGINFO =' + REGINFO + '& PROINFO =' + PROINFO
                + '& REFERER =' + REFERER; // 加密原串
        strMD5 = hex_md5(tmp); // 调用加密函数生成 MAC 值对应的加密串
        subfor.action = bankURL + '?' + tmp + '&MAC=' + strMD5;
        var params = tmp + '&MAC=' + strMD5;
        subform.submit();
```

3、界面跳转到支付页面，客户在页面支付成功，银行进行业务处理

4、银行到商户，银行调用商户接口返回处理结果，商户接收后验证数字签名，更改单据状态

(2) 问题拓展：

1）银行通过 MD5withRSA 算法对返回结果进行加密生成数字签名

2）商户要预先提供接口给银行回调，这个接口一般是在商户到银行步骤里面，我们通过参数传递给银行

3）商户接口处理过程：

(1)获取银行传过来的数据；

(2)使用公钥进行签名的逆运算

(3)使用标准 MD5 算法运算原文

(4)比较(2) 、(3) 结果,如果一样，验证签名成功，更改单据状态，否则，返回支付失败等提示信息

```
    /*
          * 此例子是 post 方式
          */
         /*银行提供的公钥*/
         String strPubKey =
"30819d300d06092a864886f70d010101050003818b0030818702818100 9d2ac18031a59
66ae6bf4ea0c317144d4944beda90ef828298a4b30a4c31a57a8ff921e8b05c6b6b9ae5e
f7a984359b6ff46ad5c31fdc8ef24d541defa0d65ecd5ae-
dea19f803742d3526399d9c7cfb795a8ed-
cabb0b9eff58f384074f163f9f646cfc3c6b0730c900ec1acac7b6dc24f949697dbd0f9a
0658b5640c37378787020111";
         /*从上一表单提取出来要素的值*/
         String
strSrc="10000037200905090001_sadCQ450533779#0E50001004141059866666 重庆
市工程建设招标投标交易中心.012N20120904184827 上海市
http://localhost:8080/WebContent/my_pay.htm";
         RSASig rsa = new RSASig();
         /*验签串*/
```

```
            String signString =
"25ee808325b6e950c29d1fd50e05fc44f4dc9cad7aa0ef430977a4d73f1604196a2adee
b3130f471e2b480e03fedc34a33e874b58cb0c36337ee4090f58a923b03c8f5f4d936b35
e46b53c20eedd0116371f8daa1136d7afd12ec2f7cf23af140398ce91d75822297304da6
e2db65597a696788b1457fcea2b027ebe6e61eeef";
            rsa.generateSigature(strSrc);
            rsa.setPublicKey(strPubKey);
            /*验签结果判断*/
            if (rsa.verifySigature(signString, strSrc)) {
                System.out.println("验签成功！！");
            } else {
                System.out.println("验签失败！！");
        }
```

4. 我们在项目中的使用

我们在品优购里面支付模块中调用的就是微信支付接口

题目 67：redis 为什么可以做缓存？

（1.）问题分析

这个题目考得是你对 redis 的理解，他能做缓存的原因是什么

（2.）核心问题讲解

1.Redis 将其数据库完全保存在内存中，仅使用磁盘进行持久化。与其它键值数据存储相比，Redis 有一组相对丰富的数据类型。Redis 可以将数据复制到任意数量的从机中。

2.异常快 - Redis 非常快，每秒可执行大约 110000 次的设置(SET)操作，每秒大约可执行 81000 次的读取/获取(GET)操作。支持丰富的数据类型 - Redis 支持开发人员常用的大多数数据类型 ，例如列表，集合，排序集和散列等等。这使得 Redis 很容易被用来解决各种问题，因为我们知道哪些问题可以更好使用地哪些数据类型来处理解决。

3.操作具有原子性 - 所有 Redis 操作都是原子操作，这确保如果两个客户端并发访问，Redis 服务器能接收更新的值。

4.Redis 是一个内存数据库，但在磁盘数据库上是持久的，因此它代表了一个不同的权衡，在这种情况下，在不能大于存储器(内存)的数据集的限制下实现非常高的写和读速度

5.redis 支持多种数据结构，

1）Redis 不仅仅支持简单的 k/v 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。

2）Redis 支持 master-slave(主-从)模式应用

3）Redis 支持数据持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

4）Redis 单个 value 的最大限制是 1GB，memcached 只能保存 1MB 的数据。

（3）问题拓展

Redis 缓存机制：

redis 提供了三种缓存机制，LFU，LRU，TTL

LFU，least frequently used，即最小使用频率淘汰，每个对象使用共计 24bit 空间用来实施这个算法，24bit 分成 2 个部分，前 16bit 用来记录上次减少时间（减少的是后

面 8bit 计数器），后面 8 个 bit 是一个对数计数器，用来记录这个对象的访问次数。需要注意的是，这个字段不是一直增加的，也需要减少，否则会出现一个情况，一个对象很久之前被频繁的使用，但是最近没有被使用，若这个值不减少的话，那么这个对象会一直处在一个不会被淘汰的位置。前面 16bit 的作用就是，记录一个“减量时间”，这是一个降低精度的 Unix 时间，将 16bit 的时间转换成分钟，不关心回绕问题。若这个时间很大，那么 8bit 计数器的值减半，否则只是简单的每次递减 1

LRU，last recently used，即最近最少使用淘汰，一般做法是，将 hash 表的 value 做成一个指针，指向一个双链表节点，节点中保存实际的 value，双链表按照上次访问时间降序排列，当访问到一个对象之后，更新访问时间，并将这个节点移动到表头，若节点不存在就直接插入到表头。当内存门限达到的时候，从链表尾开始删除若干 entry。redis 为了减少内存使用，不使用双链表或其他结构管理对象，采用随机算法，每次从 hash 表中随机选择一些 key，一般是 5 个，将这些 key 存入一个全局的池，池大小一般是 16，池中 entry 按照上次访问时间降序排列，每次从池中选择尾部的 entry，就是最差的对象，将这个对象淘汰
TTL，即生存时间，按照生存时间设置一个对象的生命周期，一个对象生命周期结束之后将其销毁

（4）在项目中的使用
我们在做搜索以及购物车里面都是用过 redis，结合 spring 来使用，springdateredis。

题目 68：solr 怎么设置搜索结果排名靠前（得分）？

(1) 问题分析
Solr 有个 boost 值，boost 值越高，计算出来的相关度得分就越高，排名也就越靠前

(2) 核心问题详解
利用 solr 自己的排序方法，可以在查询时指定按照哪一字段进行排序，比如按照时间的倒叙等，配置多个字段权重可以通过 Solr 的 edismax 实现的方法，配置不同字段的权重最终影响 boost 的打分。这种方法比较简单，需要配置一下即可用，但有自身的局限性，对于特殊需求达不到满足，比如付费的信息最前显示。

在 edismax 方法的基础上进行修改，edismax 支持 boost 函数与 score 相乘作为打分结果，可以在建立索引时单独建立一个字段作为排序的依据字段，并且配合其他字段设置权重来共同影响最终的打分结果

（3）问题拓展
Solr 可以结合 IK 分词器来使用
那么 solr 中 lk 分词器原理是什么？
IK 分词器的分词原理是词典分词。先在内存中初始化一个词典，然后在分词过程中挨个读取字符，和字典中的字符相匹配，把文档中的所有词语拆分出来的过程。

（4）项目中的使用
我们在品优购里面搜索模块使用的就是 solr,结合 IK 分词器来使用，一般可以做推荐，我们可以通过设置 boost 值来进行热门产品的推荐

题目 69：activeMQ 在项目中如何应用的？

（1） 问题分析

Activemq 是一个消息中间件，这个主要是靠你对这个中间件的使用

（2） 核心问题讲解

Activemq 在项目中主要是完成系统之间通信，并且将系统之间的调用进行解耦。例如在添加、修改商品信息后，需要将商品信息同步到索引库、同步缓存中的数据以及生成静态页面一系列操作。在此场景下就可以使用 activemq。一旦后台对商品信息进行修改后，就向 activemq 发送一条消息，然后通过 activemq 将消息发送给消息的消费端，消费端接收到消息可以进行相应的业务处理。

（3）问题拓展

Activemq 较其他中间件对比，会产生消息丢失情况

消息丢失:

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

（4）在项目中的使用

我们在品优购里面可以用来做短信通知以及在搜索里面做保证数据实时更新。

题目 70：activeMQ 如果数据提交不成功怎么办？

（1） 问题分析

此题考得是 activemq 的消息传递机制以及传递失败我们应该怎么处理

（2）核心问题讲解

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

（3）问题拓展

在不开启事物的情况下 采用的是应答模式 4（ActiveMQSession.AUTO_ACKNOWLEDGE）消费一次 应答一次

这时候消费失败了，由于没有配置死亡队列,消息就不会被消费堆积在队列中，那么怎么才可以让消息再被消费呢？

由于项目中的应用场景，有个方案启动和停止的功能，项目启动启动监听，项目停止，停止监听

```
public class MqService {

    private JmsTemplate jmsTemplate;

    private CachingConnectionFactory cachingProductConnectionFactory;

    private CachingConnectionFactory cachingConsumersConnectionFac-
tory;

    private static final String ACTIVEMQQUEUE_OPTIMIZATION = "";
// 性能优化参数

// consumer.prefetchSize 预加载消息

// 20 条

// ?consumer.prefetchSize=20

    /**

     * 发送消息

     *
```

```
     * @param ryzhMessage
     */
    public void send(final RyzhMessage ryzhMessage) {

        // 得到 MQ 工具类
        RyzhMqHolder hodler = mqHolders.get(ryzhMessage.getShce-
meId());
        // 发送信息
        jmsTemplate.setConnectionFactory(cachingProductConnectionFac-
tory);
        jmsTemplate.send(hodler.getDestination(), new MessageCreator()
{
            public Message createMessage(Session session) throws
JMSException {
                ObjectMessage objectMessage = session.createObjectMes-
sage();
                objectMessage.setObject(ryzhMessage);
                return objectMessage;
            }
        });
    }

    /**
     * 监听
     *
```

```
     * @param schemeId
     * @param receiver
     */
    public void startListen(RyzhScheme scheme) {
        RyzhMqHolder holder = mqHolders.get(scheme.getSchemeId());
        try {
            if (holder == null) {
                RyzhConsumer consumer = (RyzhConsumer) SpringBeanU-
tils.getBean("ryzhConsumer");
                consumer.setScheme(scheme);
                logger.info("RyzhMqService-启动消息队列监听器
schemeId=" + scheme.getSchemeId());
                // 如果是为空
                holder = new RyzhMqHolder();
                // 得到整合方案的配置
                TRyzhFa tRyzhFa = ((RyzhConsumer) consumer).get-
Scheme().getZhfa();
                // 得到消息队列配置的信息
                String xxdlJson = tRyzhFa.getXxdlpz();
                // json 转 map
                RyzhMqConfig ryzhMqConfig = JSON.parseObject(xxdlJson,
RyzhMqConfig.class);
                // 设置方案 ID
                holder.setSchemeId(scheme.getSchemeId());
```

```
                // 设置目的地

                ActiveMQQueue destination = new Ac-
tiveMQQueue(RyzhMqConfig.QUEUE_NAME + scheme.getSchemeId() + AC-
TIVEMQQUEUE_OPTIMIZATION);

                holder.setDestination(destination);

                // 创建监听器

                DefaultMessageListener listener = new DefaultMessage-
Listener();

                // 给监听器设置消费者

                listener.setReceiver(consumer);

                // 将监听器保存在 holder 中

                holder.setListener(listener);

                // 创建监听容器

                DefaultMessageListenerContainer listenerContainer = new
DefaultMessageListenerContainer();

                // 监听容器属性的配置

                listenerContainer.setConnectionFactory(cachingConsum-
ersConnectionFactory);

                // 设置目的地

                listenerContainer.setDestination(destination);

                // 设置监听器

                listenerContainer.setMessageListener(listener);

                // 设置消费者集群数

                listenerContainer.setConcurrentConsumers(ryzhMqCon-
fig.getConcurrentConsumers());
```

```
                // 设置监听队列还是主题 默认是队列
                listenerContainer.setPubSubDomain(RyzhMqConfig.PUB-
SUB_DOMAIN);
                listenerContainer.setPubSubNoLocal(false);
                // listenerContainer.setAcceptMessagesWhileStop-
ping(true);
                // 设置应答模式 默认是 4
                listenerContainer.setSessionAcknowledgeMode(RyzhMqCon-
fig.SESSION_ACKNOWLEDGEMODE);
                // 设置是否启动事物 默认不开启
                listenerContainer.setSessionTransacted(RyzhMqCon-
fig.SESSION_TRANSACTED);
                // 将监听容器保存在 holder 中
                holder.setListenerContainer(listenerContainer);
                // 将 holder 缓存在 map 中
                mqHolders.put(scheme.getSchemeId(), holder);
                // 初始化容器
                holder.getListenerContainer().initialize();
                // 启动监听
                holder.getListenerContainer().start();
                logger.info("RyzhMqService-消息队列监听器启动成功
schemeId=" + scheme.getSchemeId());
            }
        } catch (Exception e) {
```

```
            logger.error("RyzhMqService-消息队列监听器启动失败
schemeId=" + scheme.getSchemeId(), e);

        }

    }


    public void stopListen(RyzhScheme scheme) {

        RyzhMqHolder ryzhMqHolder = mqHolders.get(scheme.get-
SchemeId());

        // 停止监听

        ryzhMqHolder.getListenerContainer().destroy();

        // 移除缓存

        mqHolders.remove(scheme.getSchemeId());

    }


    /**

     * 取得 MQHolder

     *

     * @param ryzhMessage

     * @return

     */


    public Map<String, RyzhMqHolder> getMqHolders() {

        return mqHolders;
```

```
    }

    public void setMqHolders(Map<String, RyzhMqHolder> mqHolders) {
        this.mqHolders = mqHolders;
    }

    public JmsTemplate getJmsTemplate() {
        return jmsTemplate;
    }

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    public CachingConnectionFactory getCachingProductConnectionFac-
tory() {
        return cachingProductConnectionFactory;
    }

    public void setCachingProductConnectionFactory(CachingConnection-
Factory cachingProductConnectionFactory) {
        this.cachingProductConnectionFactory = cachingProductConnec-
tionFactory;
    }
```

```
    public CachingConnectionFactory getCachingConsumersConnectionFac-
tory() {

        return cachingConsumersConnectionFactory;

    }



    public void setCachingConsumersConnectionFactory(CachingConnec-
tionFactory cachingConsumersConnectionFactory) {

        this.cachingConsumersConnectionFactory = cachingConsumersCon-
nectionFactory;

    }



}
```

具体的 xml 配置如下

```
    <?xml version="1.0" encoding="UTF-8"?>

    <beans xmlns="http://www.springframework.org/schema/beans"

        xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:context="http://www.springframework.org/schema/con-
    text"

        xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"

        xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"

        xmlns:lang="http://www.springframework.org/schema/lang"
```

```
    xsi:schemaLocation="

        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd

        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
3.2.xsd

        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.2.xsd

        http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd

        http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd

        http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd

        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-con-
text.xsd"

    default-autowire="byName">



    <bean id="jmsTemplate" class="org.springframe-
work.jms.core.JmsTemplate">

        <property name="connectionFactory" ref="cach-
ingProductConnectionFactory"></property>

        <!-- 持久化 -->

        <property name="deliveryMode" value="2"></property>

        <!-- 不开启事物 -->

        <property name="sessionTransacted"
value="false"></property>
```

```
        <!-- 应答模式是 INDIVIDUAL_ACKNOWLEDGE -->

        <property name="sessionAcknowledgeMode"
value="4"></property>

    </bean>



    <bean id="ryzhMqService" class="com.dragon-
soft.rygl.mq.RyzhMqService"

        scope="singleton">

        <property name="cachingConsumersConnectionFactory"
ref="cachingConsumersConnectionFactory"></property>

        <property name="cachingProductConnectionFactory"
ref="cachingProductConnectionFactory"></property>

        <property name="jmsTemplate" ref="jmsTemplate" />

    </bean>



    <bean id="defaultRyzhProduceProcessor"

        class="com.dragonsoft.rygl.ryzh.service.produce.De-
faultRyzhProduceProcessor"

        scope="prototype">

        <property name="mgService" ref="ryzhMqService"></prop-
erty>

        <property name="jdbcTemplate" ref="jdbcTem-
plate"></property>

        <property name="ip" value="${jmx.ip}"></property>

        <property name="port" value="${jmx.port}"></property>
```

```
        <property name="dataSourceManager" ref="data-
SourceManager"></property>

    </bean>

    <!-- 生产者专用缓存池 -->

    <bean id="cachingProductConnectionFactory"

        class="org.springframework.jms.connection.CachingCon-
nectionFactory">

        <property name="targetConnectionFactory" ref="connec-
tionFactory"></property>

        <property name="reconnectOnException"
value="true"></property>

        <property name="sessionCacheSize" value="${jms.ses-
sionCacheSize}"></property>

    </bean>

    <!-- 消费者专用缓存池 -->

    <bean id="cachingConsumersConnectionFactory"

        class="org.springframework.jms.connection.CachingCon-
nectionFactory">

        <property name="targetConnectionFactory" ref="connec-
tionFactory"></property>

        <property name="reconnectOnException"
value="true"></property>

        <property name="sessionCacheSize" value="${jms.ses-
sionCacheSize}"></property>

        <span style="color:#ff0000;"><property name="cacheCon-
sumers" value="false"></property>

        <property name="cacheProducers" value="false"></prop-
erty></span>
```

```
    </bean>

    <bean id="connectionFactory" class="org.apache.activemq.spring.ActiveMQConnectionFactory">

        <property name="brokerURL" value="${jms.brokerURL}" />

        <property name="userName" value="${jms.userName}"></property>

        <property name="password" value="${jms.password}"></property>

    <property name="useAsyncSend" value="${jms.useAsyncSend}"></property>
```

以上就可以达到不用配置死信队列就可以重新消费信息。（其实原理就是重启了监听容器）

（4）项目中的使用

同（3）

题目 71：单点登录系统是怎么做的？

（1）问题分析：

考官主要想考察在分布式系统中不同子系统间如何实现一次登录多处访问的具体实现和流程。

（2）核心答案讲解：

单点登录（Single Sign On），简称为 SSO，是目前比较流行的企业业务整合的解决方案之一。SSO 的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。而完成单点登录典型核心技术实现有 CAS。CAS 主要完成认证工作。

CAS 结构体系包含两部分：

1、 CAS Server，CAS Server 主要负责完成对用户的认证工作，CAS Server 是一台应用需要独立部署。 CAS Server 可能会到数据库或 XML 文件中检索用户密码进行认证工作。

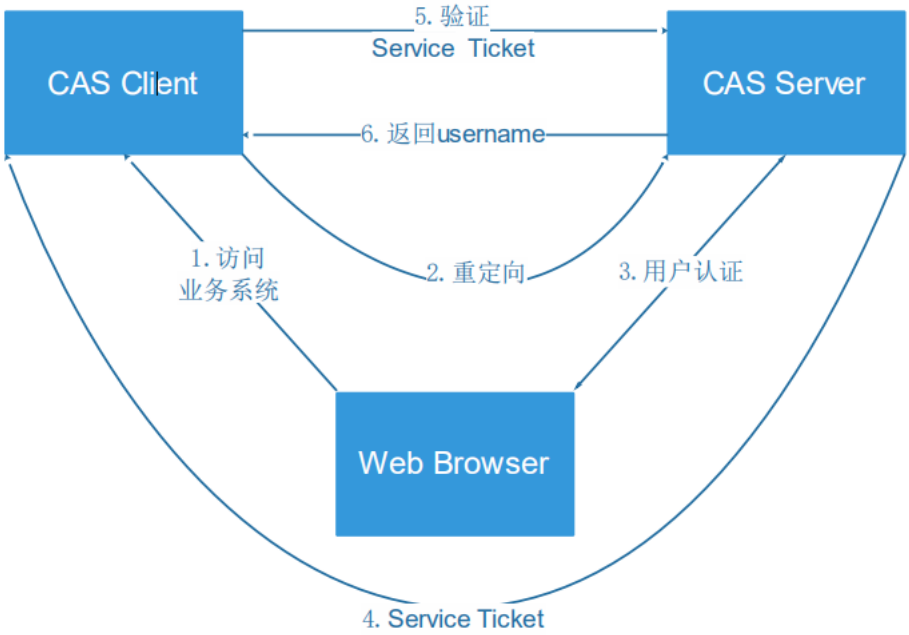
2、 CAS Client，CAS Client 一般表示业务系统（即我们开发的项目）但是在项

目中添加了 CAS 的各类过滤器；对受保护的资源进行身份认证，它将认证重定向到 CAS Server 中进行。

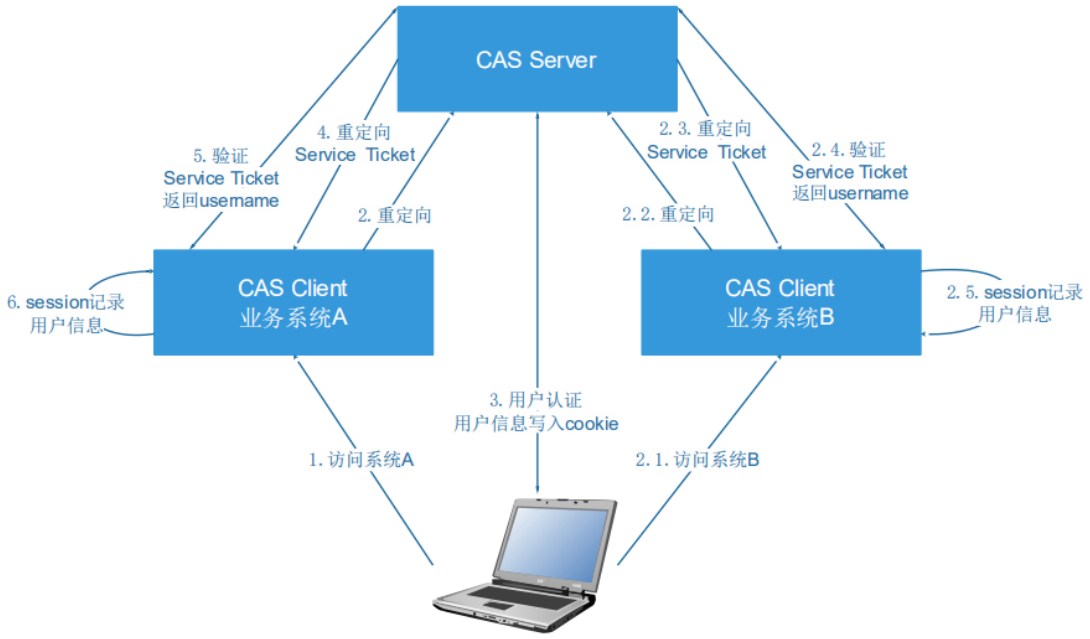
具体实现过程如下：

在用户浏览器中要访问系统 A 时，将在系统 A 里面重定向到 CAS Server 进行登录验证；如果没有登录那么跳转到 CAS Server 的登录页面，用户在该登录页面需要输入帐号和密码；然后数据正确的话，那么在 CAS Server 中将用户信息进行加密保存到 Cookie（TGC(Ticket Granted Cookie)）。 这时再由 CAS Server 生成一个 ticket 一次性凭证，随带这个 ticket 重定向到系统 A；再由系统 A 获取到的 ticket 再重定向回 CAS Server，验证 ticket 后系统 A 将用 户认证信息存入自身系统的 session 里。以后访问会先到系统 A 的 session 查找用 户信息，如果能找到则说明已经登录认证；如果 session 中读取不到用户信息那 么会到 CAS Server 根据浏览器中读取到的 cookie（TGC(Ticket Granted Cookie)）获取用户信息并返回，如果还没有那么则和前面认证流程一样需要登录等。访问系 统 B 的时候因为其 session 中无用户信息将会到 CAS Server 中根据 cookie 读取用 户信息，从而实现不用再次登录即可访问系统 B 资源。在退出的时候，CAS Server 会删除 TGC 对应的 TGT，并通知各个已登录过的服务器退出登录（删除 session 信息）。

流程图如下：

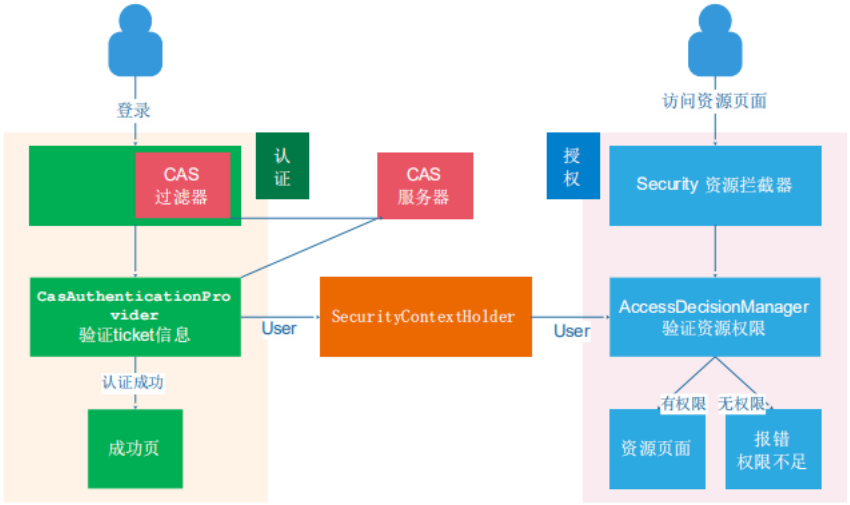


拓展图如下：



（3）问题扩展

CAS 主要是完成认证的工作，在完成认证之后还需要根据登录用户的权限进行授权完成对不同权限访问不同资源的工作，即授权，授权工作可以通过 CAS 整合 Springsecurity 进行授权，在认证通过后将用户信息存储在 SecurityContextHolder 中，再通过 Springsecurity 的 AccessDecisionManager 进行授权。具体流程如下：



（4）结合项目中使用

在没登陆情况下访问分布式商城的个人中心后，在访问购物车系统。

题目 72：dubbo 服务开发流程，运行流程？

（1）问题分析：

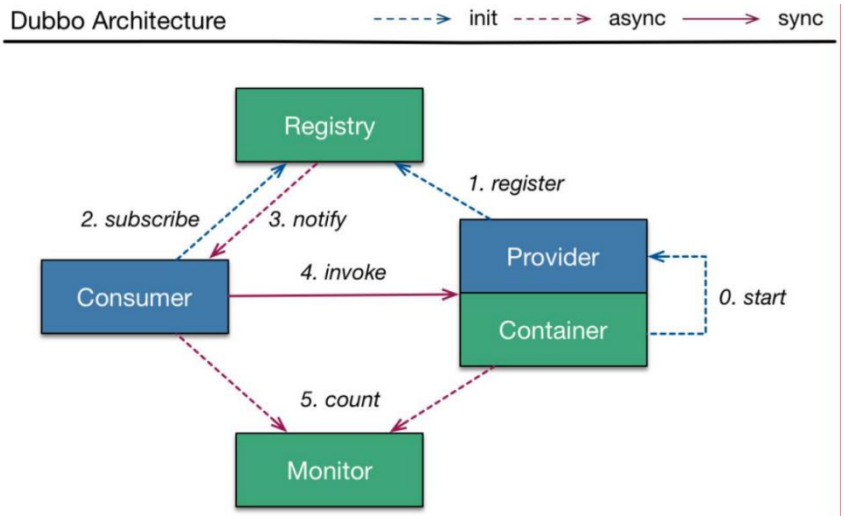
考官主要想考核 dubbo 的原理，还有 dubbo 在项目中的使用。

（2）核心答案讲解：

dubbo 服务开发流程:

1. maven 工程中 pom 文件先导入 dubbo 依赖 jar 包
2. 搭建 zookeeper 注册中心
3. 写好服务端工程并配置 dubbo 服务端配置，并关联上 zookeeper 注册中心
4. 写好客户端工程并配置 dubbo 客户端配置，并关联上 zookeeper 注册中心
5. 客户端工程依赖注入服务端实习类，即可调用相关暴露的接口

dubbo 运行流程：



1. 服务容器负责启动，加载，运行服务提供者。
2. 服务提供者在启动时，向注册中心注册自己提供的服务。
3. 服务消费者在启动时，向注册中心订阅自己所需的服务。
4. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
5. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
6. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

（3）问题扩展

新版本 dubbo 的工作原理，REST 风格的使用。

（4）结合项目中使用

在分布式架构项目中，可以使用 dubbo 做接口暴露跟扩域调用。

题目 73：solr 跟数据库的区别，你清楚么？

（1）问题分析：这个问题有待商榷，solr 是搜索引擎，而数据库是一个文件系统，本身就不是一个类型的东西，何谈区别？考官应该是想考察 solr 索引库、以及数据库的区别。那首先就要搞清楚，什么是索引库？为什么要建立索引库？索引库有什么好处？数

据存在数据库与索引库有什么区别？

（2）核心答案讲解：
数据库，简而言之可视为电子化的文件柜—存储电子文件的处所，用户可以对文件中的数据进行新增、截取、修改、删除等操作。索引库，可以理解为索引的集合。那么为什么要有索引？索引是为了提高查询数据库的效率，就好比一本书，如果没有目录，岂不是很麻烦？索引库就是索引的集合。所以，弄明白什么是数据库，什么是索引库之后，他们俩我的区别也就迎刃而解了。索引库的存在是基于数据库的，为了提高查询效率，我们要建立索引，索引太多，我们要统一管理，所以有了索引库。

（3）问题扩展：
索引的原理：对要查询的字段建立索引其实就是把该字段按照一定的方式排序；建立的索引只对该字段有用，如果查询的字段改变，那么这个索引也就无效了，比如图书馆的书是按照书名的第一个字母排序的，那么你想要找作者叫张三的就不能用改索引了；还有就是如果索引太多会降低查询的速度
索引的优缺点：　首先明白为什么索引会增加速度，DB 在执行一条 Sql 语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。那么在任何时候都应该加索引么？这里有几个反例：1、如果每次都需要取到所有表记录，无论如何都必须进行全表扫描了，那么是否加索引也没有意义了。2、对非唯一的字段，例如“性别”这种大量重复值的字段，增加索引也没有什么意义。3、对于记录比较少的表，增加索引不会带来速度的优化反而浪费了存储空间，因为索引是需要存储空间的，而且有个致命缺点是对于 update/insert/delete 的每次执行，字段的索引都必须重新计算更新。所以并不是任何情况下都改建立索引的

（4）结合项目中使用：
在实际项目中，当数据量特别大，数据库查询速度较慢的时候，我们可以建立索引，提高查询效率。在电商项目中，当我们在前台页面搜索商品名称关键词时，我们这时是在 Solr 库中去查找相应的商品信息，然后将搜索关键词高亮.那么 Solr 库中的商品信息又是如何添加的呢？ 当我们在给商品上架的时候，将商品信息 update 到 mysql 数据库中的 bbs_product 表中，然后同样的将相应的信息 添加到 Solr 库中.

题目 74：广告数据是怎么用 redis 缓存的？

（1） 问题分析：
考官主要考核 redis 的使用。
（2） 核心答案讲解：
使用 SpringDataRedis 实现广告数据的缓存，使用的 hash 类型操作，先获取广告列表对象，再将这个对象使用 redisTemplate.boundHashOps("xxx").put("x", "xx")方法存入缓存，当广告数据发生变更时，需要将缓存数据清除，使用 redisTemplate.boundHashOps("xxx").delete("x", "xx")方法清除缓存，这样再次查询才能获取最新的数据，考虑到用户可能会修改广告的分类，需要把原分类的缓存和新分类的缓存都清除掉。
（3） 问题扩展：

我们目前的系统已经实现了广告后台管理和广告前台展示，但是对于首页每天有大量的人访问，对数据库造成很大的访问压力，甚至是瘫痪。那如何解决呢？我们通常的做法有两种：一种是数据缓存、一种是网页静态化。

（4） 结合项目中使用：

现在我们首页的广告每次都是从数据库读取，这样当网站访问量达到高峰时段，对数据库压力很大，并且影响执行效率。我们需要将这部分广告数据缓存起来。

题目 75：项目中权限是怎么做的？

（1）问题分析：

首先分清楚系统中的岗位与角色关系

1.1、在项目中，我们可以将岗位看做角色，根据岗位来进行权限的分配。但是在大公司这种方案就不适用了，岗位太多，所以我们将岗位提取出来，给具有共性的岗位赋予一个角色，然后给角色分配权限。

1.2、权限角色是系统功能权限设置的基础，相当于用户分组，所有用户对应到相应权限的角色，便具有该权限角色所赋予的所有功能权限。岗位是在组织架构下的惊喜岗位划分概念，是因为岗位非常多，而很多不同的机构、部门下的同一职务拥有同样的功能权限，如果直接用岗位来设置将极大增加重复工作量。权限角色实际上有些相当于岗位权限分类的概念，即具有同样功能权限的岗位集合在一起，这样可以减少权限设置的工作量。

权限分配的几种模板

2.1、我们给用户分配权限，不是直接给用户分配权限，因为用户太多了，这样分配效率太低。通常我们使用的是基于角色的权限模型，即用户，角色，权限。

（2）核心答案讲解：

利用第三方开源框架 Shiro 或 Spring security 来完成项目权限的控制。一般我们都是用 Shiro 框架来完成项目权限，因为 Spring security 涉及太过复杂，难上手。

（3）问题扩展

权限其实就是利用 Java web 中的拦截器来拦截指定的字符串，假如我们不利用第三方框架来完成权限，那么我们可以利用拦截器来做。

（4）结合项目中使用

shiro 能做什么？

认证：验证用户的身份

授权：对用户执行访问控制：判断用户是否被允许做某事

会话管理：在任何环境下使用 Session API，即使没有 Web 或 EJB 容器。

加密：以更简洁易用的方式使用加密功能，保护或隐藏数据防止被偷窥

Realms：聚集一个或多个用户安全数据的数据源

单点登录（SSO）功能。

为没有关联到登录的用户启用 "Remember Me“ 服务

Shiro 的四大核心部分

Authentication(身份验证)：简称为“登录”，即证明用户是谁。

Authorization(授权)：访问控制的过程，即决定是否有权限去访问受保护的资源。

Session Management(会话管理)：管理用户特定的会话，即使在非 Web 或 EJB 应用程序。

Cryptography(加密)：通过使用加密算法保持数据安全

shiro 的三个核心组件：

Subject ： 正与系统进行交互的人，或某一个第三方服务。所有 Subject 实例都被绑定到（且这是必须的）一个 SecurityManager 上。

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，并通过它来提供安全管理的各种服务。当 Shiro 与一个 Subject 进行交互时，实质上是幕后的 SecurityManager 处理所有繁重的 Subject 安全操作。

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。Shiro 提供了多种可用的 Realms 来获取安全相关的数据。如关系数据库(JDBC)，INI 及属性文件等。可以定义自己 Realm 实现来代表自定义的数据源。

题目 76：如何处理 activeMQ 消息丢失的问题?

（1）问题分析：

mq 有个基本原则，就是数据不能多一条，也不能少一条，不能多，就是说重复消费和幂等性问题。不能少，就是说这数据别搞丢了。那这个问题你必须得考虑一下。这个丢数据，mq 一般分为两种，要么是 mq 自己弄丢了，要么是我们消费的时候弄丢了。

（2）核心答案讲解：

（1）生产者弄丢了数据

生产者将数据发送到 mq 的时候，可能数据就在半路给搞丢了，因为网络啥的问题，都有可能。

此时可以选择用 mq 提供的事务功能，就是生产者发送数据之前开启 mq 事务（channel.txSelect），然后发送消息，如果消息没有成功被 mq 接收到，那么生产者会收到异常报错，此时就可以回滚事务（channel.txRollback），然后重试发送消息；如果收到了消息，那么可以提交事务（channel.txCommit）。但是问题是，mq 事务机制一搞，基本上吞吐量会下来，因为太耗性能。

所以一般来说，如果你要确保说写 mq 的消息别丢，可以开启 confirm 模式，在生产者那里设置开启 confirm 模式之后，你每次写的消息都会分配一个唯一的 id，然后如果写入了 mq 中，mq 会给你回传一个 ack 消息，告诉你说这个消息 ok 了。如果 mq 没能处理这个消息，会回调你一个 nack 接口，告诉你这个消息接收失败，你可以重试。而且你可以结合这个机制自己在内存里维护每个消息 id 的状态，如果超过一定时间还没接收到这个消息的回调，那么你可以重发。

事务机制和 cnofirm 机制最大的不同在于，事务机制是同步的，你提交一个事务之后会阻塞在那儿，但是 confirm 机制是异步的，你发送个消息之后就可以发送下一个消息，然后那个消息 mq 接收了之后会异步回调你一个接口通知你这个消息接收到了。

所以一般在生产者这块避免数据丢失，都是用 confirm 机制的。

（2）mq 弄丢了数据

就是 mq 自己弄丢了数据，这个你必须开启 mq 的持久化，就是消息写入之后会持久化到磁盘，哪怕是 mq 自己挂了，恢复之后会自动读取之前存储的数据，一般

数据不会丢。除非极其罕见的是，mq 还没持久化，自己就挂了，可能导致少量数据会丢失的，但是这个概率较小。

设置持久化有两个步骤，第一个是创建 queue 的时候将其设置为持久化的，这样就可以保证 mq 持久化 queue 的元数据，但是不会持久化 queue 里的数据；第二个是发送消息的时候将消息的 deliveryMode 设置为 2，就是将消息设置为持久化的，此时 mq 就会将消息持久化到磁盘上去。必须要同时设置这两个持久化才行，mq 哪怕是挂了，再次重启，也会从磁盘上重启恢复 queue，恢复这个 queue 里的数据。

而且持久化可以跟生产者那边的 confirm 机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者 ack 了，所以哪怕是在持久化到磁盘之前，mq 挂了，数据丢了，生产者收不到 ack，你也是可以自己重发的。

哪怕是你给 mq 开启了持久化机制，也有一种可能，就是这个消息写到了 mq 中，但是还没来得及持久化到磁盘上，结果不巧，此时 mq 挂了，就会导致内存里的一点点数据会丢失。

3）消费端弄丢了数据

mq 如果丢失了数据，主要是因为你消费的时候，刚消费到，还没处理，结果进程挂了，比如重启了，那么就尴尬了，mq 认为你都消费了，这数据就丢了。

这个时候得用 mq 提供的 ack 机制，简单来说，就是你关闭 mq 自动 ack，可以通过一个 api 来调用就行，然后每次你自己代码里确保处理完的时候，再程序里 ack 一把。这样的话，如果你还没处理完，不就没有 ack？那 mq 就认为你还没处理完，这个时候 mq 会把这个消费分配给别的 consumer 去处理，消息是不会丢的。

（3）问题扩展：

1、你知道不知道你们系统里为什么要用消息队列这个东西？

2、kafka、activemq、rabbitmq、rocketmq 都有什么优点和缺点啊？

3、如何保证消息队列的高可用啊？

4、如何保证消息不被重复消费啊（如何保证消息消费时的幂等性）？

5、如何保证消息的顺序性？

6、如何解决消息队列的延时以及过期失效问题？消息队列满了以后该怎么处理？有几百万消息持续积压几小时，说说怎么解决？

7、如果让你写一个消息队列，该如何进行架构设计啊？说一下你的思路

（4）结合项目中使用：

1. 异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种 1.串行的方式；2.并行方式。

2.应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。

3.流量削锋

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛。

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用容易挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

可以控制活动的人数.

可以缓解短时间内高流量压垮应用；

（1）用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面；

（2）秒杀业务根据消息队列中的请求信息，再做后续处理。

4.消息通讯

消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。

点对点通讯：客户端 A 和客户端 B 使用同一队列，进行消息通讯。

聊天室通讯：客户端 A，客户端 B，客户端 N 订阅同一主题，进行消息发布和接收。实现类似聊天室效果。

以上实际是消息队列的两种消息模式，点对点或发布订阅模式。

题目 77：token 校验的过程?

（1）问题分析：

考官想了解 token 的校验过程。

（2）核心答案讲解：

访问服务： sso 客户端发送请求访问应用系统提供的服务资源

定向认证：sso 客户端会重定向用户请求到 SSO 服务器

用户认证：用户身份认证

发放票据：sso 服务器会产生一个随机的 service ticket

验证票据：sso 服务器验证票据 service ticket 的合法性，验证通过后，允许客户端访问服务

传输用户信息：sso 服务器验证票据通过后，传输用户认证结果信息给客户端

校验有效性。

（3）问题扩展：

无

（4）结合项目中使用：

使用 CAS 进行用户校验。

题目 78：solr 和数据库怎么交互的？

（1）问题分析：

考官主要考核对 solr 的熟悉程度和 solr 创建索引方面的知识

（2）核心答案讲解：

Solr 和数据库的交互是将数据库的数据进行分词后生成索引保存在 solr 的服务中;

而生成索引的方式有三种:

A:通过 solr 服务的可视化界面的 dataimport 插件批量导入索引数据,需要配置 data-config.xml 文件中的字段和数据库的连接信息等来实现.

B:通过 solrJ 来实现,solrJ 是 solr 服务的 java 客户端,提供索引和搜索的请求方法.

C:通过使用 Spring data Solr 框架来实现生成索引,其底层使用的还是 solrJ;通过配置 solrTemplate 来对索引库进行增删改查的操作,项目中就是使用这种方式的.

(3) 问题扩展:

修改商品信息后需要将新的商品信息同步到索引库,从而保证 solr 的索引数据和数据库的数据一致;

(4) 结合项目中使用:

在商城中，使用 solr 服务来完成商品的搜索,从而提高搜索效率。具体参考品优购的第九天和第十天.

题目 79：redis 缓存与数据库同步，怎么做的？

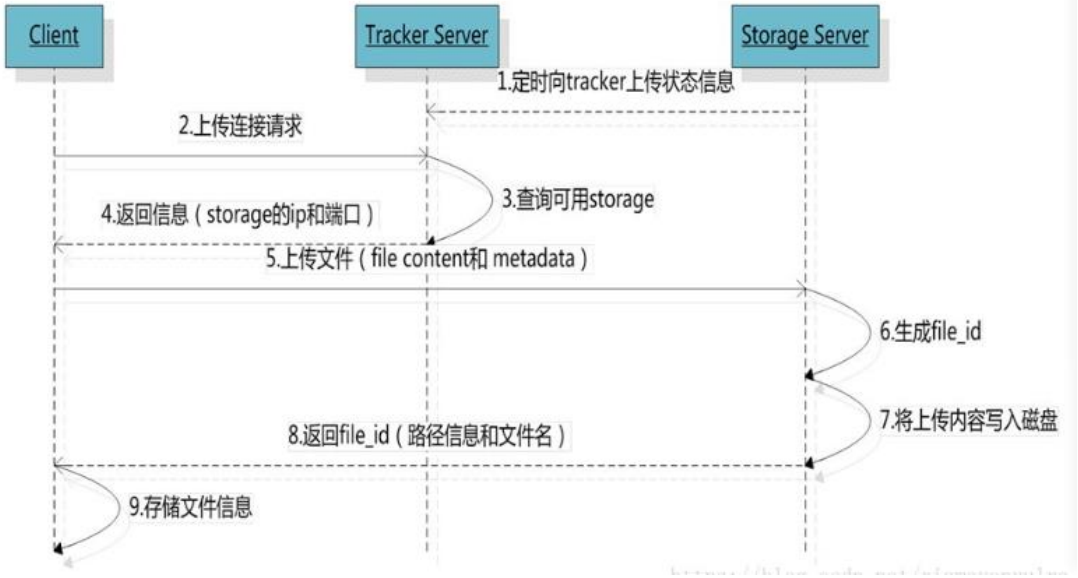
(1) 问题分析：
考官主要考察面试者对于项目中缓存使用的能力
(2) 核心答案讲解：
答 1:
方案 1：我们会先去 redis 中判断数据是否存在，如果存在，则直接返回缓存好的数据。而如果不存在的话，就会去数据库中，读取数据，并把数据缓存到 Redis 中。适用场合：如果数据量比较大，但不是经常更新的情况(比如用户排行)

方案 2：先去 redis 中判断数据是否存在，如果存在，则直接更新对应的数据(这一步会把对应更新过的 key 记录下来，比如也保存到 redis 中比如：key 为：save_update_keys【用-push 列表记录】)，并把更新后的数据返回给页面。而如果不存在的话，就会去先更新数据库中内容，然后把数据保存一份到 Redis 中。

答 2:
只要使用了缓存就涉及到缓存同步的问题。缓存同步其实就是当缓存的信息发生变化，也就是对后台对缓存的数据进行增、删、改操作后，数据库中的数据发生了变化同时要把缓存中的数据对应删除即可。当页面再次请求数据时，缓存中不能命中就会从数据库中查询并且添加到缓存中，即实现了缓存同步。
(3) 问题扩展:
无
(4) 结合项目中使用:
在商城中，关于主页在缓存中的商品信息和数据库里的信息做同步

题目 80：fastDFS 的执行流程，你清楚么？

（5） 问题分析：
考察对于分布式文件系统的使用及理解情况
（6） 核心答案讲解：



（7） 问题扩展：
无
（8） 结合项目中使用：
商家商品后台里对应的图片信息的录入

题目 81：zookeeper 集群，全部都挂了，怎么办？

（5） 问题分析：
考察面试者对 zookerper 的认知程度，已经具体业务中的处理方式
（6） 核心答案讲解：
注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
（7） 问题扩展：
本身不是高可用设计，master 撑不住高流量容易导致系统 crash
Zookeeper 选举速度很慢
难以避免数据的不一致
Zookeeper 性能是有限的
Zookeeper 在具体使用的时候会受到网络抖动的影响，有时候这些影响会造成灾难性的后果，例如网络发生问题的时候 Zookeeper 集群开始进行选主，如果选主时间持续太久，应用都会抛异常，并且可能导致 follower 不能及时跟上 leader 的情况，这个持续十分钟，会导致应用在这个时间内无法提供服务
*从任一节点到其子树中每个叶子节点的路径都包含相同数量的黑色节点
（8） 结合项目中使用：
作为 dubbo 的注册中心,暴露服务,然后消费方订阅服务用的
作为 solr 集群的调配中心,达到负载均衡的效果

题目 82：如何解决购物车内存大小的问题？

（5） 问题分析：

主要考核商城系统购物车存储方式，以及数据量问题

（6） 核心答案讲解：

1.Session（Memcached）方式

优点：购物车信息保存在服务端，可以保存 1M 信息。

缺点：对于大型网站会占有过多的服务器内存资源，造成服务器压力过大。Session 保存的信息会在用户退出登录后丢失。用户下次登录，购物车中商品信息丢失，用户只能从新选择。

2. Cookie 方式

优点：购物车信息存储在客户端，不占用服务器资源，基本可以到达持久化存储。

缺点：Cookie 有大小的限制，不能超过 4K，而且不够安全。

3.数据库存储

优点：持久化存储，可以分析用户购买行为。

缺点： 网站速度变慢，成本和维护增加。

（7） 问题扩展

数据库存储，不会像 cookie 那种容易丢失，也没有客户端的限制，你想怎么存，存多少都行，当数据太多我们可以进行分表分库处理。

（8） 结合项目中使用

品优购项目中的购物车使用 redis+Cookie

题目 83：springboot、springcloud 等微服务的概念和使用？

（1）问题分析：

主要是考核 springboot，springcloud 的基本概念

（2）核心答案讲解：

Spring boot 是 Spring 的一套快速配置脚手架，对第三方技术进行了很好的封装和整合，提供了大量第三方接口；可以通过依赖自动配置，不需要 XML 等配置文件

spring-colud 是一种云端分布式架构解决方案，基于 spring boot,在 spring boot 做较少的配置，便可成为 spring cloud 中的一个微服务

（3）问题扩展:

Spring Cloud 主要的组件,以及它的访问流程

1、外部或者内部的非 Spring Cloud 目都统一通过 API 网关(Zuul)来访可内

部服务.
2、网关接收到请求后,从注册中心(Eureka)获取可用服务
3、由 Ribbon 进行均负载后,分发到后端的具体实例
4、微服务之间通过 Feign 进行通信处理业务
5、Hystrix 负责处理服务超时熔断
6、Turbine 监控服务间的调用和焠断相关指标

（4）结合项目中使用:
结合十次方项目，全部采用微服务框架，更好的理解微服务

题目 84：SpringDataJPA 怎样使用？

（1）问题分析：
考官主要考核 SpringDataJPA 的用法。
（2）核心答案讲解：
1.导入 SpringDataJPA 依赖
2.添加配置
3.添加注解。
（3）问题扩展:
无
（4）结合项目中使用:
十次方 dao 层使用 SpringDataJPA