目录

一 Java 基础 4
1.1 Collection 和 Map 4
1.2 多线程 6
1.3 IO 9
1.4 反射 10
二 web 12
2.1 HTTP 请求的 GET 与 POST 方式的区别 12
2.2 session 与 cookie 区别 12
2.3 JDBC 流程 14
2.4 MVC 设计思想 15
2.5 Ajax 15
2.6 设计模式 17
2.7 JS 选择器 19
2.8 内置对象 20
三 数据库 22
3.1 Sql 之连接查询 22
3.2 Sql 之聚合函数： 22
3.3 Sql 之 SQL 注入 22
3.4 SQL Select 语句完整的执行顺序: 23
3.5 什么是存储过程？它有什么优点？ 23
3.6 Mysql 性能优化举例 24
四 JAVA 框架 27
4.1 Spring 27
4.2 SpringMVC 32
4.3 Mybatis 35
4.4 Struts2 36
4.5 Hibernate 39
五 电商项目 41
5.1.电商行业技术特点 41
5.2.系统功能 42
5.3.本系统人员配置情况 42
5.4.开发流程 42
5.5.后台开发环境 43
5.6.涉及技术 43
5.7 开发工具和环境 43
5.8 商城的 6 大模块 44
六 电商项目中的技术点 50
6.1 Nginx（web 服务器） 50
6.2 Httpclient 53
6.3 JSONP 54
6.4 Redis（缓存数据库） 54
6.5 Quartz 56

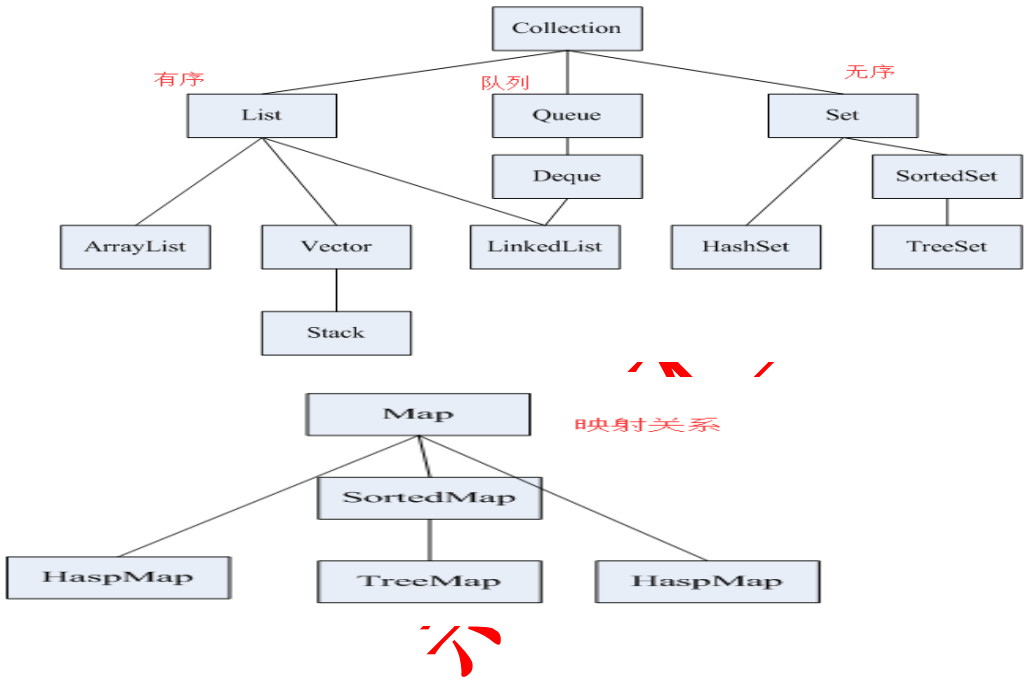
6.6 MQ 57
6.7 Dubbo 58
6.8 Solr 59
七 电商项目面试问题 60
7.1 简单的介绍一下你这个项目 60
7.2 整个项目的架构如何？ 60
7.3 这个项目为用户提供了哪些服务？包括哪些功能？ 62
7.4 你承担这个项目的哪些核心模块？ 62
7.5 这些模块的实现思路说一下？ 63
7.6 项目中哪些功能模块涉及了大数据量访问？你是如何解决的？ 67
7.7 在做这个项目的时候你碰到了哪些问题？你是怎么解决的？ 68
7.8 你做完这个项目后有什么收获？ 68
7.9 你这个项目中使用什么构建的？多模块开发是如何划分的呢？为什么要这么做？ 70
7.10 你觉得在图片上传功能上面需要注意什么？ 71
7.11 在你这个项目中，是如何设计商品规格的？ 72
7.12 在这个项目中你是如何实现跨系统调用的？ 72
7.13 你这个项目中 CMS 系统是如何设计的，简单的说一下其设计思想？ 73
7.14 在这个项目中，你们主要使用什么样的数据格式来进行数据的传输的？你对 JSON 了解么？能说说 JSON 对象如何转换成 Java 对象的？ 74
7.15 单点系统的设计思想你了解吗？他在系统架构中的作用是什么？位置如何？ 75
7.16 你们这个项目中订单 ID 是怎么生成的？我们公司最近打算做一个电商项目，如果让你设计这块，你会考虑哪些问题？ 77
7.17 各个服务器的时间不统一怎么办？ 78
7.18 在问题 17 的基础上，可能存在毫秒级的偏差情况，怎么办？ 78
7.19 你们线上部署时什么样的，能画一下吗？ 78
7.20.如何解决并发问题的？ 78
7.21 你们生产环境的服务器有多少台？（重点以 web 服务器为主） 79
7.22.数据备份是怎么做的？有没有做读写分离？ 79
7.23 你们使用什么做支付的？如果使用易宝做支付，请求超时了怎么处理？ 79
7.24 付款成功后易宝会有数据返回吗？如果付款后易宝没有返回，或者返回超时了，但是钱又已经扣了，你怎么办？ 79
7.25 你们怎么做退款功能的，要多长时间才能把钱退回给用户？ 80
7.26 不同域名的网站如何实现用户信息共享 80
7.27 点一个链接访问到一个页面，这个页面上既有静态数据，又有动态数据（需要查数据库的），打开这个页面的时候就是很慢但是也能打开。怎么解决这个问题，怎么优化？(静态化) 81
7.28 如果用户一直向购物车添加商品怎么办？并且他添加一次你查询一次数据库？互联网上用户那么多，这样会对数据库造成很大压力你怎么办？(购物车 redis 存储) 83
7.29 做促销时，商品详情页面的静态页面如何处理价格问题。 83
7.30 一个电商项目，在 tomcat 里面部署要打几个 war 包？ 85

7.31 你说你用了 redis 缓存，你 redis 存的是什么格式的数据，是怎么存的？ 85
7.32 购物车知识补充(在设计购物车时需要注意哪些细节) 87
八 传统项目 89
8.1 什么是 BOS ？ 89
8.2 什么是 EasyUI? 90
8.3Activity 工作流 93
8.4Apache POI 报表技术 95
8.5HibernateSearch 全文搜索 96
8.6ApacheShiro 权限控制 98
8.7WebService 100
8.8EhCache 102
8.9Highcharts 103
8.10BOS 中的完成的功能模块 103

一 Java 基础

1.1 Collection 和 Map

1.1.1 掌握 Collection 和 Map 的继承体系



1.1.2 List 和 Set 区别

List,Set 都是继承自 Collection 接口

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，重复元素会覆盖掉

（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，加入 Set 的 Object 必须定义 equals()方法 ，另外 list 支持 for 循环，也就是通过下标来遍历，也可以用迭代器，但是 set 只能用迭代，因为他无序，无法用下标来取得想要的值。）

Set 和 List 对比：

Set：检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。

List：和数组类似，List 可以动态增长，查找元素效率高，插入删除元素效率低，因为会引起其他元素位置改变。

1.1.3 List 和 Map 区别

List 是对象集合，允许对象重复。

Map 是键值对的集合，不允许 key 重复

1.1.4 ArrayList 与 Vector 区别

ArrayList 和 Vector 都是用数组实现的，主要有这么三个区别：

Vector 是多线程安全的，线程安全就是说多线程访问同一代码，不会产生不确定的结果。而 ArrayList 不是，这个可以从源码中看出，Vector 类中的方法很多有 synchronized 进行修饰，这样就导致了 Vector 在效率上无法与 ArrayList 相比；

两个都是采用的线性连续空间存储元素，但是当空间不足的时候，两个类的增加方式是不同。

Vector 可以设置增长因子，而 ArrayList 不可以。

Vector 是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用。

适用场景分析：

Vector 是线程同步的，所以它也是线程安全的，而 ArrayList 是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用 ArrayList 效率比较高。

如果集合中的元素的数目大于目前集合数组的长度时，在集合中使用数据量比较大的数据，用 Vector 有一定的优势。

1.1.5 Arraylist 与 LinkedList 区别

Arraylist：

优点：ArrayList 是实现了基于动态数组的数据结构,因为地址连续，一旦数据存储好了，查询操作效率会比较高（在内存里是连着放的）。

缺点：因为地址连续， ArrayList 要移动数据,所以插入和删除操作效率比较低。

LinkedList：

优点：LinkedList 基于链表的数据结构,地址是任意的，所以在开辟内存空间的时候不需要等一个连续的地址，对于新增和删除操作 add 和 remove，LinedList 比较占优势。LinkedList 适用于要头尾操作或插入指定位置的场景

缺点：因为 LinkedList 要移动指针,所以查询操作性能比较低。

适用场景分析：

当需要对数据进行对此访问的情况下选用 ArrayList，当需要对数据进行多次增加删除修改时采用 LinkedList。

1.1.6 HashMap 和 Hashtable 的区别

1. hashMap 去掉了 HashTable 的 contains 方法,但是加上了 containsValue（）和 containsKey（）方法。

2. hashTable 同步的，而 HashMap 是非同步的，效率上逼 hashTable 要高。

3. hashMap 允许空键值，而 hashTable 不允许。

注意：

TreeMap：非线程安全基于红黑树实现。TreeMap 没有调优选项，因为该树总处于平衡状态。

Treemap：适用于按自然顺序或自定义顺序遍历键(key)。

1.1.7 HashSet 和 HashMap 区别

set 是线性结构,set 中的值不能重复,hashset 是 set 的 hash 实现,hashset 中值不能重复是用 hashmap 的 key 来实现的。

map 是键值对映射，可以空键空值。HashMap 是 Map 接口的 hash 实现，key 的唯一性是通过 key 值 hash 值的唯一来确定，value 值是则是链表结构。

他们的共同点都是 hash 算法实现的唯一性，他们都不能持有基本类型，只能持有对象

1.2 多线程

1.2.1 创建线程的方式及实现

Java 中创建线程主要有三种方式：

一、继承 Thread 类创建线程类

（1）定义 Thread 类的子类，并重写该类的 run 方法，该 run 方法的方法体就代表了线程要完成的任务。因此把 run()方法称为执行体。

（2）创建 Thread 子类的实例，即创建了线程对象。

（3）调用线程对象的 start()方法来启动该线程。

二、通过 Runnable 接口创建线程类

（1）定义 runnable 接口的实现类，并重写该接口的 run()方法，该 run()方法的方法体同样是该线程的线程执行体。

（2）创建 Runnable 实现类的实例，并依此实例作为 Thread 的 target 来创建 Thread 对象，该 Thread 对象才是真正的线程对象。

（3）调用线程对象的 start()方法来启动该线程。

三、通过 Callable 和 Future 创建线程

（1）创建 Callable 接口的实现类，并实现 call()方法，该 call()方法将作为线程执行体，并且有返回值。

（2）创建 Callable 实现类的实例，使用 FutureTask 类来包装 Callable 对象，该 FutureTask 对象封装了该 Callable 对象的 call()方法的返回值。

（3）使用 FutureTask 对象作为 Thread 对象的 target 创建并启动新线程。

（4）调用 FutureTask 对象的 get()方法来获得子线程执行结束后的返回值

1.2.2 创建线程的三种方式的对比

采用实现 Runnable、Callable 接口的方式创见多线程时，优势是：线程类只是实现了 Runnable 接口或 Callable 接口，还可以继承其他类。

在这种方式下，多个线程可以共享同一个 target 对象，所以非常适合多个相同线程来处理同一份资源的情况，从而可以将 CPU、代码和数据分开，形成清晰的模型，较好地体现了面向对象的思想。

劣势是：编程稍微复杂，如果要访问当前线程，则必须使用 Thread.currentThread()方法。

使用继承 Thread 类的方式创建多线程时优势是：编写简单，如果需要访问当前线程，则无需使用 Thread.currentThread()方法，直接使用 this 即可获得当前线程。

劣势是：线程类已经继承了 Thread 类，所以不能再继承其他父类。

1.2.3 线程池的几种方式

`newFixedThreadPool(int nThreads)`

创建一个固定长度的线程池，每当提交一个任务就创建一个线程，直到达到线程池的最大数量，这时线程规模将不再变化，当线程发生未预期的错误而结束时，线程池会补充一个新的线程

`newCachedThreadPool()`

创建一个可缓存的线程池，如果线程池的规模超过了处理需求，将自动回收空闲线程，而当需求增加时，则可以自动添加新线程，线程池的规模不存在任何限制

newSingleThreadExecutor()

这是一个单线程的 Executor，它创建单个工作线程来执行任务，如果这个线程异常结束，会创建一个新的来替代它；它的特点是能确保依照任务在队列中的顺序来串行执行

newScheduledThreadPool(int corePoolSize)

创建了一个固定长度的线程池，而且以延迟或定时的方式来执行任务，类似于 Timer。

1.2.4 线程的生命周期

生命周期的五种状态

新建（new Thread）

当创建 Thread 类的一个实例（对象）时，此线程进入新建状态（未被启动）。例如：Thread t1=new Thread();

就绪（runnable）

线程已经被启动，正在等待被分配给 CPU 时间片，也就是说此时线程正在就绪队列中排队等候得到 CPU 资源。例如：t1.start();

运行（running）

线程获得 CPU 资源正在执行任务（run()方法），此时除非此线程自动放弃 CPU 资源或者有优先级更高的线程进入，线程将一直运行到结束。

死亡（dead）

当线程执行完毕或被其它线程杀死，线程就进入死亡状态，这时线程不可能再进入就绪状态等待执行。

自然终止：正常运行 run()方法后终止

异常终止：调用 stop()方法让一个线程终止运行

堵塞（blocked）

由于某种原因导致正在运行的线程让出 CPU 并暂停自己的执行，即进入堵塞状态。

正在睡眠：用 sleep(long t) 方法可使线程进入睡眠方式。一个睡眠着的线程在指定的时间过去可进入就绪状态。

正在等待：调用 wait()方法。（调用 motify()方法回到就绪状态）

被另一个线程所阻塞：调用 suspend()方法。（调用 resume()方法恢复）

1.2.5 悲观锁 乐观锁

乐观锁 悲观锁

是一种思想。可以用在很多方面。

比如数据库方面。

悲观锁就是 for update（锁定查询的行）
乐观锁就是 version 字段（比较跟上一次的版本号，如果一样则更新，如果失败则要重复读-比较-写的操作。）

JDK 方面

悲观锁就是 sync
乐观锁就是原子类（内部使用 CAS 实现）

本质来说，就是悲观锁认为总会有人抢我的。
乐观锁就认为，基本没人抢。

1.2.5.1 乐观锁的业务场景及实现方式

乐观锁（Optimistic Lock）：

每次获取数据的时候，都不会担心数据被修改，所以每次获取数据的时候都不会进行加锁，但是在更新数据的时候需要判断该数据是否被别人修改过。如果数据被其他线程修改，则不进行数据更新，如果数据没有被其他线程修改，则进行数据更新。由于数据没有进行加锁，期间该数据可以被其他线程进行读写操作。

乐观锁：比较适合读取操作比较频繁的场景，如果出现大量的写入操作，数据发生冲突的可能性就会增大，为了保证数据的一致性，应用层需要不断的重新获取数据，这样会增加大量的查询操作，降低了系统的吞吐量。

1.3 IO

1.3.1 什么是 IO 流？

它是一种数据的流从源头流到目的地。比如文件拷贝，输入流和输出流都包括了。输入流从文件中读取数据存储到进程(process)中，输出流从进程中读取数据然后写入到目标文件。

1.3.2 字节流和字符流的区别

字节流在 JDK1.0 中就被引进了，用于操作包含 ASCII 字符的文件。JAVA 也支持其他的字符如 Unicode，为了读取包含 Unicode 字符的文件，JAVA 语言设计者在 JDK1.1 中引入了字符流。ASCII 作为 Unicode 的子集，对于英语字符的文件，可以使用字节流也可以使用字符流。

1.3.3 Java 中流类的超类主要有那些？

- java.io.InputStream
- java.io.OutputStream
- java.io.Reader
- java.io.Writer

1.3.4 FileInputStream 和 FileOutputStream 是什么？

这是在拷贝文件操作的时候，经常用到的两个类。在处理小文件的时候，它们性能表现还不错，在大文件的时候，最好使用 BufferedInputStream（或 BufferedReader）和 BufferedOutputStream (或 BufferedWriter)

1.3.5 字节流和字符流，你更喜欢使用哪一个？

个人来说，更喜欢使用字符流，因为他们更新一些。许多在字符流中存在的特性，字节流中不存在。比如使用 BufferedReader 而不是 BufferedInputStreams 或 DataInputStream，使用 newLine()方法来读取下一行，但是在字节流中我们需要做额外的操作。

1.4 反射

1.4.1 什么是 Java 的反射

Java 的反射机制是在编译并不确定是哪个类被加载了，而是在程序运行的时候才加载、探知、自审。使用在编译期并不知道的类。这样的特点就是反射。

1.4.2 Java 反射有什么作用

Java 的反射机制它知道类的基本结构，这种对 Java 类结构探知的能力，我们称为 Java 类的“自审”。大家都用过 eclipse。当我们构建出一个对象的时

候，去调用该对象的方法和属性的时候。一按点，编译工具就会自动的把该对象能够使用的所有的方法和属性全部都列出来，供用户进行选择。这就是利用了 Java 反射的原理，是对我们创建对象的探知、自审。

1.4.3 反射的用途及实现

Java 反射的主要功能：

- 确定一个对象的类
- 取出类的 modifiers,数据成员,方法,构造器,和超类.
- 找出某个接口里定义的常量和方法说明.
- 创建一个类实例,这个实例在运行时刻才有名字(运行时间才生成的对象).
- 取得和设定对象数据成员的值,如果数据成员名是运行时刻确定的也能做到.
- 在运行时刻调用动态对象的方法.
- 创建数组,数组大小和类型在运行时刻才确定,也能更改数组成员的值.

反射的应用

spring 的 ioc/di 也是反射….

javaBean 和 jsp 之间调用也是反射….

struts 的 FormBean 和页面之间…也是通过反射调用….

JDBC 的 classForName()也是反射…..

hibernate 的 find(Class clazz) 也是反射….

二 web

2.1 HTTP 请求的 GET 与 POST 方式的区别

1. get 是把参数数据队列加到提交表单的 ACTION 属性所指的 URL 中，在 URL 中可以看到。

2.post 是通过 HTTPPOST 机制，将表单内各个字段与其内容放置在 HTML HEADER 内一起传送到 ACTION 属性所指的 URL 地址。用户看不到这个过程。

3. get 安全性非常低，post 安全性较高。但是执行效率却比 Post 方法好。

4. get 传送的数据量较小，不能大于 2KB。post 传送的数据量较大，一般被默认为不受限制。但理论上，IIS4 中最大量为 80KB，IIS5 中为 100KB。

5. 对于 get 方式，服务器端用 Request.QueryString 获取变量的值，对于 post 方式，服务器端用 Request.Form 获取提交的数据。

所以综上建议：

1、包含机密信息的话，建议用 Post 数据提交方式；

2、在做数据查询时，可以使用 Get 方式；而在做数据添加、修改或删除时，建议用 Post 方式；

2.2 session 与 cookie 区别

Cookie与Session的区别

Cookie	Session
存储在客户端	存储在服务器端
两种类型 ※有生命周期 ※无 生命 周期	两种实现方式 ※依赖于cookie ※url重写
父路径不能访问子路径的cookie	同一个session 的窗口共享一个 session
典型应用： ※3个月不用再登陆 ※购物车	典型应用： ※用户登陆 ※购物车也可以用session 实现。
不可靠	可靠

1:cookie 数据存放在客户的浏览器上(客户端),session 数据放

@1: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,如果主要考虑到安全应当使用 session

@2: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,如果主要考虑到减轻服务器性能方面,应当使用 cookie

2:单个 cookie 在客户端的限制是 3K,就是说一个站点在客户端存放的 cookie 不能 3K

@3: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

3:session 不能区分路径,同一个用户在访问一个网站期间,所有的 session 在任何一个地方都可以访问到.而 cookie 中如果设置了路径参数,那么同一个网站中不同路径下的 cookie 互相是访问不到的.cookie 只能是子路径访问父路径设置的 cookie

4.分析:

http 是无状态的协议,客户每次读取 web 页面时,服务器都打开新的会话,而且服务器也不会自动维护客户的上下文信息,那么要怎么才能实现网上商店中的购物车呢:

session 就是一种保存上下文信息的机制,它是针对每一个用户的,变量的值保存在服务器端,通过 SessionID 来区分不同的客 户,session 是以 cookie 或 URL 重写 为基础的。

默认使用 cookie 来实现,系统会创造一个名为 JSESSIONID 的输出 cookie，我 们叫做 session cookie,以区分 persistent cookie(我们通常所说的 cookie)

注意 session cookie 是存储于浏览器内存中的，并不是写到硬盘上的，这也就是我们刚才看到的 JSESSIONID，我们通常情是看不到 JSESSIONID 的，但 是当我们把浏览器的 cookie 禁止后，web 服务器会采用 URL 重写的方式传递 Sessionid,我们就可以在地址栏看到 sessionid=KWJHUG6JJM65HS2K6 之类的字符串。

明白了原理，我们就可以很容易的分辨出 persistent cookie 和 session cookie 的区别了，网上那些关于两者安全性的讨论也就一目了然了，session cookie 针对某一次会话而言，会话结束 session cookie 也就随着消失了,而 persistent cookie 只是存在于客户端硬盘上的一段文本(通常是加密的)，而且可能会遭到 cookie 欺骗以及针对 cookie 的跨站脚本攻击,自然不如 session cookie 安全了。

5:通常 session cookie 是不能跨窗口使用的,当你新开了一个浏览器窗口进入相同页面时,系统会赋予你一个新的 sessionid,这样我们信息共享的目的就达不到了,此时我们可以先把 sessionid 保存在 persistent cookie 中,然后在新窗口中读出来,就可以得到上一个窗口 SessionID 了,这样通过 session cookie 和 persistent cookie 的结合我们就实现了跨窗口的 session tracking()会话跟踪)

2.3 JDBC 流程

1、 加载 JDBC 驱动程序：

在连接数据库之前，首先要加载想要连接的数据库的驱动到 JVM（Java 虚拟机），
这通过 java.lang.Class 类的静态方法 forName(String className)实现。

2、 提供 JDBC 连接的 URL

连接 URL 定义了连接数据库时的协议、子协议、数据源标识。
- 书写形式：协议：子协议：数据源标识

协议：在 JDBC 中总是以 jdbc 开始 子协议：是桥连接的驱动程序或是数据库管理系统名称。
数据源标识：标记找到数据库来源的地址与连接端口。

3、创建数据库的连接

要连接数据库，需要向 java.sql.DriverManager 请求并获得 Connection 对象， 该对象就代表一个数据库的连接。
- 使用 DriverManager 的 getConnection(String url , String username , String password)方法传入指定的欲连接的数据库的路径、数据库的用户名和 密码来获得。

4、 创建一个 Statement

要执行 SQL 语句，必须获得 java.sql.Statement 实例，Statement 实例分为以下 3 种类型：
1、执行静态 SQL 语句。通常通过 Statement 实例实现。
2、执行动态 SQL 语句。通常通过 PreparedStatement 实例实现。
3、执行数据库存储过程。通常通过 CallableStatement 实例实现。
具体的实现方式：
Statement stmt = con.createStatement() ; PreparedStatement pstmt = con.prepareStatement(sql) ; CallableStatement cstmt = con.prepareCall(“{CALL demoSp(? , ?)}”) ;

5、执行 SQL 语句

Statement 接口提供了三种执行 SQL 语句的方法：executeQuery 、executeUpdate 和 execute
1、ResultSet executeQuery(String sqlString)：执行查询数据库的 SQL 语句 ，返回一个结果集（ResultSet）对象。
2、int executeUpdate(String sqlString)：用于执行 INSERT、UPDATE 或 DELETE 语句以及 SQL DDL 语句，如：CREATE TABLE 和 DROP TABLE 等
3、execute(sqlString):用于执行返回多个结果集、多个更新计数或二者组合的 语句。 具体实现的代码：
ResultSet rs = stmt.executeQuery(“SELECT * FROM …”) ; int rows =

stmt.executeUpdate("INSERT INTO …") ; boolean flag = stmt.execute(String sql) ;

6、处理结果

两种情况：

1、执行更新返回的是本次操作影响到的记录数。
2、执行查询返回的结果是一个 ResultSet 对象。
• ResultSet 包含符合 SQL 语句中条件的所有行，并且它通过一套 get 方法提供了对这些 行中数据的访问。
• 使用结果集（ResultSet）对象的访问方法获取数据：

```
while(rs.next()){
String name = rs.getString("name") ;
String pass = rs.getString(1) ; //  此方法比较高效
}
```

（列是从左到右编号的，并且从列 1 开始）

7、关闭 JDBC 对象

操作完成以后要把所有使用的 JDBC 对象全都关闭，以释放 JDBC 资源，关闭顺序和声 明顺序相反：
1、关闭记录集
2、关闭声明
3、关闭连接对象

2.4 MVC 设计思想

MVC 就是

M:Model 模型
V:View 视图
C:Controller 控制器
模型就是封装业务逻辑和数据的一个一个的模块,控制器就是调用这些模块的(java 中通常是用 Servlet 来实现,框架的话很多是用 Struts2 来实现这一层),视图就主要是你看到的,比如 JSP 等.
当用户发出请求的时候,控制器根据请求来选择要处理的业务逻辑和要选择的数据,再返回去把结果输出到视图层,这里可能是进行重定向或转发等.

2.5 Ajax

Ajax 并不算是一种新的技术，全称是 asychronous javascript and xml，可以说是已有技术的组合，主要用来实现客户端与服务器端的异步通信效果，实现页面的局部刷新

2.5.1 页面编码和被请求的资源编码如果不一致如何处理？

对于 ajax 请求传递的参数，如果是 get 请求方式，参数如果传递中文，在有些浏览器会乱码，不同的浏览器对参数编码的处理方式不同，所以对于 get 请求的参数需要使用 encodeURIComponent 函数对参数进行编码处理，后台开发语言都有相应的解码 api。对于 post 请求不需要进行编码

2.5.2 Ajax 的过程

1. 创建 XMLHttpRequest 对象,也就是创建一个异步调用对象
2. 创建一个新的 HTTP 请求,并指定该 HTTP 请求的方法、URL 及验证信息
3. 设置响应 HTTP 请求状态变化的函数
4. 发送 HTTP 请求
5. 获取异步调用返回的数据
6. 使用 JavaScript 和 DOM 实现局部刷新

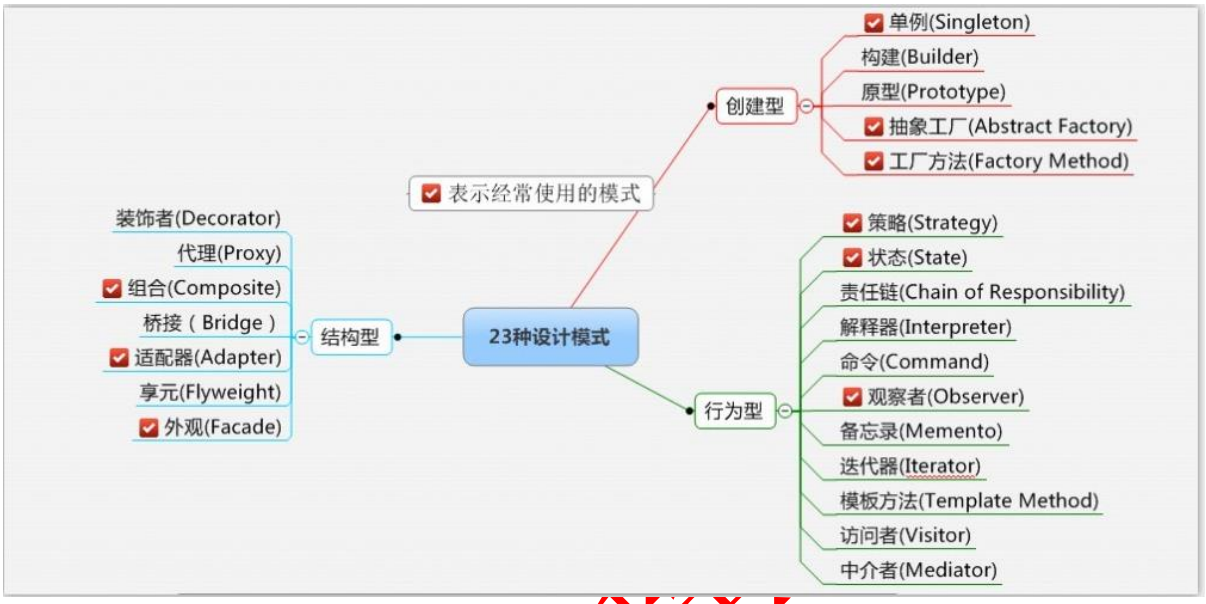
2.5.3 阐述一下异步加载 JS

1. 异步加载的方案： 动态插入 script 标签
2. 通过 ajax 去获取 js 代码，然后通过 eval 执行
3. script 标签上添加 defer 或者 async 属性
4. 创建并插入 iframe，让它异步执行 js

2.5.4 ajax 请求时，如何解释 json 数据

使用 eval() 或者 JSON.parse() 鉴于安全性考虑，推荐使用 JSON.parse()更靠谱，对数据的安全性更好.

2.6 设计模式



2.6.1 单例模式

通用代码：（是线程安全的）

```
public class Singleton {
    private static final Singleton singleton = new Singleton();
//限制产生多个对象
    private Singleton(){
    }
    //通过该方法获得实例对象
    public static Singleton getSingleton(){
            return singleton;
    }
    //类中其他方法，尽量是 static
    public static void doSomething(){
    }
}
```

使用场景：

- 要求生成唯一序列号的环境；

● 在整个项目中需要一个共享访问点或共享数据，例如一个 Web 页面上的计数器，可以不用把每次刷新都记录到数据库中，使用单例模式保持计数器的值，并确保是线程安全的；

● 创建一个对象需要消耗的资源过多，如要访问 IO 和数据库等资源；

● 需要定义大量的静态常量和静态方法（如工具类）的环境，可以采用单例模式（当然，也可以直接声明为 static 的方式）。

线程不安全实例：

```
public class Singleton {
    private static Singleton singleton = null;
    //限制产生多个对象
    private Singleton(){
    }
    //通过该方法获得实例对象
    public static Singleton getSingleton(){
        if(singleton == null){
            singleton = new Singleton();
        }
        return singleton;
    }
}
```

解决办法：

在 getSingleton 方法前加 synchronized 关键字，也可以在 getSingleton 方法内增加 synchronized 来实现。最优的办法是如通用代码那样写。

2.6.2 工厂模式

Product 为抽象产品类负责定义产品的共性，实现对事物最抽象的定义；
Creator 为抽象创建类，也就是抽象工厂，具体如何创建产品类是由具体的实现工厂 ConcreteCreator 完成的。

具体工厂类代码：

```
public class ConcreteCreator extends Creator {
public <T extends Product> T createProduct(Class<T> c){
        Product product=null;
        try {
            product =
(Product)Class.forName(c.getName()).newInstance();
        } catch (Exception e) {
```

```
                    //异常处理
                }
                return (T)product;
            }
        }
```

简单工厂模式：

一个模块仅需要一个工厂类，没有必要把它产生出来，使用静态的方法

多个工厂类：

每个人种（具体的产品类）都对应了一个创建者，每个创建者独立负责创建对应的产品对象，非常符合单一职责原则

代替单例模式：

单例模式的核心要求就是在内存中只有一个对象，通过工厂方法模式也可以只在内存中生产一个对象

延迟初始化：

ProductFactory 负责产品类对象的创建工作，并且通过 prMap 变量产生一个缓存，对需要再次被重用的对象保留

使用场景：jdbc 连接数据库，硬件访问，降低对象的产生和销毁

2.7 JS 选择器

原生 JS 选择器有 getElementById、getElementsByName、getElementsByTagName 和 getElementsByClassName 这四个

1. getElementById(通过 ID 获取元素)

用法:document.getElementById("Id");Id 为要获取的元素的 id 属性值。

2. getElementsByName(通过 name 属性获取元素)

用法:document.getElementsByName("Name");Name 为要获取元素的 name 属性值,这个方法一般适用于提交表单数据,当元素为 form、img、iframe、applet、embed、object 的时候设置 name 属性时,会自动在 Document 对象中创建以该 name 属性值命名的属性。所以可以通过 document.domName 引用相应的 dom 对象

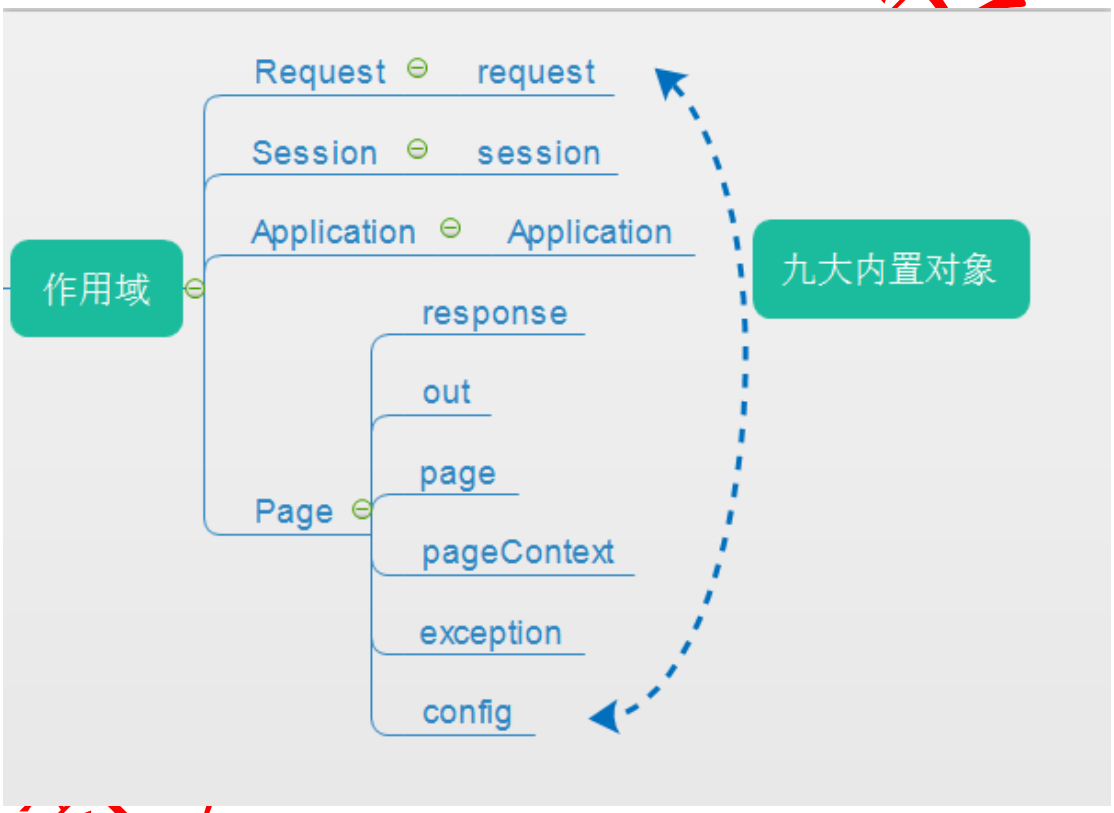
3. getElementsByTagName(通过元素名称获取元素)

用法:document.getElementsByTagName(TagName);TagName 为要获取元素的标签名称,当 TagName 为*的时候表示获取所有的元素,document 也可以换成 DOM 元素,但是这样就只能获取到该 DOM 元素后面的子集元素。

4. getElementsByClassName(通过 CSS 类来获取元素)

用法:document.getElementsByClassName(ClassName);ClassName 为要获取元素的 CSS 类名称,如果要同时获取多个的话,在每个 CSS 类后面用空格隔开。 如 document.getElementsByClassName("class2 class1")就会获取到 class1 和 class2 样式的元素,document 也可以换成 DOM 元素,这样也是只能获取到该 DOM 元素后面的子集元素。

2.8 内置对象



1 request 对象：

代表的是来自客户端的请求，例如我们在 FORM 表单中填写的信息等，是最常用的对象。客户端的请求信息被封装在 request 对象中，通过它才能了解到客户的需求，然后做出响应。它是 HttpServletRequest 类的实例。作用域为 request(用户请求期）。

2 response 对象：

对象代表的是对客户端的响应，也就是说可以通过 response 对象来组织发送到客户端的数据。但是由于组织方式比较底层，所以不建议普通读者使

用，需要向客户端发送文字时直接使用。它是 HttpServletResponse 类的实例。作用域为 page（页面执行期）。

3 session 对象：

指的是客户端与服务器的一次会话，从客户连到服务器的一个 WebApplication 开始，直到客户端与服务器断开连接为止。它是 HttpSession 类的实例。作用域 session(会话期—）。

4 out 对象 ：

out 对象是 JspWriter 类的实例,是向客户端输出内容常用的对象，作用域为 page（页面执行期）

5 page 对象：

page 对象就是指向当前 JSP 页面本身，有点象类中的 this 指针，它是 java.lang.Object 类的实例 。“page” 对象代表了正在运行的由 JSP 文件产生的类对象，不建议一般读者使用。 作用域 page

6 application 对象：

实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭，在此期间，此对象将一直存在；这样在用户的前后连接或不同用户之间的连接中，可以对此对象的同一属性进行操作；在任何地方对此对象属性的操作，都将影响到其他用户对此的访问。服务器的启动和关闭决定了 application 对象的生命。它是 ServletContext 类的实例。作用域 application

7 pageContext 对象：

提供了对 JSP 页面内所有的对象及名字空间的访问，也就是说他可以访问到本页所在的 SESSION，也可以取本页面所在的 application 的某一属性值，他相当于页面中所有功能的集大成者，它的本类名也叫 pageContext。用域 Pageconfig 对象

8 config 对象：

config 对象是在一个 Servlet 初始化时，JSP 引擎向它传递信息用的，此信息包括 Servlet 初始化时所要用到的参数（通过属性名和属性值构成）以及服务器的有关信息（通过传递一个 ServletContext 对象）。作用域 Page

9 exception 对象：

是一个例外对象，当一个页面在运行过程中发生了例外，就产生这个对象。如果一个 JSP 页面要应用此对象，就必须把 isErrorPage 设为 true，否则无法编译。他实际上是 java.lang.Throwable 的对象.作用域 page

三 数据库

3.1 Sql 之连接查询

外连接：

1） 左连接（左外连接）以左表为基准进行查询,左表数据会全部显示出来,右表如果和左表匹配
的数据则显示相应字段的数据,如果不匹配,则显示为 NULL;
2） 右连接（右外连接）以右表为基准进行查询,右表数据会全部显示出来,右表如果和左表匹配的数据则显示相应字段的数据,如果不匹配,则显示为 NULL;
3） 全连接就是先以左表进行左外连接，然后以右表进行右外连接。

内连接：

显示表之间有连接匹配的所有行。

3.2 Sql 之聚合函数：

聚合函数是对一组值执行计算并返回单一的值的函数，它经常与 SELECT 语句的 GROUP BY 子句一同使用。

1).AVG 返回指定组中的平均值，空值被忽略； COUNT 返回指定组中项目的数量。

例：select prd_no,avg(qty) from sales group by prd_no

2). MAX 返回指定数据的最大值；MIN 返回指定数据的最小值；SUM 返回指定数据的和，只能用于数字列，空值被忽略。

例：select prd_no,max(qty) from sales group by prd_no

3）使用 group by 子句对数据进行分组；对 group by 子句形成的组运行聚集函数计算每一组的值；最后用 having 子句去掉不符合条件的组；having 子句中的每个元素也必须出现在 select 列表中。有些数据库例外，如 oracle.

例：select prd_no,max(qty) from sales group by prd_no having prd_no>10

3.3 Sql 之 SQL 注入

举例：

select admin from user where username='admin' or 'a'='a' and passwd=''or 'a'='a'

防止 SQL 注入，使用预编译语句是预防 SQL 注入的最佳方式，如

select admin from user where username=？And password=?

使用预编译的 SQL 语句语义不会发生改变，在 SQL 语句中，变量用问号？表示。像上面例子中，username 变量传递的'admin' or 'a'='a' 参数，也只会当作 username 字符串来解释查询，从根本上杜绝了 SQL 注入攻击的发生。

注意：使用 mybaits 时 mapper 中#方式能够很大程度防止 sql 注入，$方式无法防止 sql 注入.

3.4 SQL Select 语句完整的执行顺序:

from--->where--->group by--->having--->计算所有的表达式--->order by--->select 输出

3.5 什么是存储过程？它有什么优点？

答：存储过程是一组予编译的 SQL 语句，

它的优点有：

允许模块化程序设计，就是说只需要创建一次过程，以后在程序中就可以调用该过程任意次。

允许更快执行，如果某操作需要执行大量 SQL 语句或重复执行，存储过程比 SQL 语句执行的要快。

减少网络流量，例如一个需要数百行的 SQL 代码的操作有一条执行语句完成，不需要在网络中发送数百行代码。

更好的安全机制，对于没有权限执行存储过程的用户，也可授权他们执行存储过程。

1）MySQL 存储过程的创建

(1). 格式

MySQL 存储过程创建的格式：

```
CREATE PROCEDURE 存储过程名(参数列表)
BEGIN
    SQL 语句代码块
END
```

举例：

```
CREATE PROCEDURE proc1(OUT s int)
    BEGIN
        SELECT COUNT(*) INTO s FROM user;
    END
```

（了解） (2). 参数： MySQL 存储过程参数有三种类型：in、out、inout。

如果仅仅想把数据传给 MySQL 存储过程，那就使用“in”类型参数；如果仅仅从 MySQL 存储过程返回值，那就使用“out”类型参数；如果需要把数据传给 MySQL 存储过程，还要经过一些计算后再传回给我们，此时，要使用“inout”类型参数。

(3). Mysql 调用储存过程

```
Set @n=1                //声明变量
Call procName(@n)   //调用储存过程
```

3.6 Mysql 性能优化举例

1） 当只要一行数据时使用 LIMIT

当你查询表的有些时候，你已经知道结果只会有一条结果，在这种情况下，加上 LIMIT 1 可以增加性能。这样一样，MySQL 数据库引擎会在找到一条数据后停止搜索，而不是继续往后查少下一条符合记录的数据。

2）选择正确的存储引擎

在 MySQL 中有两个存储引擎 MyISAM 和 InnoDB，每个引擎都有利有弊。

MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要 update 一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM 对于 SELECT COUNT(*) 这类的计算是超快无比的。

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

3）用 Not Exists 代替 Not In

Not Exists 允许用户使用相关子查询已排除一个表中能够与另一个表成功连接的所有记录。Not Exists 用到了连接，能够发挥已经建好的索引的作用，而 Not In 不能使用索引。Not In 是最慢的方式，要同每条记录比较，在数据量比较大的查询中不建议使用这种方式。

Select a.mobileid from Log_user a where not exists (select b.mobileid from magazineitem b where b.mobileid=a.mobileid);

4）对操作符的优化 尽量不采用不利用索引的操作符

如：in ，not in ， is nul， is not null，<>等

某个字段你总要会经常用来做搜索，为其建立索引：

Mysql 中可以使用 alter table 语句来为表中的字段添加索引的基本语法是：

ALTER TABLE <表名> ADD INDEX (<字段>);

例：mysql> alter table test add index(t_name);

5)mysql 分库分表：

分库分表有垂直切分和水平切分两种。

垂直切分：

即将表按照功能模块、关系密切程度划分出来，部署到不同的库上。例如，我们会建立定义数据库 workDB、商品数据库 payDB、用户数据库 userDB、日志数据库 logDB 等，分别用于存储项目数据定义表、商品定义表、用户数据表、日志数据表等。

水平切分：当一个表中的数据量过大时，我们可以把该表的数据按照某种规则，例如 userID 散列，进行划分，然后存储到多个结构相同的表，和不同的库上。例如，我们的 userDB 中的用户数据表中，每一个表的数据量都很大，就可以把 userDB 切分为结构相同的多个 userDB：part0DB、part1DB 等，再将 userDB 上的用户数据表 userTable，切分为很多 userTable：userTable0、userTable1 等，然后将这些表按照一定的规则存储到多个 userDB 上。

3.3 应该使用哪一种方式来实施数据库分库分表，这要看数据库中数据量的瓶颈所在，并综合项目的业务类型进行考虑。

如果数据库是因为表太多而造成海量数据，并且项目的各项业务逻辑划分清晰、低耦合，那么规则简单明了、容易实施的垂直切分必是首选。

而如果数据库中的表并不多，但单表的数据量很大、或数据热度很高，这种情况之下就应该选择水平切分，水平切分比垂直切分要复杂一些，它将原本逻辑上属于一体的数据进行了物理分割，除了在分割时要对分割的粒度做好评估，考虑数据平均和负载平均，后期也将对项目人员及应用程序产生额外的数据管理负担。

在现实项目中，往往是这两种情况兼而有之，这就需要做出权衡，甚至既需要垂直切分，又需要水平切分。我们的游戏项目便综合使用了垂直与水平切分，我们首先对数据库进行垂直切分，然后，再针对一部分表，通常是用户数据表，进行水平切分。

单库多表：

随着用户数量的增加，user 表的数据量会越来越大，当数据量达到一定程度的时候对 user 表的查询会渐渐的变慢，从而影响整个 DB 的性能。如果使用 mysql, 还有一个更严重的问题是，当需要添加一列的时候，mysql 会锁表，期间所有的读写操作只能等待。

可以将 user 进行水平的切分，产生两个表结构完全一样的 user_0000,user_0001 等表，user_0000 + user_0001 + …的数据刚好是一份完整的数据。

多库多表：

随着数据量增加也许单台 DB 的存储空间不够，随着查询量的增加单台数据库服务器已经没办法支撑。这个时候可以再对数据库进行水平区分。

分库分表规则举例：

通过分库分表规则查找到对应的表和库的过程。如分库分表的规则是 user_id 除以 4 的方式，当用户新注册了一个账号，账号 id 的 123,我们可以通过 id 除以 4 的方式确定此账号应该保存到 User_0003 表中。当用户 123 登录的时候，我们通过 123 除以 4 后确定记录在 User_0003 中。

mysql 读写分离：

在实际的应用中，绝大部分情况都是读远大于写。Mysql 提供了读写分离的机制，所有的写操作都必须对应到 Master，读操作可以在 Master 和 Slave 机器上进行，Slave 与 Master 的结构完全一样，一个 Master 可以有多个 Slave,甚至 Slave 下还可以挂 Slave,通过此方式可以有效的提高 DB 集群的每秒查询率.

所有的写操作都是先在 Master 上操作，然后同步更新到 Slave 上，所以从 Master 同步到 Slave 机器有一定的延迟，当系统很繁忙的时候，延迟问题会更加严重，Slave 机器数量的增加也会使这个问题更加严重。

此外，可以看出 Master 是集群的瓶颈，当写操作过多，会严重影响到 Master 的稳定性，如果 Master 挂掉，整个集群都将不能正常工作。

所以，1. 当读压力很大的时候，可以考虑添加 Slave 机器的分式解决，但是当 Slave 机器达到一定的数量就得考虑分库了。 2. 当写压力很大的时候，就必须得进行分库操作。

四 JAVA 框架

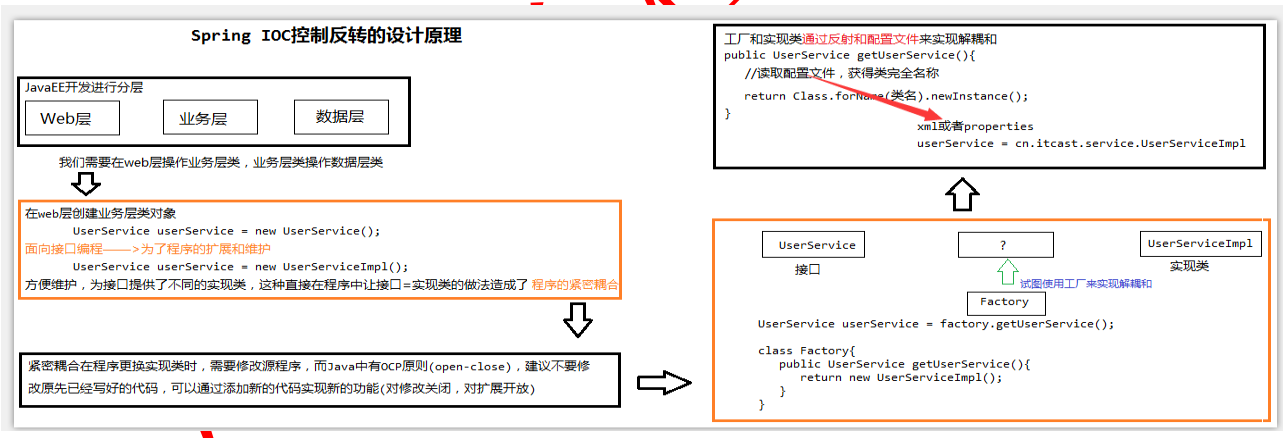
4.1 Spring

4.1.1 spring IOC 和 DI 的理解，它们有什么区别

IoC Inverse of Control 反转控制的概念，就是将原本在程序中手动创建 UserService 对象的控制权，交由 Spring 框架管理，简单说，就是创建 UserService 对象控制权被反转到了 Spring 框架

DI：Dependency Injection 依赖注入，在 Spring 框架负责创建 Bean 对象时，动态的将依赖对象注入到 Bean 组件

4.1.2 IOC 控制反转设计原理



4.1.3 BeanFactory 接口和 ApplicationContext 接口有什么区别

①ApplicationContext 接口继承 BeanFactory 接口，Spring 核心工厂是 BeanFactory ,BeanFactory 采取延迟加载，第一次 getBean 时才会初始化 Bean, ApplicationContext 是会在加载配置文件时初始化 Bean。

②ApplicationContext 是对 BeanFactory 扩展，它可以进行国际化处理、事件传递和 bean 自动装配以及各种不同应用层的 Context 实现

开发中基本都在使用 ApplicationContext, web 项目使用 WebApplicationContext，很少用到 BeanFactory

4.1.4 spring 配置 bean 实例化有哪些方式？

1）使用类构造器实例化(默认无参数)

```
<bean id="bean1" class="cn.itcast.spring.b_instance.Bean1"></bean>
```

2）使用静态工厂方法实例化(简单工厂模式)

```
//下面这段配置的含义：调用 Bean2Factory 的 getBean2 方法得到 bean2
<bean id="bean2" class="cn.itcast.spring.b_instance.Bean2Factory" factory-method="getBean2"></bean>
```

3）使用实例工厂方法实例化(工厂方法模式)

```
//先创建工厂实例 bean3Facory，再通过工厂实例创建目标 bean 实例
<bean id="bean3Factory" class="cn.itcast.spring.b_instance.Bean3Factory"></bean>
<bean id="bean3" factory-bean="bean3Factory" factory-method="getBean3"></bean>
```

4.1.5 spring 的生命周期

```
①instantiate bean 对象实例化
②populate properties 封装属性
③如果 Bean 实现 BeanNameAware 执行 setBeanName
④如果 Bean 实现 BeanFactoryAware 或者 ApplicationContextAware 设置工厂 setBeanFactory 或者上下文对象 setApplicationContext
⑤如果存在类实现 BeanPostProcessor（后处理 Bean） ，执行 postProcessBeforeInitialization，BeanPostProcessor 接口提供钩子函数，用来动态扩展修改 Bean。(程序自动调用后处理 Bean)
⑥如果 Bean 实现 InitializingBean 执行 afterPropertiesSet
⑦调用<bean init-method="init"> 指定初始化方法 init
⑧如果存在类实现 BeanPostProcessor（处理 Bean） ，执行 postProcessAfterInitialization
⑨执行业务处理
⑩如果 Bean 实现 DisposableBean 执行 destroy
⑪调用<bean destroy-method="customerDestroy"> 指定销毁方法 customerDestroy
```

4.1.6 Bean 注入属性有哪几种方式

```
1.接口注入
public interface Injection{
      public void injectionName(String name);
}
为对象注入name属性
public class User implements Injection{
      private String name;
      public void injectName(String name){
            this.name = name;
      }
}
```

```
2.构造器注入
public class User{
      private String name;
      public User(String name){
          this.name = name;
      }
}
```

```
3.set注入
public class User{
      private String name;
      public void setName(String name){
          this.name = name;
      }
}
```

spring 支持构造器注入和 setter 方法注入

构造器注入，通过 <constructor-arg> 元素完成注入

setter 方法注入， 通过<property> 元素完成注入【开发中常用方式】

4.1.7 Spring 的核心类有哪些，各有什么作用

BeanFactory：产生一个新的实例，可以实现单例模式

BeanWrapper：提供统一的 get 及 set 方法

ApplicationContext:提供框架的实现，包括 BeanFactory 的所有功能

4.1.8 Spring 如何处理线程并发问题

Spring 使用 ThreadLocal 解决线程安全问题

我们知道在一般情况下，只有无状态的 Bean 才可以在多线程环境下共享，在 Spring 中，绝大部分 Bean 都可以声明为 singleton 作用域。就是因为 Spring 对一些 Bean(如 RequestContextHolder、TransactionSynchronizationManager、LocaleContextHolder 等)中非线程安全状态采用 ThreadLocal 进行处理，让它们也成为线程安全的状态，因为有状态的 Bean 就可以在多线程中共享了。

ThreadLocal 和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。

在同步机制中，通过对象的锁机制保证同一时间只有一个线程访问变量。这时该变量是多个线程共享的，使用同步机制要求程序慎密地分析什么时候对变量进行读写，什么时候需要锁定某个对象，什么时候释放对象锁等繁杂的问题，程序设计和编写难度相对较大。

而 ThreadLocal 则从另一个角度来解决多线程的并发访问。ThreadLocal 会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal 提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进 ThreadLocal。

由于 ThreadLocal 中可以持有任何类型的对象，低版本 JDK 所提供的 get()返回的是 Object 对象，需要强制类型转换。但 JDK5.0 通过泛型很好的解决了这个问题，在一定程度地简化 ThreadLocal 的使用。

概括起来说，对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而 ThreadLocal 采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

4.1.9 Spring 的事物管理

事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。这样可以防止出现脏数据，防止数据库数据出现问题。

开发中为了避免这种情况一般都会进行事务管理，Spring 中也有自己的事务管理机制，一般是使用 TransactionMananger 进行管理，可以通过 Spring 的注入来完成此功能。spring 提供了几个关于事务处理的类：

TransactionDefinition //事务属性定义

TranscationStatus //代表了当前的事务，可以提交，回滚。

PlatformTransactionManager 这个是 spring 提供的用于管理事务的基础接口，其下有一个实现的抽象类 AbstractPlatformTransactionManager,我们使用的事务管理类例如 DataSourceTransactionManager 等都是这个类的子类。

一般事务定义步骤：

```
1.  TransactionDefinition td = new TransactionDefinition();
2.  TransactionStatus ts = transactionManager.getTransaction(td);
3.  try{
4.          //do sth
5.          transactionManager.commit(ts);
6.  }catch(Exception e){
7.          transactionManager.rollback(ts);
8.  }
```

spring 提供的事务管理可以分为两类：编程式的和声明式的。编程式的，比较灵活，但是代码量大，存在重复的代码比较多；声明式的比编程式的更灵活。

编程式主要使用 transactionTemplate。省略了部分的提交，回滚，一系列的事务对象定义，需注入事务管理对象.

```
1.      void add(){
2.              transactionTemplate.execute( new TransactionCallback(){
3.                      pulic Object doInTransaction(TransactionStatus
ts){
4.                       //do sth
5.                      }
6.              }
7.      }
```

声明式：

使用 TransactionProxyFactoryBean:PROPAGATION_REQUIRED PROPAGATION_REQUIRED PROPAGATION_REQUIRED readOnly

围绕 Poxy 的动态代理 能够自动的提交和回滚事务

4.1.10 通知有哪些类型

前置通知（Before advice）：在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

返回后通知（After returning advice）：在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。

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。

后通知（After (finally) advice）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

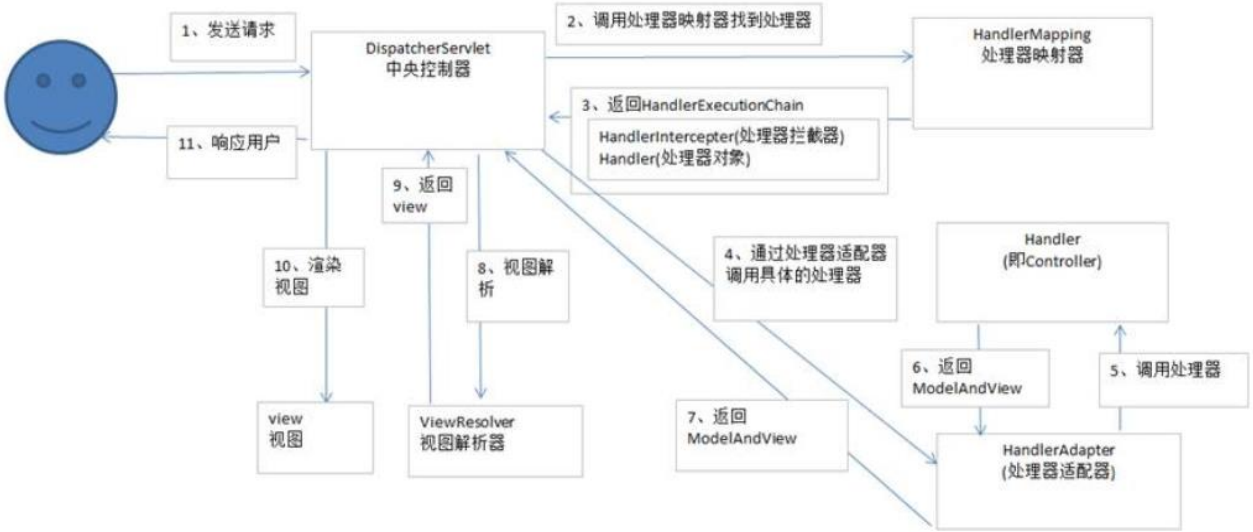
环绕通知（Around Advice）：包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。

环绕通知是最常用的一种通知类型。大部分基于拦截的 AOP 框架，例如 Nanning 和 JBoss4，都只提供环绕通知。

切入点（pointcut）和连接点（join point）匹配的概念是 AOP 的关键，这使得 AOP 不同于其它仅仅提供拦截功能的旧技术。 切入点使得定位通知（advice）可独立于 OO 层次。 例如，一个提供声明式事务管理的 around 通知可以被应用到一组横跨多个对象中的方法上（例如服务层的所有业务操作）。

4.2 SpringMVC

4.2.1 SpringMVC 的工作流程



流程

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 调用 HandlerAdapter 处理器适配器
5、HandlerAdapter 经过适配调用具体的处理器(Controller，也叫后端控制器)。
6、Controller 执行完成返回 ModelAndView
7、HandlerAdapter 将 controller 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
10、DispatcherServlet 根据 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

4.2.2 SpringMVC 的注解

1.@Controller

@Controller 用于标记在一个类上，使用它标记的类就是一个 SpringMVC Controller 对象。

2.@RequestMapping

@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。

3.@Resource 和@Autowired

@Resource 和@Autowired 都是做 bean 的注入时使用，其实@Resource 并不是 Spring 的注解，它的包是 javax.annotation.Resource，需要导入，但是 Spring 支持该注解的注入。

相同点：两者都可以写在字段和 setter 方法上。两者如果都写在字段上，那么就不需要再写 setter 方法。ssss

不同点：

@Autowired 为 Spring 提供的注解，需要导入包 org.springframework.beans.factory.annotation.Autowired；

@Autowired 注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许 null 值，可以设置它的 required 属性为 false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier 注解一起使用。如下：

```
public class TestServiceImpl {
    @Autowired
    @Qualifier("userDao")
    private UserDao userDao;
}
```

2) @Resource

@Resource 默认按照 ByName 自动注入，由 J2EE 提供，需要导入包 javax.annotation.Resource。@Resource 有两个重要的属性：name 和 type，而 Spring 将@Resource 注解的 name 属性解析为 bean 的名字，而 type 属性则解析为 bean 的类型。所以，如果使用 name 属性，则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。如果既不制定 name 也不制定 type 属性，这时将通过反射机制使用 byName 自动注入策略。

注：最好是将@Resource 放在 setter 方法上，因为这样更符合面向对象的思想，通过 set、get 去操作属性，而不是直接去操作属性。

4.@PathVariable

用于将请求 URL 中的模板变量映射到功能处理方法的参数上，即取出 uri 模板中的变量作为参数。

5.@RequestParam

@requestParam 主要用于在 SpringMVC 后台控制层获取参数，类似一种是 request.getParameter("name")，它有三个常用参数：defaultValue = "0", required = false, value = "isApp"；defaultValue 表示设置默认值，required 铜过 boolean 设置是否是必须要传入的参数，value 值表示接受的传入的参数类型。

6.@ResponseBody

作用： 该注解用于将 Controller 的方法返回的对象，通过适当的 HttpMessageConverter 转换为指定格式后，写入到 Response 对象的 body 数据区。

使用时机：返回的数据不是 html 标签的页面，而是其他某种格式的数据时（如 json、xml 等）使用；

7.@Repository

用于注解 dao 层，在 daoImpl 类上面注解。

4.2.3 如何解决 POST 请求中文乱码问题，GET 的又如何处理

在 web.xml 中加入：

```
<filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</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>
```

以上可以解决 post 请求乱码问题。对于 get 请求中文参数出现乱码解决方法有两个：

修改 tomcat 配置文件添加编码与工程编码一致，如下：

```
<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080"
protocol="HTTP/1.1" redirectPort="8443"/>
```

另外一种方法对参数进行重新编码：

```
String userName = new String(request.getParamter("userName").getBytes("ISO8859-
1"),"utf-8")
```

ISO8859-1 是 tomcat 默认编码，需要将 tomcat 编码后的内容按 utf-8 编码

4.2.4 SpringMVC 与 Struts2 的主要区别

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

②springmvc 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。

③Struts 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 对象内容进行解析成方法形参，将响应

数据和页面封装成 ModelAndView 对象，最后又将模型数据通过 request 对象传输到页面。 Jsp 视图解析器默认使用 jstl。

4.3 Mybatis

4.3.1 Mybatis 的理解

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手工设置参数以及抽取结果集。MyBatis 使用简单的 XML 或注解来配置和映射基本体，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java 对象)映射成数据库中的记录。

4.3.2 JDBC 编程有哪些不足之处，MyBatis 是如何解决这些问题的

① 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决：在 SqlMapConfig.xml 中配置数据链接池，使用连接池管理数据库链接。

② Sql 语句写在代码中造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变 java 代码。

解决：将 Sql 语句配置在 XXXXmapper.xml 文件中与 java 代码分离。

③ 向 sql 语句传参数麻烦，因为 sql 语句的 where 条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决： Mybatis 自动将 java 对象映射至 sql 语句。

④ 对结果集解析麻烦，sql 变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成 pojo 对象解析比较方便。

解决：Mybatis 自动将 sql 执行结果映射至 java 对象。

4.3.3 MyBatis 编程步骤是什么样的

① 创建 SqlSessionFactory
② 通过 SqlSessionFactory 创建 SqlSession
③ 通过 sqlsession 执行数据库操作
④ 调用 session.commit()提交事务
⑤ 调用 session.close()关闭会话

4.3.4 MyBatis 与 Hibernate 有哪些不同

Mybatis 和 hibernate 不同，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，不过 mybatis 可以通过 XML 或注解方式灵活配置要运行的 sql 语句，并将 java 对象和 sql 语句映射生成最终执行的 sql，最后将 sql 执行的结果再映射生成 java 对象。

Mybatis 学习门槛低，简单易学，程序员直接编写原生态 sql，可严格控制 sql 执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是 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。

Hibernate 对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用 hibernate 开发可以节省很多代码，提高效率。但是 Hibernate 的缺点是学习门槛高，要精通门槛更高，而且怎么设计 O/R 映射，在性能和对象模型之间如何权衡，以及怎样用好 Hibernate 需要具有很强的经验和能力才行。
总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。

4.3.5 MyBatis 的一级缓存和二级缓存

Mybatis 首先去缓存中查询结果集，如果没有则查询数据库，如果有则从缓存取出返回结果集就不走数据库。Mybatis 内部存储缓存使用一个 HashMap，key 为 hashCode+sqlId+Sql 语句。value 为从查询出来映射生成的 java 对象

Mybatis 的二级缓存即查询缓存，它的作用域是一个 mapper 的 namespace，即在同一个 namespace 中查询 sql 可以从缓存中获取数据。二级缓存是可以跨 SqlSession 的。

4.4 Struts2

4.4.1 Struts2 的处理流程

Struts2 框架的大致处理流程如下：

1、加载类（FilterDispatcher）
2、读取配置（struts 配置文件中的 Action）
3、派发请求（客户端发送请求）
4、调用 Action（FilterDispatcher 从 struts 配置文件中读取与之相对应的 Action ）

5、启用拦截器（WebWork 拦截器链自动对请求应用通用功能，如验证）
6、处理业务（回调 Action 的 execute()方法）
7、返回响应（通过 execute 方法将信息返回到 FilterDispatcher）
8、查找响应（FilterDispatcher 根据配置查找响应的是什么信息如：SUCCESS、ERROER，将跳转到哪个 jsp 页面）
9、响应用户（jsp--->客户浏览器端显示）
10、struts2 标签库（相比 struts1 的标签库，struts2 是大大加强了，对数据的操作功能很强大）

4.4.2 什么是值栈？值栈的内部结构

ValueStack 是 struts2 提供一个接口，实现类 OgnlValueStack ---- 值栈对象 （OGNL 是从值栈中获取数据的 ）每个 Action 实例都有一个 ValueStack 对象 （一个请求对应 一个 ValueStack 对象 ）在其中保存当前 Action 对象和其他相关对象 （值栈中是有 Action 引用的 ）Struts 框架把 ValueStack 对象保存在名为“struts.valueStack” 的请求属性中,request 中（值栈对象是 request 一个属性）

值栈由两部分组成，ObjectStack 和 ContextMap

ObjectStack: Struts 把动作和相关对象压入 ObjectStack 中--List
ContextMap: Struts 把各种各样的映射关系(一些 Map 类型的对象) 压入 ContextMap 中，Struts 会把下面这些映射压入 ContextMap 中
parameters: 该 Map 中包含当前请求的请求参数
request: 该 Map 中包含当前 request 对象中的所有属性
session: 该 Map 中包含当前 session 对象中的所有属性
application:该 Map 中包含当前 application 对象中的所有属性
attr: 该 Map 按如下顺序来检索某个属性: request, session, application

4.4.3 开发中，值栈主要有哪些应用

值栈主要解决 Action 向 JSP 传递数据问题

Action 向 JSP 传递数据处理结果 ，结果数据有两种形式

1）消息 String 类型数据 this.addFieldError("msg", "字段错误信息"); this.addActionError("Action 全局错误信息"); this.addActionMessage("Action 的消息信息"); * fieldError 针对某一个字段错误信息 （常用于表单校验）、actionError (普通错误信息，不针对某一个字段 登陆失败)、 actionMessage 通用消息 在 jsp 中使用 struts2 提供标签 显示消息信息

```
<s:fielderror fieldName="msg"/>
<s:actionerror/>
<s:actionmessage/>
```

2）数据 （复杂类型数据）

使用值栈 valueStack.push(products);

哪些数据默认会放入到值栈 ？？？

1）每次请求，访问 Action 对象 会被压入值栈 ------- DefaultActionInvocation 的 init 方法 stack.push(action);

* Action 如果想传递数据给 JSP，只有将数据保存到成员变量，并且提供 get 方法就可以了

2）ModelDriven 接口有一个单独拦截器

```
<interceptor name="modelDriven"
class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
```

在拦截器中，将 model 对象压入了值栈 stack.push(model);

* 如果 Action 实现 ModelDriven 接口，值栈默认栈顶对象就是 model 对象

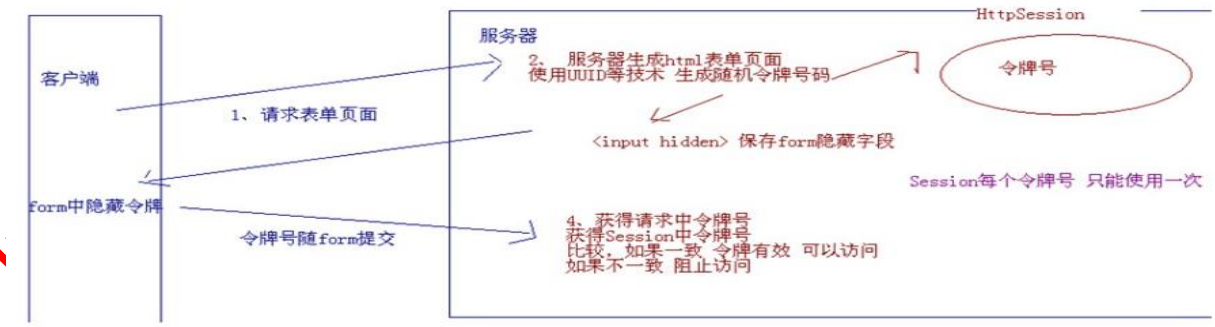
4.4.4 如何防止表单重复提交？

哪些情况会导致重复提交？

服务器处理服务后，转发页面，客户端点击刷新(重定向)

客户端网络过慢，按钮连续点击(按钮点击一次后，禁用按钮)

使用令牌机制



4.4.5 拦截器和过滤器有什么区别

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过滤器依赖与 servlet 容器，而拦截器不依赖与 servlet 容器

③拦截器只能对 action 请求起作用，而过滤器则可以对几乎所有的请求起作用

④拦截器可以访问 action 上下文、值栈里的对象，而过滤器不能

⑤在 action 的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次

拦截器 ：是在面向切面编程的就是在你的 service 或者一个方法，前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。

4.5 Hibernate

4.5.1 Hibernate 的执行流程

通过 Configuration.configure()读取并解析 hibernate.cfg.xml 配置文件；

由 hibernate.cfg.xml 中的<mapping resource=”com/xx/User.hbm.xml” />读取并解析映射信息；

通过 config.buildSerssionFactory()创建 SessionFactory;

sessionFactory.openSession()打开 session；

session.beginTransaction()创建事务 transation；

persistent operate（持久化操作）；

session.getTransaction().commit()提交事务；

关闭 session;

关闭 SessionFactory。

4.5.2 Hibernate 缓存&延迟加载

Hibernate 一级缓存又称为“Session 的缓存”。Session 内置不能被卸载，Session 的缓存是事务范围的缓存（Session 对象的生命周期通常对应一个数据库事务或者一个应用事务）。一级缓存中，持久化类的每个实例都具有唯一的 OID；

Hibernate 二级缓存又称为“SessionFactory 的缓存”。由于 SessionFactory 对象的生命周期和应用程序的整个过程对应，因此 Hibernate 二级缓存是进程范围或者集群范围的缓存，有可能出现并发问题，因此需要采用适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。第二级缓存是可选的，是一个可配置的插件，默认下 SessionFactory 不会启用这个插件。

Session 的延迟加载实现要解决两个问题：正常关闭连接、确保请求中访问的是同一个 session。Hibernate session 就是 java.sql.Connection 的一层高级封装，一个 session 对应了一个 Connection。http 请求结束后正确的关闭 session（过滤器实现了 session 的正常关闭），延迟加载必须保证是同一个 session（session 绑定在 ThreadLocal）。

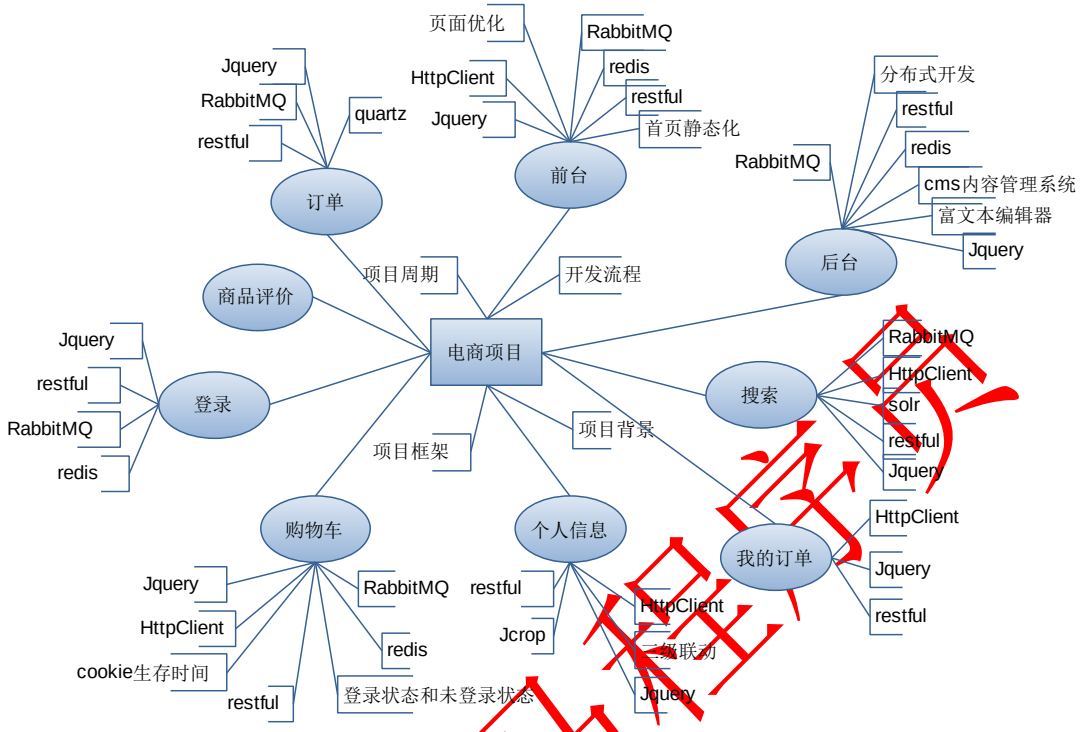
4.5.3 Hibernate 与 Mybatis 的区别

1.Mybatis 是把 sql 语句与 java 代码分离了，sql 语句在 xml 文件配置;

2.Hibernate 是 ORM 框架,它对 JDBC 进行了封装,在分层结构中处于持久化层,它能建立面向对象的域模型和关系数据模型之间的映射,它大大简化了 dao 层的编码工作;

3.Mybatis 是半自动的，Hibernate 是全自动的，就是说 Mybatis 可以配置 sql 语句,对于 sql 调优来说是比较好的,Hibernate 会自动生成所有的 sql 语句，调优不方便，Hibernate 用起来难度要大于 Mybatis。

五 电商项目



5.1.电商行业技术特点

①**技术新：**(NoSql 推广首在社区网站和电商项目)，发展快，需求推动技术的革新。

②**技术范围广：**除了 java，像淘宝前端还使用了 PHP，数据库 MySQL 或者 oracle，nosql，服务器端使用 Linux，服务器安全、系统安全

③**分布式：**以前是在一台机器上做运算，现在是分散到很多机器上，最后汇总起来。(集中式向分布式进行考虑)由需求来推动

④**高并发、集群、负载均衡、高可用：**由并发问题采用集群进行处理，其中，集群会涉及服务器的主从以及分布问题，使用负载均衡。(权重高低)高可用是对用户而言，用户的服务不中断(系统升级，服务不中断，淘宝每周更新 2 次)。

⑤**海量数据：**双 11,1207 亿的背后，订单有多少？浏览次数有多少？商品会有多少？活动相关数据？

⑥**业务复杂：**不要简单的认为是：商品展示出来后，加入购物车后购买就完成了。后台特别复杂，比如优惠(包邮、满减)

⑦**系统安全：**系统上线必须通过系统安全部门审核通过。前年 CSDN 数据泄露。快捷酒店数据泄露(通过身份证就可以查看你的开房记录)。近几年，安全意识逐步在提高。

5.2.系统功能

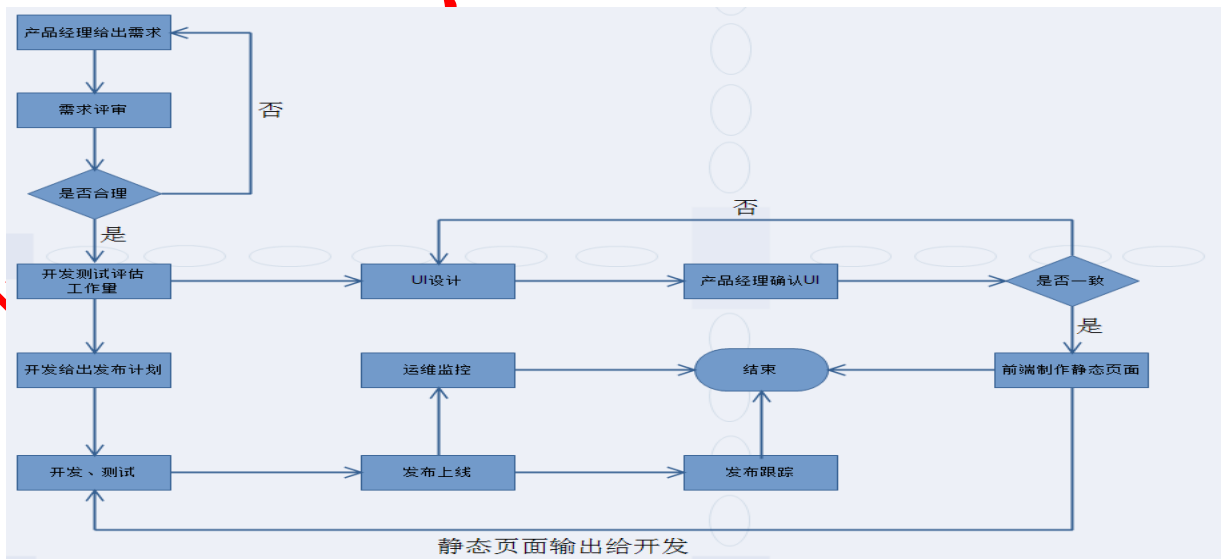
本商城系统是一个综合性的 B2C 平台.

- 会员可以在商城浏览商品、下订单，以及参加各种活动。
- 商家可以在入住淘淘商城，在该平台上开店出售自己的商品，并且得到淘淘商城提供的可靠的服务。
- 管理员、运营可以在平台后台管理系统中管理商品、订单、会员等。
- 客服可以在后台管理系统中处理用户的询问以及投诉。

5.3.本系统人员配置情况

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

5.4.开发流程



5.5.后台开发环境

需要注意，在几个环境中，预发布环境和生产环境几乎一样，开发环境和测试环境是独立存在的，每一个阶段的活是由对应的工作人员来负责的。

整个开发沿着主线进行，在一个版本定型后，前一个版本出现 bug，那么此时会对 bug 进行修复，投入使用的依旧是之前定型的那个版本，待前一个版本的 bug 修复好了之后，会定义一个新的版本，主线不会改变，如果改动不大，那么只需修订问题，继续沿用此版本(1.0.x)，只有出现较大改动时，才会升级一个新的版本号(1.1.x)。所有动作，交替进行，沿主线向前推进！

5.6.涉及技术

Spring、SpringMVC、Mybatis
JSP、JSTL、jQuery、jQuery plugin、EasyUI、KindEditor(富文本编辑器)
Lucene、Solr（搜索）
httpclient（调用系统服务）
Mysql
Redis（缓存数据库）
Nginx（web 服务器）+Tomcat
Quartz（定时任务）
RabbitMQ（消息队列）

5.7 开发工具和环境

Eclipse 4.4.1
Maven 3.2.3
Tomcat 7.0.47（Maven Tomcat Plugin）
JDK 1.7
Mysql 5.6
Nginx 1.5.1
Redis 2.8.9
Win7 操作系统
SVN（版本管理）

5.8 商城的 6 大模块

商城系统总共有：后台管理系统、前台系统、会员系统、订单系统、搜索系统、会员登录系统。

后台管理系统：管理商品、订单、类目、商品规格属性、用户管理以及内容发布等功能。

前台系统：用户可以在前台系统中进行注册、登录、浏览商品、首页、下单等操作。

会员系统：用户可以在该系统中查询已下的订单、收藏的商品、我的优惠券、团购等信息。

订单系统：提供下单、查询订单、修改订单状态、定时处理订单。

搜索系统：提供商品的搜索功能。

会员登录系统：为多个系统之间提供用户登录凭证以及查询登录用户的信息。

5.8.1 前台系统:

前台系统 是指用户在浏览网站的主页时，利用前台系统将查询到的数据显示到主页上供用户访问。

一般情况下，用户利用浏览器浏览页面，其主要的的请求流程如下：

如果请求到 nginx 后不走 tomcat，直接返回 html,那就没有后续的那么多的操作，效率就会提高很多。

这个问题的解决方案，就是页面的静态化。

那么问题来了，html 怎么生成？

这个 html 的内容肯定是来源于 tomcat,此时我们可以创建一个工程，通过 httpclient 请求到 tomcat 中的数据，然后写到 html 文件里。然后下一次 nginx 请求的时候直接获取这个 html 然后返回就可以了。

但是如果这样操作的话就会涉及到的一个数据同步的问题，如果后期数据发生了变更，那我们从静态化页面拿到的数据还是旧的。

此时的解决方案有两种：

1、实时更新　　同步更新数据

2、定时更新　　定时每隔多久更新一次

怎么选择？在选择的时候一般都要结合业务，而我们现在做的是首页，首页的话其实用定时更新就可以了。

怎么实现定时更新呢？可以利用各种定时任务，比如说 quartz 定时任务就能达到定时更新的效果了。

实际上还有一个更好的解决方案，工程在最后都是需要部署到 linux 上的，在 linux 中有一个 wget（下载资源）命令，可以用 wget 请求 tomcat

拿到文件生成 html，用 wget 替代了 httpclient 的作用，而且在 linux 中有一个定时命令 crontab，我们可以用这个 crontab 命令就可以起到定时更新的效果，那这个方案根本就不用新建一个工程就能达到目的了。

做完页面静态化后其实还可以做进一步的优化，比如我们第一次请求主页的时候还没有这个 html 静态文件，是不是还要走 tomcat 请求数据？而且并不是所有的请求路径都需要走这个步骤吧，那我们这个时候就要做一个判断，判断是不是有 html,如果这个文件不存在，就走 tomcat，如果存在就直接返回这个静态 html，那么就需要我们在 nginx 中做一个配置，配置一个判断条件，因为我们请求的是首页，因此我们可以判断请求是/做结尾或者以 index.html 做结尾接可以了（具体配置参加项目实战时的文档）。

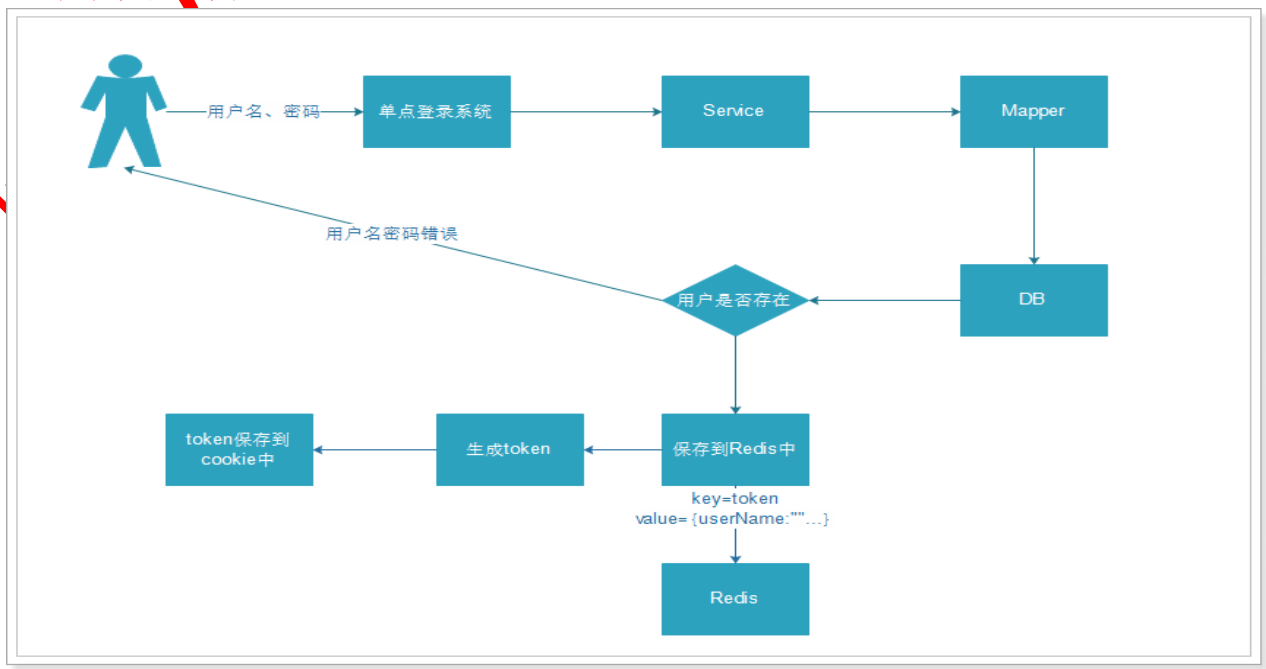
页面优化：

其实在做完页面静态化后还能够做一些加载资源的优化，我们都知道，浏览器在请求同一个域名时是有并发连接数的限制的，比如当页面在加载 js/css/images/flash 等资源时就会出现这种问题，因此，为了突破这些并发限制，我们可以通过设置不同的域名（最后访问的还是同一个资源）来达到这样的效果，其实像淘宝和京东这样的网站都做了这样的优化。

5.8.2 登录系统：

当会员需要登录的时候，首先进入的是“登录页面”，输入用户名、密码、验证码进行登录，进入单点登录系统（Controller） → Service → Mapper → DB → 提交登录页面，用户信息被提交到单点登录系统进行校验。如果登陆信息不正确，校验错误，将错误的登陆信息进行回显返回给用户，重新登录。验证通过，就将用户信息保存到 redis 中，并且生成 token 作为一个标识，并将 token 保存到 cookie 里面，发送给用户浏览器，下次请求的时候就会带回来，获取 cookie 里面的 token，根据 token 再去 redis 中查询对应的用户信息。登录成功后将重定向到登录之前的访问页面。

登录系统流程图:



每一个系统的访问域名不一样，默认的情况下，一个服务中产生的 cookie 只能当前服务使用，浏览器不会讲该 cookie 发送到其他服务。遇到这样的问题，在 cookie 中有一个属性 domain（有效域名）默认为当前的域名，我们需要将有效域名设置成“.taotao.com”结尾，以后浏览器就会把所有“.taotao.com”结尾的域名都会发送 cookie 信息。

登录成功后，发现 js、css、image 没有加载，也就涉及到了静态资源加载的问题，此处使用了 nginx 访问静态资源。具体实现为：

1.使用新域名访问静态资源如“xxx.taotao.com”，避免携带一些无用的 cookie

2.拷贝 JS 和 CSS 到磁盘路径中

3.配置 nginx

```
server {
    listen          80;
    server_name    static.taotao.com;
    #charset koi8-r;
    #access_log    logs/host.access.log  main;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    location / {
root    E:\\taotao\\taotao-static;#直接去本地查找静态资源，并返回
    }}
```

对于多系统共存的环境下，**通过 Ajax 跨域判断用户是否登录**。

5.8.3 搜索系统：

一般在各种电商网站主页都会有一个搜索的输入框，可以用来搜索各种商品。我们是单独做了一个搜索管理的模块（包括主页商品搜索和个人订单搜索），通过后台将查询到的商品数据添加到了 solr 库中，然后提供一个对外的接口，当用户在搜索比如 apple 时，会将所有 applie 的商品从 solr 库中搜索并返回给前台系统显示到页面上。Solr 是一种基于 Lucene 的全文搜索服务器。同时对其进行了扩展，提供了比 Lucene 更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎 。

而在从 solr 库中查询到的数据显示到页面上时会潜藏这一个问题，更新同步问题，一旦后台更新了商品数据，那么此时利用 solr 查询到的商品还是未更新过的旧数据，此时就要做一个更新 solr 库的逻辑，有两种方案：

方案一，可以写一个更新 solr 库的对外接口，当后台更新数据时调用这个接口进行更新即可。

方案二，可以利用 rabbitMQ 技术，当后台更新数据时发送相关的消息，如商品 id，然后在搜索系统来监听这个消息，拿到商品 id 再进行 solr 库的更新操作。

这两个方案都可以达到同步的效果，不过显然第二种更加合适一点，可以解耦合操作。

而当查询出一系列的商品之后，用户可以通过点击其中某一个商品跳转到商品详情页来查看该商品的详细信息，而为了提高用户的访问效率，我们对这些商品数据又做了一个缓存，我们用的缓存技术是 redis,当用户第一次访问该商品后会将该商品信息缓存到 redis 中，当下一次访问时会从 redis 中获取商品数据并返回。但是一旦添加了缓存就会涉及到商品更新同步的一个问题，当我们在后台修改某个商品的信息，比如修改价格后，如果我们是从缓存中获取的商品数据的话，此时的数据是还没有更新过的旧数据，因此为了同步数据，需要在添加缓存的模块中再写一个删除缓存的接口，当后台某个数据一旦更新后，就利用 httpclient 来调用这个接口并删除缓存，以便用户在下次查询商品时能直接从数据库查到最新的商品数据，然后再对这个商品做一个缓存。但是后来考虑到利用这样做的话还是考虑到耦合性太高，所有后来更换了方案，用 RabbitMQ 的消息传递来进行解耦。当我们在后台进行了商品数据的更新后，就利用 MQ 发送一个消息出去，比如说消息内容是商品 id,然后

在缓存模块中监听该消息，拿到商品 id 后到缓存中删除该商品数据就行了。

5.8.4 商品管理系统:

商城首页首先看到的是商品类目，商品类目做了商品类目的回显，和对商品类目跨域问题的解决。

那么什么是商品类目的回显呢，就是在首页的左侧有一个商品分类，当鼠标放上去的时候会显示各类商品的详细分类。做这个功能涉及到了跨域的问题，JS 有跨域限制，这是一种浏览器的限制，浏览器规定，在不同的域之间不能通过 JS 的 AJAX 请求获取数据。端口和域名都相同就不是跨域，除此之外都算跨域。对于这个问题，我们是通过 AJAX 的 jsonp 来解决这个跨域问题的，而其实 jsonp 的实现原理就是利用了 script 的 src 来获取其他域的数据的（script 的 src 是可以跨域请求数据的）。

首页除了商品类目，还有很多其他数据，如大广告、小广告、品牌推送、快报等等。有的有价格，有的有标题，还有大小不一的图片。如果给每一个类型的数据都做一张表，那么表就会太多，实际上是通过 cms 内容管理系统来进行统一管理的。这是一个后台管理前台显示内容的内容管理系统，为了管理方便，我们不用管主页显示的内容是什么，格式是不是一致，我们只是将这些数据都当作是一个内容来进行存储，这样就方便做统一的管理和开发。

为减轻数据库和服务器的压力对商品的详情页做静态化处理。我们将单品页的数据及用户的登录信息存储到 redis 缓存服务器中，其中涉及到了保证 redis 服务器中的数据与数据库数据同步问题的处理，静态页面处理动态数据（价格等），由于访问量的过大以及高并发的存在，redis 受物理内存的限制，后期使用 nginx 搭建 redis 集群，达到项目的完善运行。

5.8.5 购物车系统:

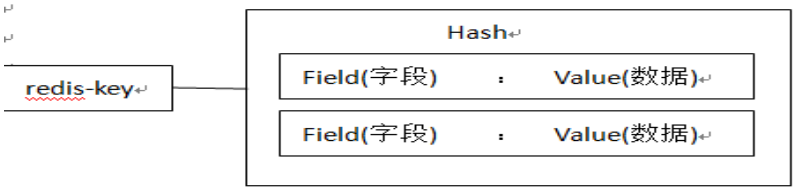
购物车功能分两种状态，未登录状态购物车和已登录状态购物车。

未登录状态购物车:

利用 cookie 中的 value（value 可以是一个 uuid 来保证唯一性）再加上一个前缀比如 REDIS_CART 做为 redis 的 key,然后将购物车数据转成 json 存入到 redis 中。这样就达到了京东的未登录状态下加入商品到购物车的类似功能。

但是这样存数据的话，如果购物车中有很多个商品，而之后要修改购物车中某个商品的数量，我们就需要将所有的商品取出来遍历、判断、修改然后再重新转换成 json 再存入到 redis 中，这样就有点麻烦。因此就采用了 redis 中的 hash 结构的存储方式。

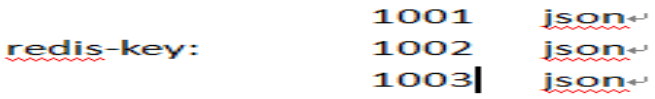
它的结构方式是：



那么我们怎么存储比较好呢？

redis-key 保持不变，然后可以将 itemId 作为字段，然后将数据存入到 value 中。

如：



如果需要修改商品数量的话直接可以通过 itemId 找到对应的商品修改而不需要全部取出了。

在购物车页面我们可以通过勾选前面的复选框来选择需要的商品进行下单，怎么实现呢，如果复选框被勾选了，在提交页面的时候将对应的商品 id 一起提交，如果全选，那就不用提交商品 id，那么在提交订单时我们只要进行判断，如果传递了商品 id，那么就指定商品下单，如果没有提交商品 id,那么就通过用户 id（在提交订单时用户肯定已经登录了）查询其下的所有订单进行全部下单。

登录状态购物车:

当我们在未登录状态下下单的时候，会跳转到登录页面，让用户先登录，这时我们就要考虑将未登录状态下的购物车数据在登录后合并到一起。

那么登录之后的合并数据的逻辑应该写在哪儿呢，由于登录功能是单独的一个工程，因此我利用了 MQ 的技术，在用户登录成功之后，发送消息，然后在购物车系统中监听到消息之后再处理数据合并。而消息内容一个是 userId，用来确定合并之后的数据保存在哪个用户的购物车中，另一个就是 cookie 中的 uuid?通过 uuid 查询到该用户在未登录状态下的购物车数据（之前购物车的优化，已经把未登录状态下的购物车数据存储到了 redis 中，而 redis 的 key 就是这个 uuid,如果用户登录之后携带的 cookie 中的 uuid 和 redis 的 key 一致，那是不是就可以合并了），然后进行数据的合并，那这个功能也就可以完成了。

需要注意的是，我们合并数据之后是不是还要把 redis 中的购物车数据删除掉，如果不删除的话，下次登录是不是又要合并了。

当我们下单之后，应该要将购物车里的数据删除掉，如果不删除的话下一下单购物车里的数据就又会增加上去了。

下单流程：从前台工程-taotaoweb 请求到订单工程-taotao-order,然后将数据保存的数据库中。

taotao-web--------taotao-order--------DB 数据库

那我们考虑一下，删除购物车逻辑应该写到哪儿呢，taotao-web 还是 taotao-order 呢？

是不是写在 taotao-order 中比较合理呢，没错，可以在 taotao-order 中通过调用 taotao-cart 的接口来实现删除购物车逻辑。但是有没有什么问题？是不是耦合度比较高啊，那怎么解决呢？就是利用 MQ，这里也是一样，在订单生成后，我们可以发送一个消息出去，然后通知 taotao-cart 工程，taotao-cart 通过监听这个消息将用户的购物车数据删除。那么发送的消息内容是什么，什么是必须的，userId 是不是必须的，还有 itemId,是不是有这两个就可以了。其他所有数据是不是都可以通过这两个 id 来查询出来。

这里有个问题，我们现在删除的是登录状态下的购物车，那么未登录状态下的购物车是不是要删除？不用吧，我们在登录的时候是不是已经将未登录状态中的购物车数据合并了啊，一旦合并之后，就会将未登录状态的购物车数据删除。

5.8.6 订单系统:

购物车中去点击去结算，就会跳转到提交订单页面，点击提交订单生成一条订单。在生成订单时，考虑到订单 ID 不能重复、订单 ID 尽可能段使用方便占用内存小、订单 ID 要求全是数字，所以我们的订单 ID 采用“用户 ID+当前系统的时间戳”作为订单的 ID。

还涉及到了对订单状态修改的操作，订单页面搜索订单的功能使用到了 solr 技术。基于订单系统接口完成下单功能，使用拦截器实现用户是否登录的校验，接受提交订单请求使用 token 进行两次查询。提交订单和遇到高并发时，要进行判断订单的有效性和数据库数据是否同步的问题，这里采用了定时器，定时与数据库数据进行交互。

六 电商项目中的技术点

6.1 Nginx（web 服务器）

6.1.1 Nginx 反向代理为什么可以提高网站性能？

对于后端是动态服务来说，比如 Java 和 PHP。这类服务器（如 JBoss 和 PHP-FPM）的 IO 处理能力往往不高。

Nginx 有个好处是它会把 Request 在读取完整之前 buffer 住，这样交给后端的就是一个完整的 HTTP 请求，从而提高后端的效率，而不是断断续续的传递（互联网上连接速度一般比较慢）。

同样，Nginx 也可以把 response 给 buffer 住，同样也是减轻后端的压力。

6.1.2 Nginx 和 Apache 各有什么优缺点？

nginx 相对 apache 的优点：

- 轻量级，同样起 web 服务，比 apache 占用更少的内存及资源

- 抗并发，nginx 处理请求是异步非阻塞的，而 apache 则是阻塞型的，在高并发下 nginx 能保持低资源低消耗高性能
- 高度模块化的设计，编写模块相对简单
- 社区活跃，各种高性能模块出品迅速啊

apache 相对 nginx 的优点：

- rewrite ，比 nginx 的 rewrite 强大
- 模块超多，基本想到的都可以找到
- 少 bug ，nginx 的 bug 相对较多
- 超稳定

存在就是理由，一般来说，需要性能的 web 服务，用 nginx 。如果不需要性能只求稳定，那就 apache 吧。

6.1.3 Nginx 多进程模型是如何实现高并发的？

进程数与并发数不存在很直接的关系。这取决取 server 采用的工作方式。

如果一个 server 采用一个进程负责一个 request 的方式，那么进程数就是并发数。那么显而易见的，就是会有很多进程在等待中。等什么？最多的应该是等待网络传输。其缺点题主应该也感觉到了，此处不述。

而 nginx 的异步非阻塞工作方式正是利用了这点等待的时间。在需要等待的时候，这些进程就空闲出来待命了。因此表现为少数几个进程就解决了大量的并发问题。

apache 是如何利用的呢，简单来说：同样的 4 个进程，如果采用一个进程负责一个 request 的方式，那么，同时进来 4 个 request 之后，每个进程就负责其中一个，直至会话关闭。

期间，如果有第 5 个 request 进来了。就无法及时反应了，因为 4 个进程都没干完活呢，因此，一般有个调度进程，每当新进来了一个 request，就新开个进程来处理。

nginx 不这样，每进来一个 request，会有一个 worker 进程去处理。但不是全程的处理，处理到什么程度呢？处理到可能发生阻塞的地方，比如向上游（后端）服务器转发 request，并等待请求返回。

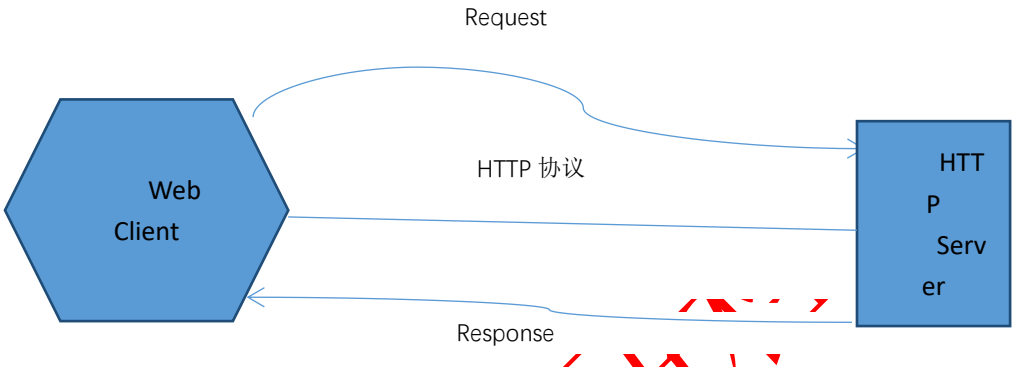
那么，这个处理的 worker 不会这么傻等着，他会在发送完请求后，注册一个事件：“如果 upstream 返回了，告诉我一声，我再接着干”。于是他就休息去了。此时，如果再有 request 进来，他就可以很快再按这种方式处理。而一旦上游服务器返回了，就会触发这个事件，worker 才会来接手，这个 request 才会接着往下走。由于 web server 的工作性质决定了每个 request 的大部份生命都是在网络传输中，实际上花费在 server 机器上的时间片不多。这是几个进程就解决高并发的秘密所在。

webserver 刚好属于网络 io 密集型应用，不算是计算密集型。异步，非阻塞，使用 epoll，和大量细节处的优化。也正是 nginx 之所以然的技术基石。

6.1.4 tomcat 与 nginx，apache 的区别是什么？

题主说的 Apache，指的应该是 Apache 软件基金会下的一个项目——Apache HTTP Server Project；Nginx 同样也是一款开源的 HTTP 服务器软件（当然它也可以作为邮件代理服务器、通用的 TCP 代理服务器）。

HTTP 服务器本质上也是一种应用程序——它通常运行在服务器之上，绑定服务器的 IP 地址并监听某一个 tcp 端口来接收并处理 HTTP 请求，这样客户端（一般来说是 IE，Firefox，Chrome 这样的浏览器）就能够通过 HTTP 协议来获取服务器上的网页（HTML 格式）、文档（PDF 格式）、音频（MP4 格式）、视频（MOV 格式）等等资源。下图描述的就是这一过程：



不仅仅是 Apache HTTP Server 和 Nginx，绝大多数编程语言所包含的类库中也都实现了简单的 HTTP 服务器方便开发者使用：

HttpServer (Java HTTP Server)

Python SimpleHTTPServer

使用这些类库能够非常容易的运行一个 HTTP 服务器，它们都能够通过绑定 IP 地址并监听 tcp 端口来提供 HTTP 服务。

Apache Tomcat 则是 Apache 基金会下的另外一个项目，与 Apache HTTP Server 相比，Tomcat 能够动态的生成资源并返回到客户端。Apache HTTP Server 和 Nginx 都能够将某一个文本文件的内容通过 HTTP 协议返回到客户端，但是这个文本文件的内容是固定的——也就是说无论何时、任何人访问它得到的内容都是完全相同的，这样的资源我们称之为静态资源。动态资源则与之相反，在不同的时间、不同的客户端访问得到的内容是不同的，

例如：

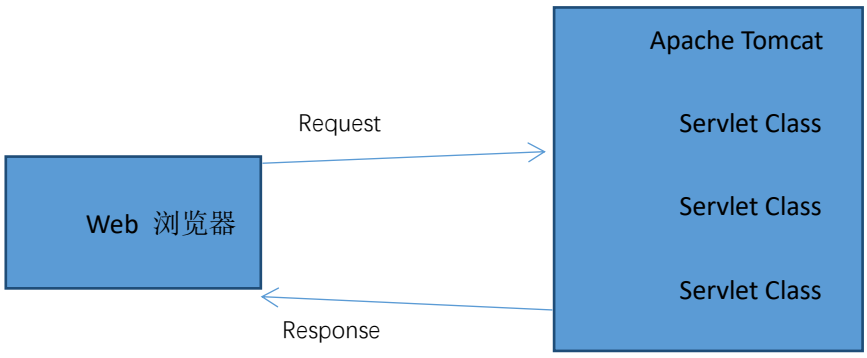
包含显示当前时间的页面

显示当前 IP 地址的页面

Apache HTTP Server 和 Nginx 本身不支持生成动态页面，但它们可以通过其他模块来支持（例如通过 Shell、PHP、Python 脚本程序来动态生成内容）。

如果想要使用 Java 程序来动态生成资源内容，使用这一类 HTTP 服务器很难做到。Java Servlet 技术以及衍生的 Java Server Pages 技术可以让 Java 程序也具有处理 HTTP 请求并且返回内容（由程序动态控制）的能力，Tomcat

正是支持运行 Servlet/JSP 应用程序的容器（Container）：



Tomcat 运行在 JVM 之上，它和 HTTP 服务器一样，绑定 IP 地址并监听 TCP 端口，同时还包含以下指责：

管理 Servlet 程序的生命周期

将 URL 映射到指定的 Servlet 进行处理

与 Servlet 程序合作处理 HTTP 请求——根据 HTTP 请求生成 HttpServletResponse 对象并传递给 Servlet 进行处理，将 Servlet 中的 HttpServletResponse 对象生成的内容返回给浏览器

虽然 Tomcat 也可以认为是 HTTP 服务器，但通常它仍然会**和 Nginx 配合在一起使用：**

动静态资源分离——运用 Nginx 的反向代理功能分发请求：

所有动态资源的请求交给 Tomcat，而静态资源的请求（例如图片、视频、CSS、JavaScript 文件等）则直接由 Nginx 返回到浏览器，这样能大大减轻 Tomcat 的压力。

负载均衡，当业务压力增大时，可能一个 Tomcat 的实例不足以处理，那么这时可以启动多个 Tomcat 实例进行水平扩展，而 Nginx 的负载均衡功能可以把请求通过算法分发到各个不同的实例进行处理

6.2 Httpclient

6.2.1 Httpclient 是什么

HttpClient不是一个浏览器。它是一个客户端的HTTP通信实现库。HttpClient的目标是发送和接收HTTP报文。HttpClient不会去缓存内容，执行嵌入在HTML页面中的javascript代码，猜测内容类型，重新格式化请求/重定向URI，或者其它和HTTP运输无关的功能.它主要就是支持HTTP传输协议的.

6.2.2 Httpclient 的使用

我们知道，HTTP协议的连接方法有GET、POST、PUT和HEAD方式，在创建Method实例的时候可以更具具体的方法来创建。HttpClient的使用一般分如下几步：

1、创建HttpClient实例。

2、创建具体连接方法的实例。如POST方法创建PostMethod的实例，在实例化时从构造函数中传入待连接的URL地址。

3、对post的发送内容等信息进行配置

4、执行HttpClient的execute方法

5、如果返回的状态码正常，表明连接成功，可以读取response的内容

6.3 JSONP

6.3.1jsonp 到底是什么？

JSONP 的原理非常简单，为了克服跨域问题，利用没有跨域限制的 script 标签加载预设的 callback 将内容传递给 js。一般来说我们约定通过一个参数来告诉服务器 JSONP 返回时应该调用的回调函数名，然后拼接出对应的 js。

6.4 Redis（缓存数据库）

6.4.1Redis 是什么？

通常而言目前的数据库分类有几种，包括 SQL/NOSQL，关系数据库，键值数据库等等，分类的标准也不以，Redis 本质上也是一种键值数据库的，但它在保持键值数据库简单快捷特点的同时，又吸收了部分关系数据库的优点。从而使它的位置处于关系数据库和键值数 据库之间。Redis 不仅能保存 Strings 类型的数据，还能保存 Lists 类型（有序）和 Sets 类型（无序）的数据，而且还能完成排序（SORT） 等高级功能，在实现 INCR，SETNX 等功能的时候，保证了其操作的原子性，除此以外，还支持主从复制等功能。

6.4.2Redis 用来做什么？

通常局限点来说，Redis 也以消息队列的形式存在，作为内嵌的 List 存在，满足实时的高并发需求。而通常在一个电商类型的数据处理过程之中，

有关商品，热销，推荐排序的队列，通常存放在 Redis 之中，期间也包扩 Storm 对于 Redis 列表的读取和更新。

6.4.3Redis 的优点？

3.1.性能极高 – Redis 能支持超过 100K+ 每秒的读写频率。

3.2.丰富的数据类型 – Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

3.3.原子 – Redis 的所有操作都是原子性的，同时 Redis 还支持对几个操作全并后的原子性执行。

3.4.丰富的特性 – Redis 还支持 publish/subscribe, 通知, key 过期等等特性。

6.4.4Redis 的缺点？

是数据库容量受到物理内存的限制,不能用作海量数据的高性能读写,因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

6.4.5Redis 持久化

RDB 持久化：

该机制是指在制定的时间间隔内将内存中的数据集快照写入磁盘。

优点 1.只有一份 rdb 文件，可随时备份

2.比 AOF 文件小，加载效率高

3.只提供 fork 子进程，不阻塞主进程，IO 操作比较少

AOF 持久化：

该机制将以日志的形式记录服务器所处理的每一个写操作，在 Redis 服务器启动之初会读取该文件来重新构建数据库，以保证启动后数据库中的数据是完整的。

优点：1.每次改动同步数据安全性好

2.APPEND 方式追加日志，不会对旧日志文件产生影响

无持久化：

我们可以通过配置的方式禁用 Redis 服务器的持久化功能，这样我们就可以将 Redis 视为一个功能加强版的 memcached 了

6.4.6Redis 集群

群指的是将几台服务器集中在一起，实现同一业务

1. 目的：高可用、负载均衡、易扩展、数据安全、性能提升
2. 技术：集群地址（虚拟 IP）、网络通信（监控消息）
3. 功能：负载均衡、读写分离、故障转移

6.5 Quartz

6.5.1Quartz 简介

Quartz 作为一个优秀的开源调度框架，
Quartz 具有以下特点：
强大的调度功能，
支持立即调度、定时调度、周期调度、并发调度； **灵活的应用方式，**
支持 job 间通过 listener 实现依赖调度，可以方便的进行调度组合

6.5.2 用 Quartz 做定时任务调度

需求是这样的，以整点时间戳为文件名，每隔一小时创建一个文件，在这一小时内不断的写文件，达到下一小时关闭当前文件句柄和流，并以当前整点小时创建新文件!

写一个单例服务类，服务类两个成员变量，一个是文件句柄，一个是流，可 get 可 set。把它作为 spring 的一个 bean，在 quartz 和你的读写线程都注入这个 bean。quartz 里面用 set 来改句柄(百度上有很好的解释)和流，读写线程用 get 来读

6.5.3 如何监控 Quartz 的 job 执行状态：运行中，暂停中，等待中？

目前我能想到的解决办法是通过往表（新建一个操作日志表）里插入日志的形式：

运行中：通过 JobListener 监听器来实现运行时更改表信息
暂停中：调用 scheduler.pauseTrigger()方法时，更改表中 job 信息
等待中：新添加的 job 默认给其等待中的状态，也是更改表中的 job 信息
但是上面这种形式的麻烦之处是得频繁的往表里插入数据。

6.6 MQ

6.6.1RabbitMQ：

6.6.1.1 什么是 RabbitMQ？

RabbitMQ 是实现 **AMQP**（高级消息队列协议）的消息中间件的一种，最初起源于金融系统，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。消息中间件主要用于组件之间的解耦，消息的发送者无需知道消息使用者的存在，反之亦然.

RabbitMQ 本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正是如此，使的它变的非常重量级，更适合于企业级的开发。同时实现了一个经纪人(Broker)构架，这意味着消息在发送给客户端时先在中心队列排队。对路由(Routing)，负载均衡(Load balance)或者数据持久化都有很好的支持。

6.6.1.2 RabbitMQ 的优点（适用范围）

1.2.1 基于 erlang 语言开发具有高可用高并发的优点，适合集群服务器。
1.2.2 健壮、稳定、易用、跨平台、支持多种语言、文档齐全。
1.2.3 有消息确认机制和持久化机制，可靠性高。
1.2.4 开源

其他 MQ 的优势：

1.2.1 Apache ActiveMQ 曝光率最高，但是可能会丢消息。
1.2.2. ZeroMQ 延迟很低、支持灵活拓扑，但是不支持消息持久化和崩溃恢复。

6.6.2AcitveMQ

6.6.2.1 AcitveMQ 的作用、原理？（生产者。消费者。 p2p、订阅实现流程）

Activemq 的作用就是系统之间进行通信。当然可以使用其他方式进行系统间通信，如果使用 Activemq 的话可以对系统之间的调用进行解耦，实现系统间的异步通信。原理就是生产者生产消息，把消息发送给 activemq。Activemq 接

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

6.6.2.2 ActiveMQ 在项目中应用场景？

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。

6.6.2.3 ActiveMQ 如果数据提交不成功怎么办？

Activemq 有两种通信方式，点到点形式和发布订阅模式。如果是点到点模式的话，如果消息发送不成功此消息默认会保存到 activemq 服务端知道有消费者将其消费，所以此时消息是不会丢失的。

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

6.7 Dubbo

6.7.1、dubbo 服务开发流程，运行流程？zookeeper 注册中心的作用？

使用流程：

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

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

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

Zookeeper 注册中心的作用主要就是注册和发现服务的作用。类似于房产中介的作用，在系统中并不参与服务的调用及数据的传输。

6.8 Solr

6.8.1solr 的原理

用通俗的语言描述下 Apache_Solr 是做什么用的？

solr 是基于 lucene 搜索库的一个搜索引擎框架。

1、简单来说所谓的索引是为了全文数据存储和查询准备的，全文数据即非结构化数据，比如一篇文章，如果你要用一般的手段进行搜索的话，比如用数据库的 like 命令会有不少限制，而且还不能支持一些相同语义的问题。

举个简单的例子，你在亚马逊上面搜索 solr，却可以查询出来 lucene 内容，这就是关联性查询和语义查询。

2、高性能的，上亿条数据通过索引的方式可以秒级查询出来，优化后可能更好。

3、solr 将非结构化数据，通过分词、词法分析、过滤停词、语义分析等手段来转成结构化数据，存储为索引，里面有文档概念这个和 mysql 的一条记录又类似。

6.8.2solr 怎么设置搜索结果排名靠前（得分）？

可以设置文档中域的 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。此方法可以把热点商品或者是推广商品的排名提高。

6.8.3solr 里面 IK 分词器的原理

IK 分析器的分词原理本质上是词典分词。现在内存中初始化一个词典，然后在分词过程中逐个读取字符，和字典中的字符相匹配，把文档中的所有的词语拆分出来的过程。

七 电商项目面试问题

7.1 简单的介绍一下你这个项目

1、 xxx 商城项目打造的是“社区+电商”的模式，用户不只是在社区中有自己的圈子，还可以将电商加入到社区中，整个电商网站实现的功能非常之多，采用分布式的架构设计，包括后台管理、前台系统、订单系统、单点登录系统、搜索系统、会员系统等。

①该项目是自己公司的产品，我们公司负责整个网站的运营，属于 B2C 平台；

②系统的用途，主要是提供 B2C 的平台，其中自营商品也有商家入住，类似天猫。

③系统架构，采用分布式的系统架构，其中前台系统和单点登录系统采用了集群的方式部署，在后台管理系统中采用了 Maven 的多模块化的管理，其中采用了水平切分的方式，将 pojo、dao、service、web 分层开发，这样做的好处就是可以重用性更高。

2、 系统内部接口调用采用 Httpclient，并且使用 Httpclient 的连接池技术，接口提供端采用 RESTful 方式的接口定义；

3、 系统之间的通知机制采用 MQ 的方式，使用 RabbitMQ 的实现，使用了 RabbitMQ 的消息订阅模式的消息机制；

4、 系统的接口还对 JS 的跨域做了支持，采用了 jsonp 的解决方法，在后台接口中扩展了 spirng 提供的 jackson 数据转化器实现；

5、 ④部署方面，采用了 Nginx+tomcat 的模式，其中 nginx 的作用一方面是做反向代理、负载均衡、另一方面是做图片等静态资源的服务器。

6、 项目介绍应该从这个项目的模式、功能、架构、解决了什么问题、部署以及投入使用情况来说。

7.2 整个项目的架构如何？

一般情况这个问题要从两个方面去回答，从技术架构和功能架构去谈。

技术架构：本项目使用市场上较为主流的框架 spring+springmvc+mybatis 进行开发，采用分布式的系统架构，前台系统和单点登录系统采用了集群的方式部署，后台管理系统中采用了 Maven 的多模块化的管理，其中采用了水平切分的方式，将 pojo、dao、service、web 分层开发，这样做的好处就是可以重

用性更高。系统内部接口调用采用 Dubbo，在后台接口中扩展了 spirng 提供的 jackson 数据转化器实现；搜索系统使用了 solr 实现，在保证系统高性能的前提下，尽可能为公司节约成本，我们使用 MySQL 数据库进行集群(oracle 收费)。在部署方面，采用了 Nginx+tomcat 的模式，其中 nginx 的作用一方面是做反向代理、负载均衡、另一方面是做图片等静态资源的服务器。

功能架构：分布式系统架构

各个系统说明：

- 后台管理系统：管理商品、订单、类目、商品规格属性、用户管理以及内容发布等功能。
- 前台系统：用户可以在前台系统中进行注册、登录、浏览商品、首页、下单等操作。
- 会员系统：用户可以在该系统中查询已下的订单、收藏的商品、我的优惠券、团购等信息。
- 订单系统：提供下单、查询订单、修改订单状态、定时处理订单。
- 搜索系统：提供商品的搜索功能。
- 会员登录系统：为多个系统之间提供用户登录凭证以及查询登录用户的信息。

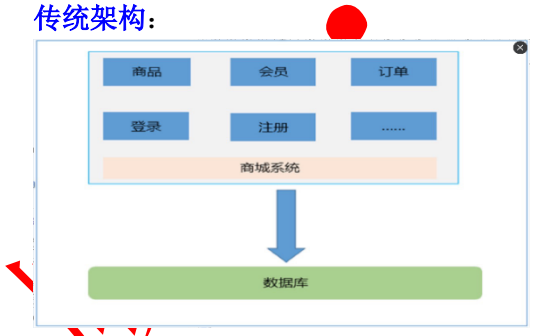
谈到分布式架构，我们必须对比传统架构才能彰显其优势。

①最为明显的一点，在传统的架构中，如果某个功能需要进行维护，那么我们必须停掉整个服务，这对于公司的运营会造成损失。分布式系统在核心功能模块使用单独服务器，维护部分模块不影响用户的其他操作。

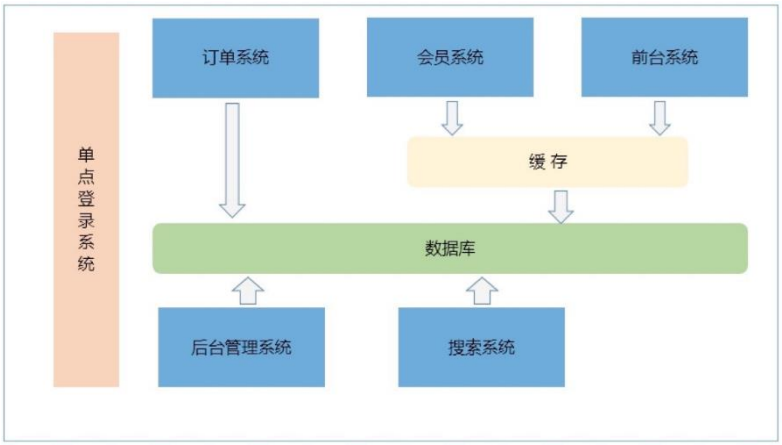
②在海量数据处理方面，传统架构显得比较乏力；分布式系统架构采用服务器集群，使用负载均衡，海量数据处理游刃有余！

③在性能(检索)以及维护方面，分布式系统架构也有较为明显的优势。

传统架构：



分布式架构：



7.3 这个项目为用户提供了哪些服务？包括哪些功能？

商品管理模块：其中包括品牌管理，属性管理商品录入/上下架管理，商品添加审核，静态页面发布

订单模块：其中包括使用 activiti 工作流订单的查询和订单的流转,定时作废

商品前台首页：其中主要负责首页商品列表筛选，首页上动态展示筛选条件，点击每一个筛选条件下面的商品列表要做联动

单品页面：采用 freemarker 来实现页面静态化，展示商品详情信息和商品购买，该页面采用静态化以减轻系统压力，使用了 cxf 框架发布服务

提交订单页面：提交用户的订单信息，处理并发问题。

个人中心，包括用户的登录，个人信息的管理，收货地址的管理，用户所下的订单的管理

购物车：把购物车的信息存在 cookie 里面管理

7.4 你承担这个项目的哪些核心模块？

在项目中主要负责相关系统的开发，主要有：

1) 后台管理系统，主要实现商品管理、商品规格参数管理、订单管理、会员管理等、CMS(内容管理系统)等，并且提供了跨域支持；

2) 前台系统，主要是面向用户访问，使用 Httpclient 和后台系统接口做交互，并且该系统在部署上采用集群的方式；

3) 会员登录系统，主要是提供集中用户登录凭证的集中解决方案，提供和用户信息相关的接口，比如说用户注册、查询等接口。

4) 订单系统，主要是提供和订单相关的业务接口，在订单系统了做了严格的数据校验以及高并发写的支持（这里可以说使用队列实现），并且使用了 Quartz 定时任务实现对订单的定时扫描，比如说关闭超时未付款的订单；

5) 搜索系统，主要是提供商品的搜索，采用开源企业级系统 Solr 实现，采用了 MQ 机制保证了商品数据可以及时同步到 solr 中；

6) 会员系统，主要是维护用户的信息，已购买订单、优惠券、系统消息、修改密码、绑定手机等功能；

7) 缓存，主要是用 Redis 实现，并且对 Redis 做了集群来保证 Redis 服务的高可用。

8) 支付系统，主要是负责订单的支付、对账等功能，主要是对接了支付宝的接口；

7.5 这些模块的实现思路说一下？

①商品管理模块

品牌管理：
主要掌握的核心是品牌的添加
图片服务器的搭建，
上传图片到图片服务器
表单的验证，离开焦点的验证，点击完成时的验证，后台服务器的 ajax 验证，表单规范
防止表单的二次提交
编辑时不需要修改品牌名，区别 readOnly 和 disabled
删除时的二次确认

注意：

自定义属性必须掌握：html 中自定义的属性可以帮助索引元素

图片服务器的搭建

1. 创建一个 maven 的 web 工程，在工程中创建一个存放资源的目录
2. 把 tomcat 的 web.xml 中 DefaultServlet 的只读属性改成 false
3. 编写上传到图片服务器的代码

由于应用服务器与图片服务器出于不同的两台机器之中，所以可以提高系统的性能，能起到负载均衡的作用。上传图片时使用 Jersey 客户端 API 调用 REST 风格的 Web 服务，Jersey 1 是一个开源的、可以用于生产环境的 JAX-RS（RESTful Web Services 的 Java API 规范，JSR-311）实现。通过 Jersey 可以很方便的使用 Java 来创建一个 RESTful Web Services。

4. 图片服务器中 Upload 文件夹中随便建立一个文件，防止空文件夹的情况下发布后 upload 消失

表单验证

1. 提交时做验证

做好约定，每个文本中设置 reg 属性和 tip 自定义的属性，reg 存放正则表达式，tip 中存放不合法时的提示信息，还有品牌名称重复的验证。
reg2,tip 属于自定义的属性，这种定义方式方便使用 jquery 的属性选择器

```
<p>
  <label><samp>*</samp>品牌名称：</label>
  <input type="text" id="brandName" name="brandName" class="text state" reg2="^[a-zA-Z0-9\u4e00-\u9fa5]{1,20}$" tip="必须是中英文或数字字符，长度 1-20"/>
```

```
    <span></span>
</p>
```

2. 在表单提交时做验证使用$(“form”).submit(function(){ return false });,必填字段和非必填的字段需要区别对待

3. 使用离焦事件做友好的提示

4. 表单的二次提交处理

锁屏

锁按钮

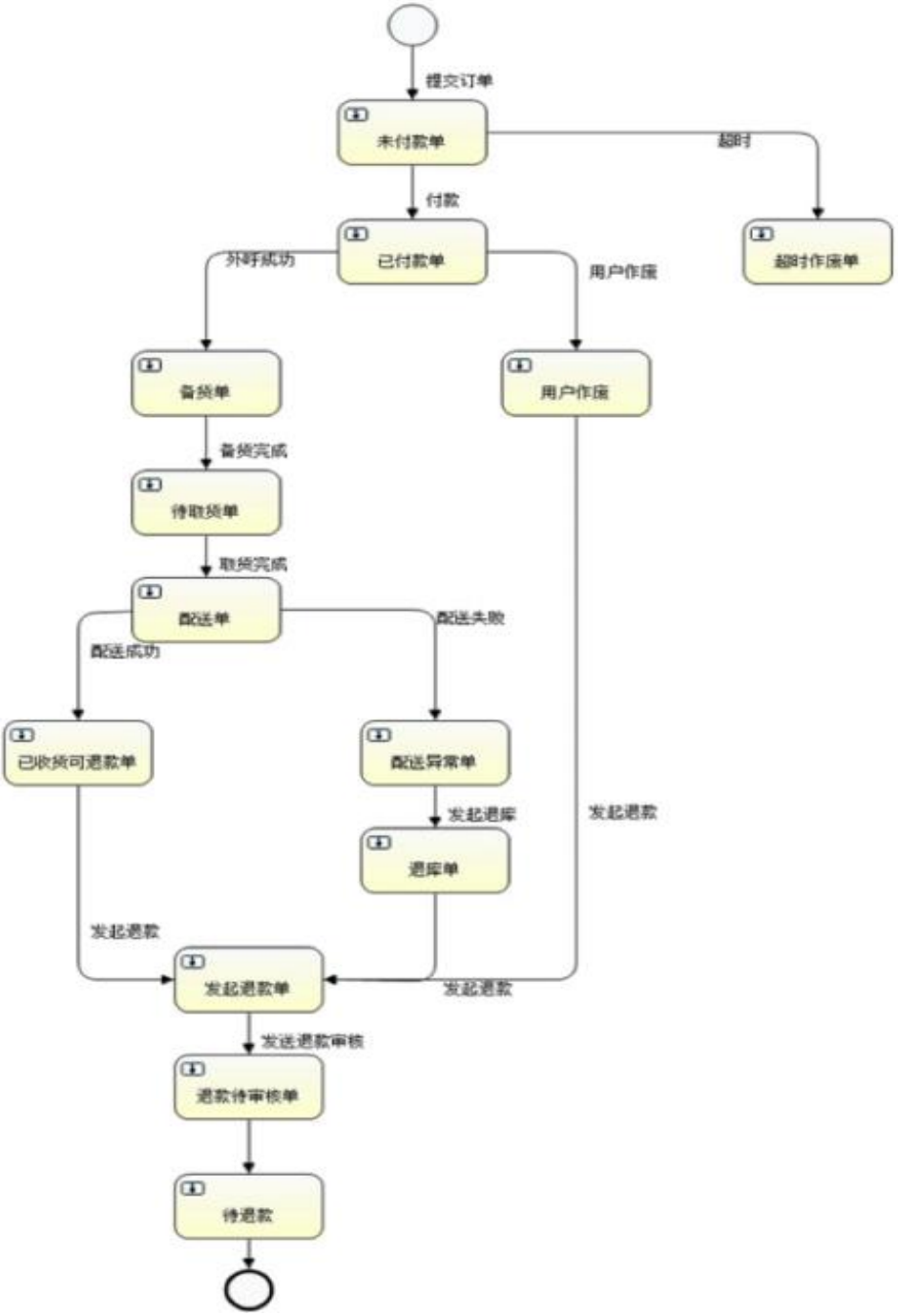
②商品的查询

商品查询需要组合条件加分页查询

组合条件：品牌，审核状态，商品名称，需要动态 sql

③订单管理模块

1) 流程设计



2) 订单整合 activiti 工作流

1.activiti 流程和 spring 整合

创建 activiti-context.xml 文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

    <bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 配置事务管理器，统一事务 -->
        <property name="transactionManager" ref="txManager" />
        <!-- 设置建表策略 -->
        <property name="databaseSchemaUpdate" value="true" />
    </bean>

    <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
        <property name="processEngineConfiguration" ref="processEngineConfiguration" />
    </bean>
    <bean id="repositoryService" factory-bean="processEngine" factory-method="getRepositoryService" />
    <bean id="runtimeService" factory-bean="processEngine" factory-method="getRuntimeService" />
    <bean id="taskService" factory-bean="processEngine" factory-method="getTaskService" />
    <bean id="historyService" factory-bean="processEngine" factory-method="getHistoryService" />
</beans>
```

1. 画流程图
2. 创建流程服务类

```
package cn.itcast.service.impl;
import java.io.File;
import org.activiti.engine.RepositoryService;
import org.activiti.engine.repository.DeploymentBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import cn.itcast.service.IWorkFlowService;

//@Service
public class WorkflowServiceImpl implements IWorkFlowService {
    @Autowired
```

```
    RepositoryService repositoryService;
    /*public RepositoryService getRepositoryService() {
        return repositoryService;
    }
    public void setRepositoryService(RepositoryService repositoryService) {
        this.repositoryService = repositoryService;
    }*/
    public void deployFlow() {
        //创建发布流程配置对象
        DeploymentBuilder builder = repositoryService.createDeployment();
        //指定流程资源路径
        builder.addClasspathResource("activit-orderflow.bpmn").addClasspathResource("activit-
orderflow.png");
        builder.deploy();
    }
}
```

1. 部署流程
2. 查询业务任务
3. 办理任务

7.6 项目中哪些功能模块涉及了大数据量访问？你是如何解决的？

系统前台是互联网上的用户访问的，会有大量用户来访问。
假定有 1w 个人打开你的网站来订商品，问你如何解决并发问题(可扩展到任何高并发网站要考虑的并发读写问题)（具体询问相关技术老师）
问题，1w 个人来访问，商品没出去前要保证大家都能看到有商品，不可能一个人在看到商品的时候别人就不能看了。到底谁能抢到，那得看这个人的“运气”（网络快慢等）
其次考虑的问题，并发，1w 个人同时点击购买，到底谁能成交？总共只有一张商品。

```
Update eb_sku t sett.stock = t.stock – 1 where t.sku_id = #{skuId} and t.stock > 0

Update eb_sku t set t.sale = t.sale +1 where t.sku_id = #{skuId}
```

首先我们容易想到和并发相关的几个方案：
解决大量用户访问量问题方案是集群部署，防止宕机，负载均衡
1.我们采用 4 台 portal 服务器来集群，使用 2 台 nginx 代理服务器
1）反向代理，把 4 台 portal 的服务（host）集中起来，访问动态链接代理地址（代理 IP：192.168.1.100）时候会把请求转发到 4 太 portal 上，如果

访问静态页面直接访问 nginx 上的资源文件就可以了，静态 html 中有 ajax 的请求由 nginx 的反向代理功能来转发。

2）部署静态资源（html 和图片）

2.Rsync 用作资源同步

当 console 上传的图片，由于 rsync 的部署，会指定一个具体的同步目录（上传图片的目录），一旦发现目录中有文件就立刻同步到 nginx 上

3.Redis 负责管理 session 和缓存搜索的数据

管理 session 的原因：用于多台服务器之间需要有相同的 session，同享策略耗费资源，所以采用 redis 来存储 session。缓存频繁被搜索的数据。

7.7 在做这个项目的时候你碰到了哪些问题？你是怎么解决的？

①.redis 受物理内存限制，操作海量数据读写的时候性能非常低；后期我们采用集群；

②.httpclient 耦合度偏高，后期我们采用 MQ;

③.项目中用到了曾经没有用过的技术，解决方式：用自己的私人时间主动学习

④.在开发过程中与测试人员产生一些问题，本地环境 ok 但是测试环境有问题，环境的问题产生的，浏览器环境差异，服务器之间的差异

⑤.系统运行环境问题，有些问题是在开发环境下 OK，但是到了测试环境就问题，比如说系统文件路径问题，导出报表中的中文问题（报表采用 highcharts），需要在系统 jdk 中添加相应的中文字体才能解决；

7.8 你做完这个项目后有什么收获？

首先，在数据库方面，我现在是真正地体会到数据库的设计真的是一个程序或软件设计的重要和根基。因为数据库怎么设计，直接影响到一个程序或软件的功能的实现方法、性能和维护。由于我做的模块是要对数据库的数据进行计算和操作的，所以我对数据库的设计对程序的影响是深有体会，就是因为我们的数据库设计得不好，搞得我在对数据库中的数据进行获取和计算利润、总金时，非常困难，而且运行效率低，时间和空间的复杂也高，而且维护起来很困难，过了不久，即使自己有注释，但是也要认真地看自己的代码才能明白自己当初的想法和做法。加上师兄的解说，让我对数据库的重要的认识更深一层，数据库的设计真的是重中之重。

其次，就是分工的问题。虽然这次的项目我们没有在四人选出一个组长，但是，由于我跟其他人都比较熟，也有他们的号码，然后我就像一个小组长一样，也是我对他们进行了分工。俗话也说，分工合作，分好了工，才能合作。但是这次项目，我们的分工却非常糟糕，我们在分工之前分好了模块，每个模块实现什么功能，每个人负责哪些模块。本以为我们的分工是明确的，后

来才发现，我们的分工是那么的一踏糊涂，一些功能上紧密相连的模块分给了两个人来完成，使两个人都感到迷惘，不知道自己要做什么，因为两个人做的东西差不多。我做的，他也在做，那我是否要继续做下去？总是有这样的疑问。从而导致了重复工作，浪费时间和精力，并打击了队员的激情，因为自己辛辛苦苦写的代码，最后可能没有派上用场。我也知道，没有一点经验的我犯这样的错是在所难免，我也不过多地怪责自己，吸取这次的教训就好。分工也是一门学问。

再者，就是命名规范的问题。可能我们以前都是自己一个人在写代码，写的代码都是给自己看的，所以我们都没有注意到这个问题。就像师兄说的那样，我们的代码看上去很上难看很不舒服，也不知道我们的变量是什么类型的，也不知道是要来做什么的。但是我觉得我们这一组人的代码都写得比较好看，每个人的代码都有注释和分隔，就是没有一个统一的规范，每个人都人自己的一个命名规则和习惯，也不能见名知义。还有就是没有定义好一些公共的部分，使每个人都有一个自己的“公共部分”，从而在拼起来时，第一件事，就是改名字。而这些都应该是在项目一开始，还没开始写代码时应该做的。

然后，我自己在计算时，竟然太大意算错了利润，这不能只一句我不小心就敷衍过去，也是我的责任，而且这也是我们的项目的核心部分，以后在做完一个模块后，一定要测试多次，不能过于随便地用一个数据测试一下，能成功就算了，要用可能出现的所有情况去测试程序，让所有的代码都有运行过一次，确认无误。

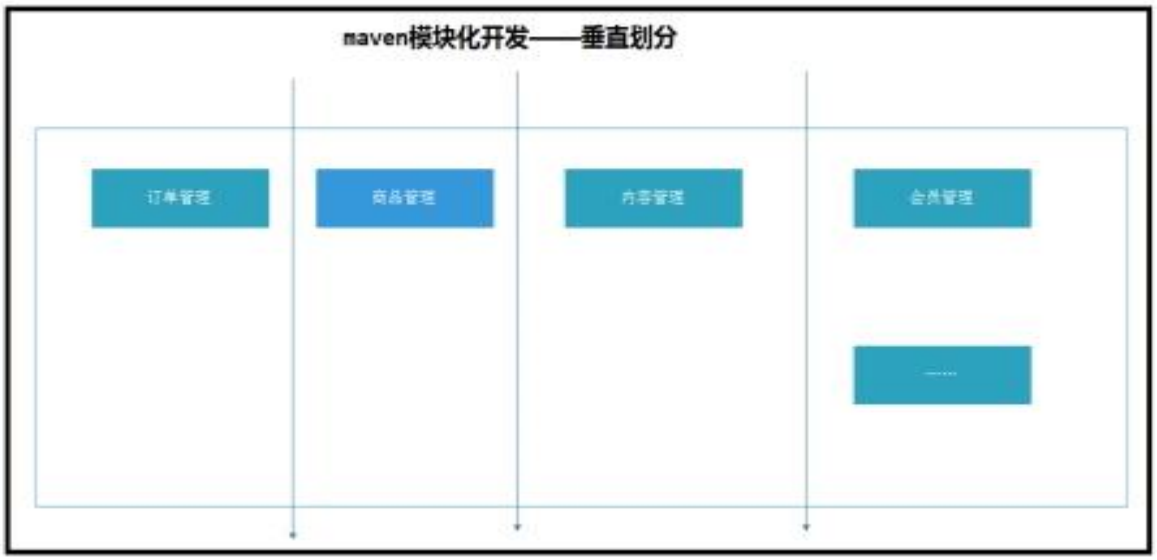
最后，也是我比较喜欢的东西，就是大家一起为了一个问题去讨论和去交流。因为我觉得，无论是谁，他能想的东西都是有限的，别人总会想到一些自己想不到的地方。跟他人讨论和交流能知道别人的想法、了解别人是怎样想一个问题的，对于同样的问题自己又是怎样想的，是别人的想法好，还是自己的想法好，好在什么地方。因为我发现问题的能力比较欠缺，所以我也总是喜欢别人问我问题，也喜欢跟别人去讨论一个问题，因为他们帮我发现了我自己没有发现的问题。在这次项目中，我跟植荣的讨论就最多了，很多时候都是不可开交的那种，不过我觉得他总是能够想到很多我想不到的东西，他想的东西也比我深入很多，虽然很多时候我们好像闹得很僵，但是我们还是很要好的!嘻嘻！而且在以后的学习和做项目的过程中，我们遇到的问题可能会多很多，复杂很多，我们一个人也不能解决，或者是没有想法，但是懂得与他人讨论与交流就不怕这个问题，总有人的想法会给我们带来一片新天地。相信我能做得更好。

还有就是做项目时要抓准客户的要求，不要自以为是，自己觉得这样好，那样好就把客户的需求改变，项目就是项目，就要根据客户的要求来完成。

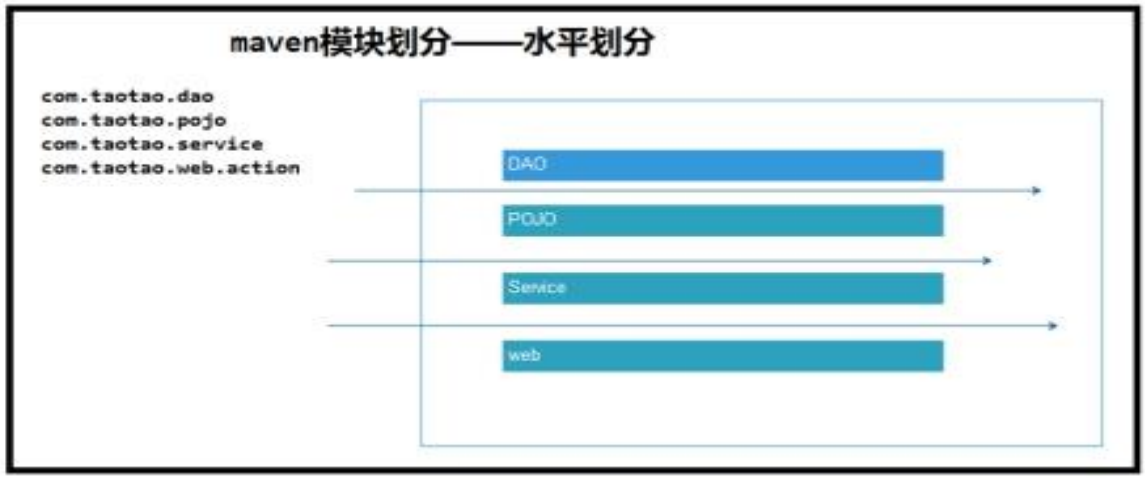
7.9 你这个项目中使用什么构建的？多模块开发是如何划分的呢？为什么要这么做？

我们这个项目使用 Maven 进行构建，并使用了水平划分，这样划分层次清晰，代码重用性高，易于独立维护。

① 垂直划分



②水平划分



优缺点：

垂直：功能模块明确，层次不够清晰，代码重用性差。

水平：层次清晰，代码重用性高，独立维护。

淘淘商城后台管理系统采取水平划分。

7.10 你觉得在图片上传功能上面需要注意什么？

SpringMVC 图片上传需要三步：

1、 加入 commons-fileupload 依赖

Taotao-manage-web　　pom.xml 里

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
```

2、 添加文件上传解析器

```
<!--视图解析器 -->
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"></property>
    <property name="suffix" value=".jsp"></property>
    <property name="order" value="2"></property>
</bean>

<!-- 过滤掉所有的静态资源，把静态资源交给服务器，spirngmvc 自己不处理 -->
<mvc:default-servlet-handler />

<!-- 配置文件的上传解析器 -->
<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!-- 设定默认编码 -->
    <property name="defaultEncoding" value="UTF-8"></property>
    <!-- 设定文件上传的最大值 5MB，5*1024*1024,如果一次性上传多个文件，则只的时　　候所有文件的总和 -->
    <property name="maxUploadSize" value="5242880"></property>
</bean>
```

3、 编写上传逻辑

图片回显 用的 Nginx 使用 nginx 访问图片（优化）

在实际企业中，一般不会使用 tomcat 作为图片的服务器，通过 nginx、apache 的 web 服务器作为图片的服务器。

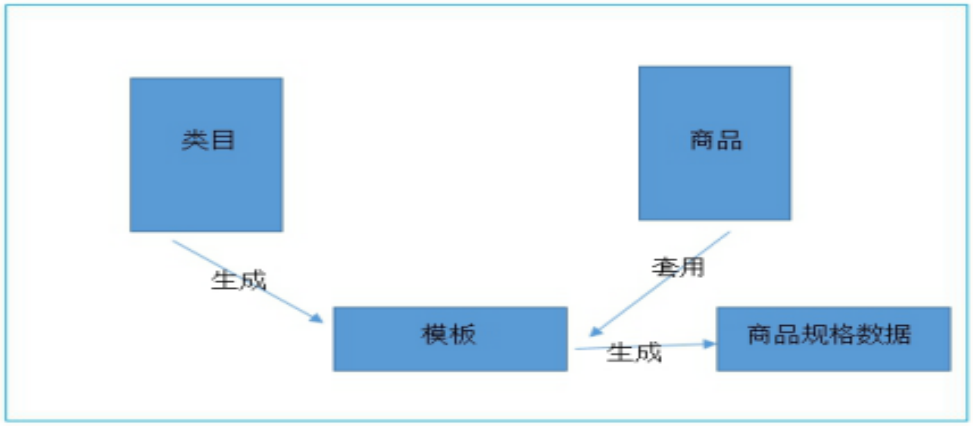
nginx 配置：　　nginx.conf 配置文件

```
55      server {
56          listen       80;
57          server_name  image.taotao.com;
58
59          #charset koi8-r;
60
61          #access_log  logs/host.access.log  main;
62
63          proxy_set_header X-Forwarded-Host $host;
64          proxy_set_header X-Forwarded-Server $host;
65          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
66
67          location / {
68              root  E:\\0716\\taotao-upload;
69          }
70
71      }
72
```

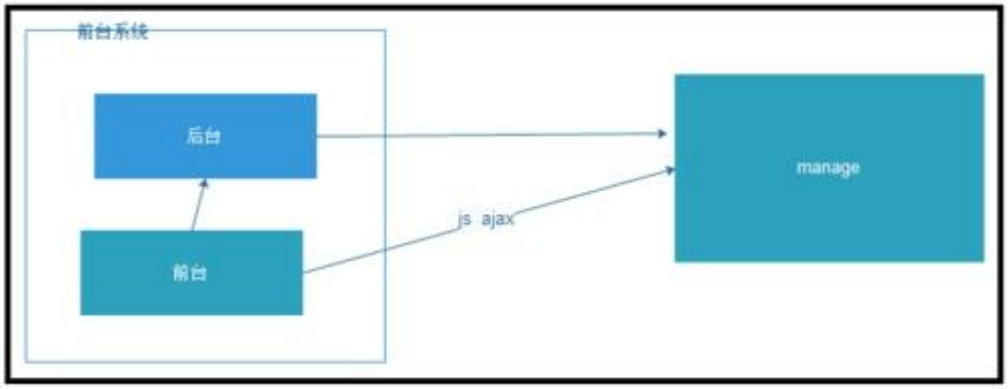
!

7.11 在你这个项目中，是如何设计商品规格的？

实现思路很重要！



7.12 在这个项目中你是如何实现跨系统调用的？



有两种调用方式：

1、 Ajax，走前台 js，通过 jsonp 来跨域，关于 jsonp 请参考：

http://www.cnblogs.com/yuzhongwusan/archive/2012/12/11/2812849.html

a) 效率

b) 带宽

document.domain="taotao.com"

jsonp，部署子域名的情况

需要在 SpringMVC 中扩展 MappingJackson2HttpMessageConverter，支持 jsonp。跨域问题，因为我们是子域名访问子系统接口的，采用 jsonp 解决；

2、 后台转发请求，走后台，通过 httpclient 来调。

a) 可以加逻辑(加缓存只能这条路走)

b) 安全，接口不在公网公开

重点学习 httpclient 中的示例

掌握 spring 和 Httpclient 的集成

我们这个项目 2 种方式都使用到了。

7.13 你这个项目中 CMS 系统是如何设计的，简单的说一下其设计思想？

隐藏在内容管理系统(CMS)之后的基本思想是分离内容的管理和设计。页面设计存储在模板里，而内容存储在数据库或独立的文件中。 当一个用户请求页面时，各部分联合生成一个标准的 HTML（标准通用标记语言下的一个应用）页面。

内容管理系统被分离成以下几个层面：各个层面优先考虑的需求不同

1，后台业务子系统管理（管理优先：内容管理）：新闻录入系统，BBS 论坛子系统，全文检索子系统等，针对不同系统的方便管理者的内容录入：所见即所得的编辑管理界面等，清晰的业务逻辑：各种子系统的权限控制机制等；

2，Portal 系统（表现优先：模板管理）：大部分最终的输出页面：网站首页，子频道/专题页，新闻详情页一般就是各种后台子系统模块的各种组合，这种发布组合逻辑是非常丰富的，Portal 系统就是负责以上这些后台子系统的组合表现管理；

3，前台发布（效率优先：发布管理）：面向最终用户的缓存发布，和搜索引擎 spider 的 URL 设计等……

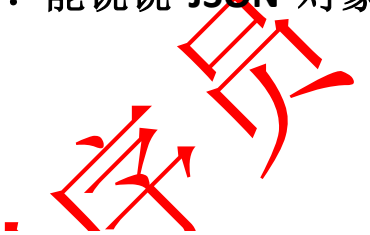
内容管理和表现的分离：很多成套的 CMS 系统没有把后台各种子系统和 Portal 分离开设计，以至于在 Portal 层的模板表现管理和新闻子系统的内容管理逻辑混合在一起，甚至和 BBS 等子系统的管理都耦合的非常高，整个系统会显得非常庞杂。而且这样的系统各个子系统捆绑的比较死，如果后台的模块很难改变。但是如果把后台各种子系统内容管理逻辑和前台的表现/发布分离后，Portal 和后台各个子系统之间只是数据传递的关系：Portal 只决定后台各个子系统数据的取舍和表现，而后台的各个子系统也都非常容易插拔。

内容管理和数据分发的分离：需要要 Portal 系统设计的时候注意可缓存性（Cache Friendly）性设计：CMS 后台管理和发布机制，本身不要过多考虑"

效率"问题，只要最终页面输出设计的比较 Cacheable，效率问题可通过更前端专门的缓存服务器解决。

此外，就是除了面向最终浏览器用户外，还要注意面向搜索引擎友好(Search engine Friendly)的 URL 设计：通过 URL REWRITE 转向或基于 PATH_INFO 的参数解析使得动态网页在链接（URI）形式上更像静态的目录结构，方便网站内容被搜索引擎收录；

7.14 在这个项目中，你们主要使用什么样的数据格式来进行数据的传输的？你对 JSON 了解么？能说说 JSON 对象如何转换成 Java 对象的？



第一种方法，使用 JSON-lib 。

Json-lib 依赖下面几个包：

在实际项目中，为了代码复用性，减少代码冗余，常常对某些方法进行封装，用到的时候直接调用，不必要每次都重写相同代码。

Json-lib 可以处理集合或者单个对象转换成 json 字符串，针对集合或者单个对象 json-lib 采用了不同的方法来分开处理：

1) 当参数为单个对象

```
//将参数对象转化成 JSONObject 对象
JSONObject jsonObject = JSONObject.fromObject(pObject);
//通过 JSONObject 的 toString()方法得到 json 字符串
String jsonString = jsonObject.toString();
```

2) 当参数为集合类型

```
//将参数对象转换成 JSONArray 对象
JSONArray jsonArray = JSONArray.fromObject(pObject);
//通过 jsonArray 的 toString()方法得到 json 字符串
```

```
String  jsonString = jsonArray.toString();
```

※json-lib 在处理日期格式化的时候要注意，如果你要格式化输出日期，需要手动实现 JsonValueProcessor 接口，来对日期进行处理。
JsonValueProcessor 接口中声明了两个方法：
processArrayValue(Object value, JsonConfig jsonConfig)
和 processObjectValue(String key, Object value,JsonConfig jsonConfig)顾名思义，分别针对数组集合和单个对象进行处理。

第二种方法，使用 JACKSON。

第三种方法，用 google 的一个 gson 包

第四种方法，JavaScript 对象表示法（JavaScript Object Notation）

3.总结

Jackson 关于 Json 的操作主要如上所示，其方法使用起来很便利，而且也很灵活，即提供了一次性完成的操作，也提供了可以按需读取信息的操作。并且 Jackson 的功能很齐全，可以对序列化和反序列化进行多种细节的控制，例如注解功能和对于 Hibernate 的延迟注入功能以及设置时间格式功能等，因为这些功能目前不太需要，所以仔细研究留待以后。同时，Jackson 还支持对 XML 的一系列序列化和反序列化的操作，其思路与解析 Json 的大致相同。

对于 Jackson 目前的缺点，网上有人测试所比 Json-lib 更占内存一些。而利用空间换时间，一般是值得的。

7.15 单点系统的设计思想你了解吗？他在系统架构中的作用是什么？位置如何？

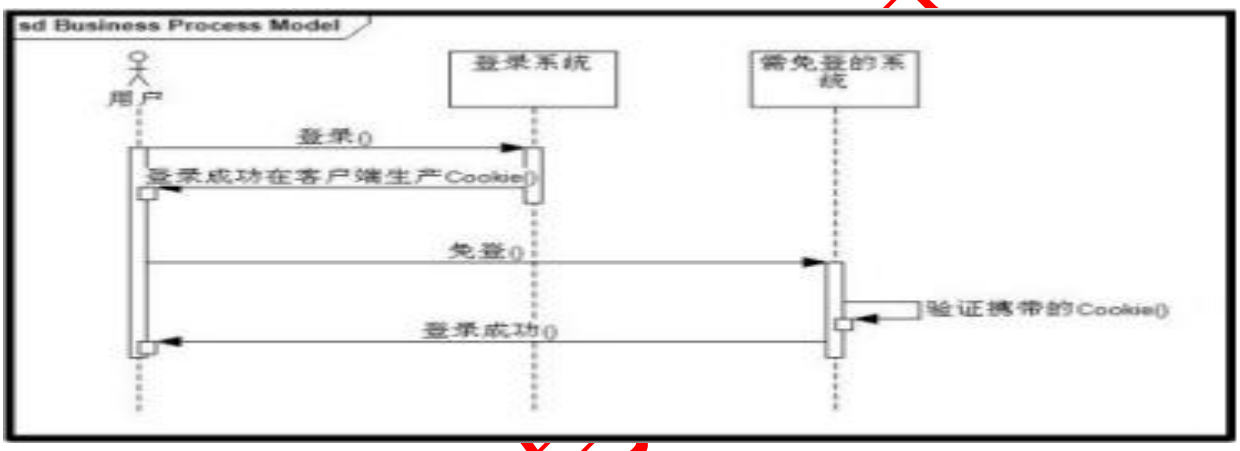
单点登录 SSO（Single Sign On）说得简单点就是在一个多系统共存的环境下，用户在一处登录后，就不用在其他系统中登录，也就是用户的一次登录能得到其他所有系统的信任。单点登录在大型网站里使用得非常频繁，例如像阿里巴巴这样的网站，在网站的背后是成百上千的子系统，用户一次操作或交易可能涉及到几十个子系统的协作，如果每个子系统都需要用户认证，不仅用

户会疯掉，各子系统也会为这种重复认证授权的逻辑搞疯掉。实现单点登录说到底就是要解决如何产生和存储那个信任，再就是其他系统如何验证这个信任的有效性，因此要点也就以下几个：

存储信任

验证信任

只要解决了以上的问题，达到了开头讲得效果就可以说是 SSO。最简单实现 SSO 的方法就是用 Cookie，实现流程如下所示：



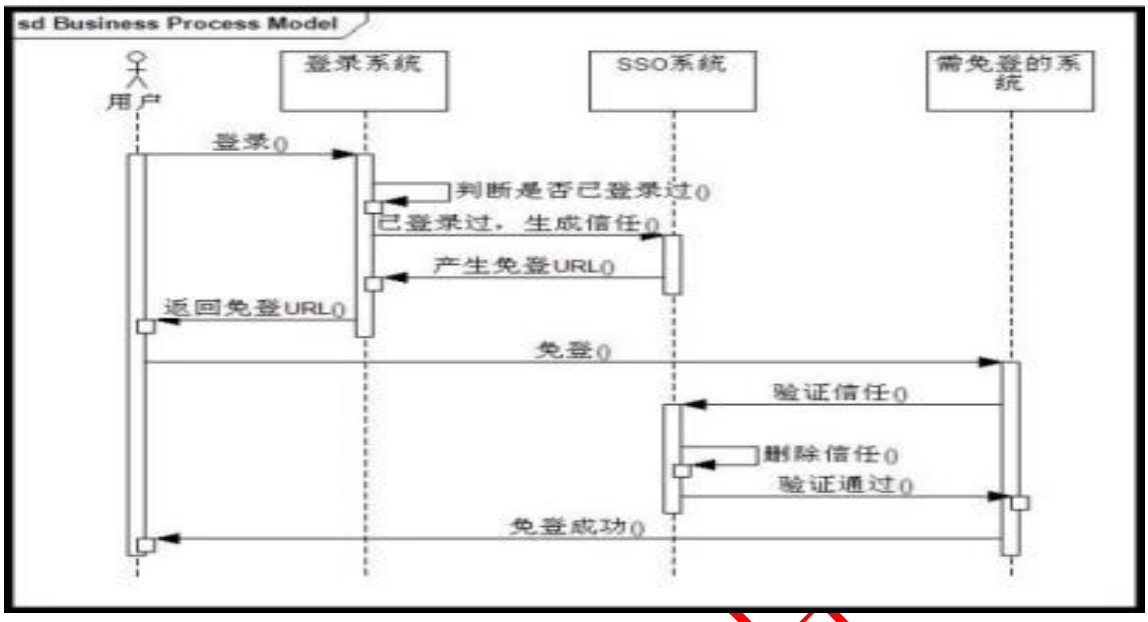
不难发现以上的方案是把信任存储在客户端的 Cookie 里，这种方法虽然实现方便但立马会让人质疑两个问题：

Cookie 不安全

不能跨域免登

对于第一个问题一般都是通过加密 Cookie 来处理，第二个问题是硬伤，其实这种方案的思路的就是要把这个信任关系存储在客户端，要实现这个也不一定只能用 Cookie，用 flash 也能解决，flash 的 Shared Object API 就提供了存储能力。

一般说来，大型系统会采取在服务端存储信任关系的做法，实现流程如下所示：



以上方案就是要把信任关系存储在单独的 SSO 系统（暂且这么称呼它）里，说起来只是简单地从客户端移到了服务端，但其中几个问题需要重点解决：

如何高效存储大量临时性的信任数据

如何防止信息传递过程被篡改

如何让 SSO 系统信任登录系统和免登系统

对于第一个问题，一般可以采用类似与 redis 的分布式缓存的方案，既能提供可扩展数据量的机制，也能提供高效访问。对于第二个问题，一般采取数字签名的方法，要么通过数字证书签名，要么通过像 md5 的方式，这就需要 SSO 系统返回免登 URL 的时候对需验证的参数进行 md5 加密，并带上 token 一起返回，最后需免登的系统进行验证信任关系的时候，需把这个 token 传给 SSO 系统，SSO 系统通过对 token 的验证就可以辨别信息是否被改过。对于最后一个问题，可以通过白名单来处理，说简单点只有在白名单上的系统才能请求生产信任关系，同理只有在白名单上的系统才能被免登录。

7.16 你们这个项目中订单 ID 是怎么生成的？我们公司最近打算做一个电商项目，如果让你设计这块，你会考虑哪些问题？

生成订单 ID 的目的是为了使订单不重复，本系统订单 ID 生成规则：
用户 ID+当前系统的时间戳

. String orderId = order.getUserId() + "" + System.currentTimeMillis();

设计的时候我会考虑：

订单 ID 不能重复

订单 ID 尽可能的短（占用存储空间少，实际使用方便，客服相关）

订单 ID 要求是全数字（客服）

7.17 各个服务器的时间不统一怎么办？

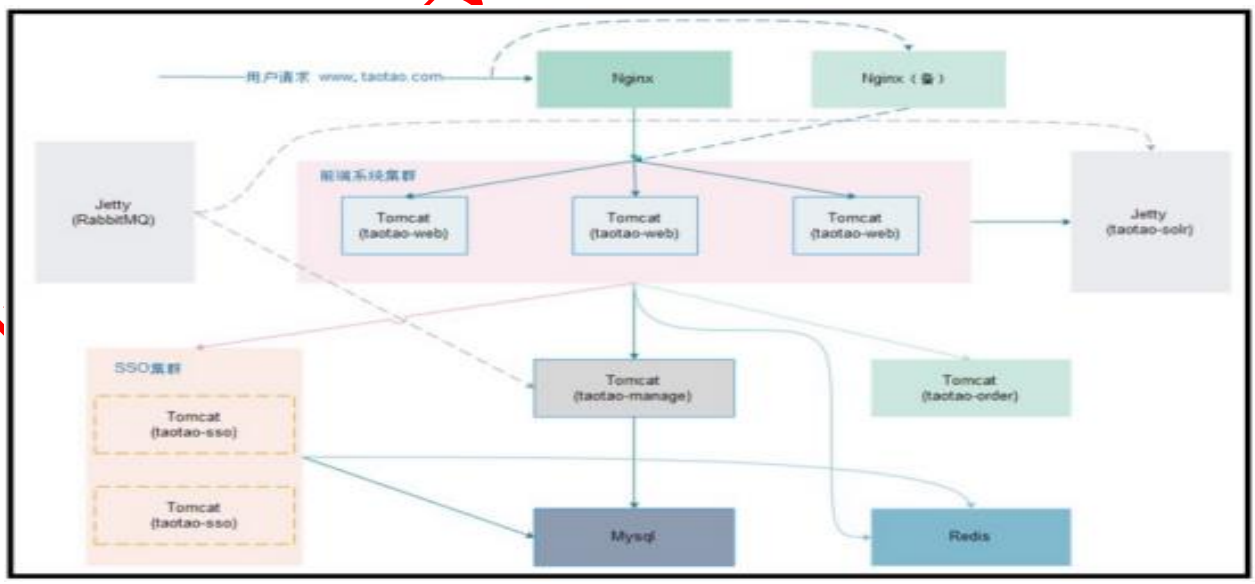
在各个服务器上做时间的统一；（运维）

7.18 在问题 17 的基础上，可能存在毫秒级的偏差情况，怎么办？

修改订单生成规则：

用户 ID+当前系统的时间戳+随机数（3~4 位） 问题：太长？ 把时间戳中的 2016 中的 20 拿掉；

7.19 你们线上部署时什么样的，能画一下吗？



7.20.如何解决并发问题的？

水桶原理,主要对于最短板的模块进行弥补.

1. 前台首页 静态化 缓存 服务器的集群

2. 后台系统 服务器集群 缓存 读写分离
3. 订单模块 使用 MQ 进行流量限制.

7.21 你们生产环境的服务器有多少台？（重点以 web 服务器为主）

面试前要数好，一般是十几到二十台。(用在哪里？这是重点)

Nginx 至少 2 台
Tomcat 至少 3 台以上(每个模块需要搭集群至少 3 台)
数据库至少 2 台
Redis 至少一台
什么是硬件什么是虚拟化.硬件就是实际用的机器.虚拟化就是一台硬件机器可以开启多态的虚拟化.

7.22.数据备份是怎么做的？有没有做读写分离？

主从(一主多从，主要是备份主)，每天备份，备份的文件不要放到数据库服务器上，可以 FTP。要检查有效否。读写分离自己查一下，分库分表做过。

7.23 你们使用什么做支付的？如果使用易宝做支付，请求超时了怎么处理？

①重试，一般三次，每次重试都要停顿一会，比如，以第一次停顿 1 秒，第二次停顿 2 秒，第三次停顿 3 秒；
②给订单标识付款异常状态，并且发出警告(邮件 JavaMail、短信)给相关人员。
③写个定时任务，定时处理异常状态的订单。(一般在凌晨比较空闲的时间)

7.24 付款成功后易宝会有数据返回吗？如果付款后易宝没有返回，或者返回超时了，但是钱又已经扣了，你怎么办？

①我们请求了易宝，但是没有接受到响应，我们就认为该订单没有支付成功，并且将订单标识为异常状态；
②使用定时任务处理；
③做一个对账的任务，实时处理异常状态的订单。

7.25 你们怎么做退款功能的，要多长时间才能把钱退回给用户？

用户申请退款后，经过客服审核通过会将退款请求提交到易宝，具体到账时间要看易宝的处理。

7.26 不同域名的网站如何实现用户信息共享

更换电脑，必须登录才能看到之前购物车的商品。

跨域 cookie 同步方案：

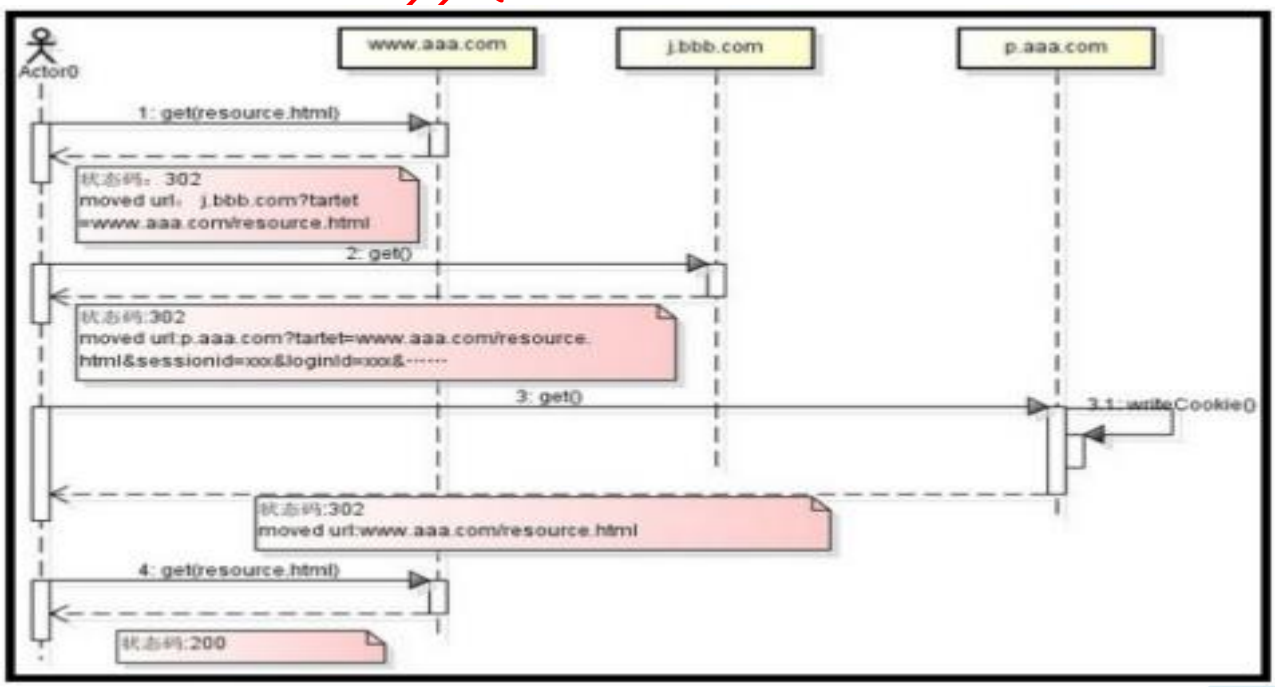
场景：有时一个公司可能有多个不同域名的网站，比如 sina.com 和 weibo.cn,比如 taobao.com 和 tmall.com。

这些网站背后很多是同一套会员体系。由于 http 协议规定 cookie 是跟着域名走的，这时就需要在不同的域名下同步登陆状态，避免出现用户体验上出现需要二次登陆验证的情况。

假设下面这样一个场景：

用户在 bbb.com 上已经登陆，现在要去 aaa.com 上玩，在 aaa.com 域名下暂未登录。需要访问的 aaa.com/resource.html 资源需 要登录才能访问。两个网站是同一套会员体系，同一个公司的。这是要让用户体验上做到用户在 aaa.com 上玩也能识别出登录状态。

以上面场景为例，下面画了个实现跨域同步简单流程图：



解释如下：

第一步 ：用户向 aaa.com 发起 get 请求，获取 resource.html 资源，aaa.com 发现用户未登录，返回 302 状态和外部重定向 url:

Java 代码 收藏代码

. j.bbb.com?target=www.aaa.com/resource.html

注意 j.bbb.com 子域名上部署的应用可以认为是专门用了跨域同步。

第二步 ：用户根据重定向 url，访问 j.bbb.com?target=www.aaa.com/resource.html，由于在 bbb.com 上已经登录，所以 bbb.com 上能拿到从 client 端传递过来 cookie 信息。子域 j.bbb.com 上的应用负责将 cookie 读取出来，并作为参数再次 重定向到

Java 代码 收藏代码

. p.aaa.com?tartet=www.aaa.com/resource.html&sessionid=xxx&loginId=xxx&……

第三步 ：用户根据第二步重定向 url，访问 p.aaa.com。p.aaa.com 子域名上的应用专门负责根据请求参数里的参数对，往 aaa.com 域写入 cookie，并重定向到用户第一步请求的 url。

第四步 ：经过前三步，已经完成了再 aaa.com 域名下同步 bbb.com 的登录状态，用户再次请求 aaa.com/resource.html,这是就能成功访问了。

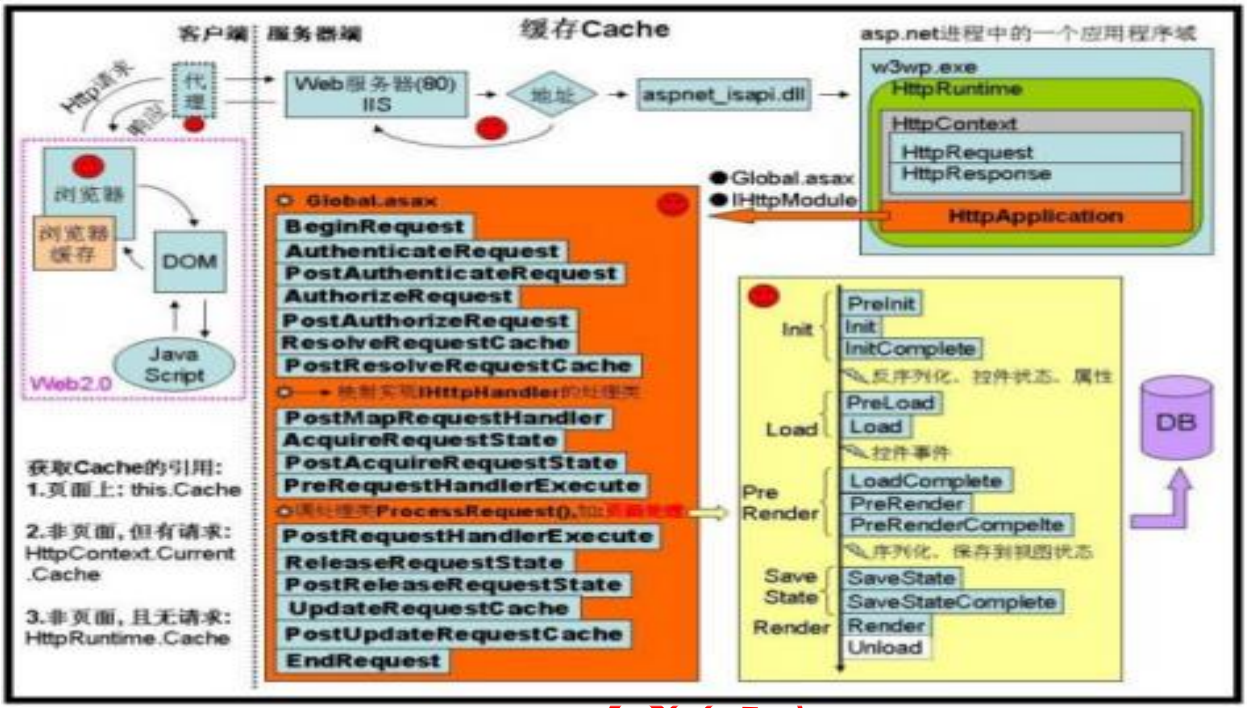
7.27 点一个链接访问到一个页面，这个页面上既有静态数据，又有动态数据（需要查数据库的），打开这个页面的时候就是很慢但是也能打开。怎么解决这个问题，怎么优化？(静态化)

如果要静态页面的话 那就得用 freemarker 或者通过 ajax 异步,通过 js 操作异步刷新表单,通过 js 对返回结果组装成 html。

缓存、动态页面静态化：

所谓缓存，是指将那些经常重复的操作结果暂时存放起来，在以后的执行过程中，只要使用前面的暂存结果即可.

那么在我们开发 Web 网站的过程中，到底有多少工作可以采用用缓存呢？或者说，我们可以在哪些地方使用缓存呢？见下图：下图是客户端浏览器和 Web 服务器之间的一次完整的通信过程，红色圆圈标示了可以采用缓存的地方.



首先，最好的情况是客户端不发送任何请求直接就能获得数据，这种情况下，用于缓存的数据保存在客户端浏览器的缓存中.

其次，在具有代理服务器的网络环境中，代理服务器可以针对那些经常访问的网页制作缓存，当局域网中第一台主机请求了某个网页并返回结果后，局域网中的第二台主机再请求同一个网页，这时代理服务器会直接返回上一次缓存的结果，并不会向网络中的 IIS 服务器发送请求，例如：现在的连接电信和网通线路的加速器等. 但是代理服务器通常有自己专门的管理软件和管理系统，作为网站开发人员对代理服务器的控制能力有限.

再次，前面也说过，当用户将请求地址发送到 IIS 服务器时，IIS 服务器会根据请求地址选择不同的行为，如：对于*.aspx 页面会走应用程序管道，而对*.html、*.jpg 等资源会直接返回资源，那么我们可以把那些频繁访问的页面做成*.html 文件，这样用户请求*.html，将不用再走应用程序管道，因此会提升效率. 例如：网站首页、或某些突发新闻或突发事件等，可以考虑做成静态网页. 就拿”天气预报的发布页面”打比方：天气预报的发布页面的访问用户非常多，我们可以考虑将发布页做成静态的*.html 网页，之后在整个网站程序启动时，在 Gboabl.asax 的 Application_Start 事件处理器中，创建子线程以实现每 3 个小时重新获取数据生成新的天气发布页面内容.

之后的 asp.net 的处理流程，作为程序员我们是无法干涉的. 直到启动 HttpApplication 管道后，我们才可以通过 Global.asax 或 IHttpModule 来控制请求处理过程，在应用程序管道中适合做整页或用户控件的缓存. 如：缓存热门页面，我们可以自动缓存整个网站中访问量超过一定数值(阀值)的页面，其中为了减小 IO 操作，将缓存的页面放在内容中.

7.28 如果用户一直向购物车添加商品怎么办？并且他添加一次你查询一次数据库？互联网上用户那么多，这样会对数据库造成很大压力你怎么办？(购物车 redis 存储)

首先我们使用 redis 作为一个缓冲的内存数据库作为存储用户购物车的信息,当用户登陆以后,讲 redis 的信息写入关系型数据库.第二当用户是登录状态的情况下,也是使用 redis 作为缓存,当用户做出关闭当前页面操作的时候,将 redis 的内容统一写入到数据库之中.为了提高 redis 的运行效率我没还可以使用 redis 中的 hash 结构.来具体的找到某个字段,做小范围的修改.也可以减轻 redis 的压力.

7.29 做促销时，商品详情页面的静态页面如何处理价格问题。

京东商品详情页虽然仅是单个页面，但是其数据聚合源是非常多的，除了一些实时性要求比较高的如价格、库存、服务支持等通过 AJAX 异步加载加载之外，其他的数据都是在后端做数据聚合然后拼装网页模板的。整个京东有数亿商品，如果每次动态获取如上内容进行模板拼装，数据来源之多足以造成性能无法满足要求；最初的解决方案是生成静态页，但是静态页的最大的问题：

1、无法迅速响应页面需求变更；

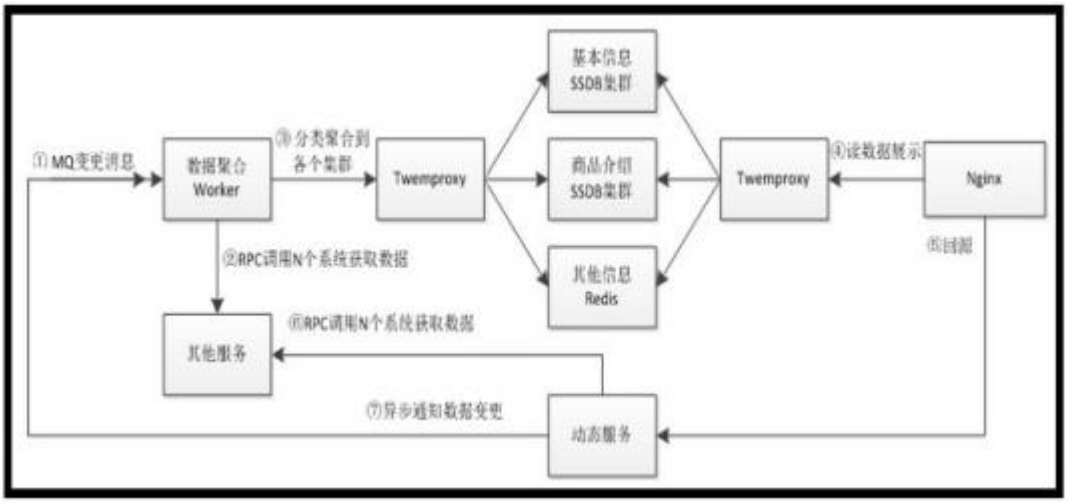
2、很难做多版本线上对比测试。如上两个因素足以制约商品页的多样化发展，因此静态化技术不是很好的方案。

数据主要分为四种：商品页基本信息、商品介绍（异步加载）、其他信息（分类、品牌、店铺等）、其他需要实时展示的数据（价格、库存等）。而其他信息如分类、品牌、店铺是非常少的，完全可以放到一个占用内存很小的 Redis 中存储；而商品基本信息我们可以借鉴静态化技术将数据做聚合存储，这样的好处是数据是原子的，而模板是随时可变的，吸收了静态页聚合的优点，弥补了静态页的多版本缺点；另外一个非常严重的问题就是严重依赖这些相关系统，如果它们挂了或响应慢则商品页就挂了或响应慢；商品介绍我们也通过 AJAX 技术惰性加载（因为是第二屏，只有当用户滚动鼠标到该屏时才显示）；而实时展示数据通过 AJAX 技术做异步加载

1、接收商品变更消息，做商品基本信息的聚合，即从多个数据源获取商品相关信息如图片列表、颜色尺码、规格参数、扩展属性等等，聚合为一个大的 JSON 数据做成数据闭环，以 key-value 存储；因为是闭环，即使依赖的系统挂了我们商品页还是能继续服务的，对商品页不会造成任何影响；

2、接收商品介绍变更消息，存储商品介绍信息；

3、介绍其他信息变更消息，存储其他信息



技术选型

MQ 可以使用如 Apache ActiveMQ；

Worker/动态服务可以通过如 Java 技术实现；

RPC 可以选择如 alibaba Dubbo；

KV 持久化存储可以选择 SSDB（如果使用 SSD 盘则可以选择 SSDB+RocksDB 引擎）或者 ARDB（LMDB 引擎版）；

缓存使用 Redis

SSDB/Redis 分片使用如 Twemproxy，这样不管使用 Java 还是 Nginx+Lua，它们都不关心分片逻辑；

前端模板拼装使用 Nginx+Lua；

数据集群数据存储的机器可以采用 RAID 技术或者主从模式防止单点故障；

因为数据变更不频繁，可以考虑 SSD 替代机械硬盘。

核心流程

1、首先我们监听商品数据变更消息；

2、接收到消息后，数据聚合 Worker 通过 RPC 调用相关系统获取所有要展示的数据，此处获取数据的来源可能非常多而且响应速度完全受制于这些系统，可能耗时几百毫秒甚至上秒的时间；

3、将数据聚合为 JSON 串存储到相关数据集群；

4、前端 Nginx 通过 Lua 获取相关集群的数据进行展示；商品页需要获取基本信息+其他信息进行模板拼装，即拼装模板仅需要两次调用（另外因为其他信息数据量少且对一致性要求不高，因此我们完全可以缓存到 Nginx 本地全局内存，这样可以减少远程调用提高性能）；当页面滚动到商品介绍页面时异步调用商品介绍服务获取数据；

5、如果从聚合的 SSDB 集群/Redis 中获取不到相关数据；则回源到动态服务通过 RPC 调用相关系统获取所有要展示的数据返回（此处可以做限流处理，因为如果大量请求过来的话可能导致服务雪崩，需要采取保护措施），此

处的逻辑和数据聚合 Worker 完全一样；然后发送 MQ 通知数据变更，这样下次访问时就可以从聚合的 SSDB 集群/Redis 中获取数据了。

基本流程如上所述，主要分为 Worker、动态服务、数据存储和前端展示；因为系统非常复杂，只介绍动态服务和前端展示、数据存储架构；Worker 部分不做实现。

7.30 一个电商项目，在 tomcat 里面部署要打几个 war 包？

分布式的系统,有几个独立的系统就需要打几个 war 包。

7.31 你说你用了 redis 缓存，你 redis 存的是什么格式的数据，是怎么存的？

1 Key

Key 不能太长，比如1024字节，但antirez也不喜欢太短如"u:1000:pwd"，要表达清楚意思才好。他私人建议用":"分隔域，用"."作为单词间的连接，如"comment:1234:reply.to"。

Keys，返回匹配的key，支持通配符如 "keys a*" 、 "keys a?c"，但不建议在生产环境大数据量下使用。

2 String

最普通的key-value类型，说是String，其实是任意的byte[]，比如图片，最大512M。所有常用命令的复杂度都是O(1)，普通的Get/Set方法，可以用来做Cache，存Session，为了简化架构甚至可以替换掉Memcached。

3 Hash

Key-HashMap结构，相比String类型将这整个对象持久化成JSON格式，Hash将对象的各个属性存入Map里，可以只读取/更新对象的某些属性。

这样有些属性超长就让它一边呆着不动，另外不同的模块可以只更新自己关心的属性而不会互相并发覆盖冲突。

另一个用法是土法建索引。比如User对象，除了id有时还要按name来查询。

可以有如下的数据记录:

(String) user:101 -> {"id":101,"name":"calvin"...}
(String) user:102 -> {"id":102,"name":"kevin"...}
(Hash) user:index-> "calvin"->101, "kevin" -> 102

底层实现是hash table，一般操作复杂度是O(1)，要同时操作多个field时就是O(N)，N是field的数量。

4 List

List是一个双向链表，支持双向的Pop/Push，江湖规矩一般从左端Push，右端Pop——LPush/RPop，而且还有Blocking的版本BLPop/BRPop，客户端可以阻塞在那直到有消息到来，所有操作都是O(1)的好孩子，可以当Message Queue来用。

在消息队列中，并没有JMS的ack机制，如果消费者把job给Pop走了又没处理完就死机了怎么办？

解决方法之一是加多一个sorted set，分发的时候同时发到list与sorted set，以分发时间为score，用户把job做完了之后要用ZREM消掉sorted set里的job，并且定时从sorted set中取出超时没有完成的任务，重新放回list。

另一个做法是为每个worker多加一个的list，弹出任务时改用RPopLPush，将job同时放到worker自己的list中，完成时用LREM消掉。

如果集群管理(如zookeeper)发现worker已经挂掉，就将worker的list内容重新放回主list。

5 Set

Set就是Set，可以将重复的元素随便放入而Set会自动去重，底层实现也是hash table。

SAdd/SRem/SIsMember/SCard/SMove/SMembers，各种标准操作。除了SMembers都是O(1)。

SInter/SInterStore/SUnion/SUnionStore/SDiff/SDiffStore，各种集合操作。交集运算可以用来显示在线好友(在线用户 交集 好友列表)，共同关注(两个用户的关注列表的交集)。

O(N)，并集和差集的N是集合大小之和，交集的N是小的那个集合的大小*2。

6 Sorted Set

有序集，元素放入集合时还要提供该元素的分数。

ZRange/ZRevRange，按排名的上下限返回元素，正数与倒数。

ZRangeByScore/ZRevRangeByScore，按分数的上下限返回元素，正数与倒数。

ZRemRangeByRank/ZRemRangeByScore，按排名/按分数的上下限删除元素。

ZCount，统计分数上下限之间的元素个数。

ZRank/ZRevRank ，显示某个元素的正倒序的排名。

ZScore/ZIncrby，显示元素的分数/增加元素的分数。

ZAdd(Add)/ZRem(Remove)/ZCard(Count)，ZInsertStore(交集)/ZUnionStore(并集)，Set操作，与正牌Set相比，少了IsMember和差集运算。

Sorted Set的实现是hash table(element->score，用于实现ZScore及判断element是否在集合内)，和skip list(score->element,按score排序)的混合体。

skip list有点像平衡二叉树那样，不同范围的score被分成一层一层，每层是一个按score排序的链表。

ZAdd/ZRem是O(log(N))，ZRangeByScore/ZRemRangeByScore是O(log(N)+M)，N是Set大小，M是结果/操作元素的个数。

可见，原本可能很大的N被很关键的Log了一下，1000万大小的Set，复杂度也只是几十不到。

当然，如果一次命中很多元素M很大那谁也没办法了。

7.32 购物车知识补充(在设计购物车时需要注意哪些细节)

为什么购物车的设计很重要？

①购物车是消费的最后一环

购物车在用户整体消费过程中一般是在最后一环，用户完整的消费体验应该是：打开APP或网站->浏览商品->加入购物车->确认订单并支付，在这个过程中，购物车和支付环节可以合并成一环，基本上用户点开购物车并开始填写地址的时候，就有很大的几率要完成购买，做好商品展现以及推送的环节，如果在最后的购物一环没有好的用户体验，岂不呜呼哀哉。

②购物车隐含的对比收藏功能

与现实购物车不同的是，网络消费者也比较喜欢把看中但不计划买的商品先放入购物车，或者把商品统一放到购物车直接进行比较，以备日后购买，因此从购物车保存的信息，就能够知道用户的大致偏好。

③购物车的重交易属性

用户在浏览商品涉及的只是前端展示，但购物车这一环涉及到最终的交易，对于用户来说，需要了解本次交易的基本物品信息、价格信息；而对于商户来说，确认收款、订单生成、物流环节都需要在这里获取到信息，才能完成本次的交易。

购物车设计需要展示的基本信息

购物车主要作用就是告诉用户买了什么，价格多少，不同类型的物品可能会有不同展示方式，但最基本的包括商品名称、价格、数量（若是服务，可能是次数）、其他附属信息。

哪些细节要让用户买得舒服？

亲，记得前面说的用户是如何看待购物车的功能吗？还记得你的用户会多次使用购物车，如果你只是完整做好信息展示不做好其他事情真的好吗？

①登录环节不要放在加入购物车前

请让用户先加入购物车，并在进行结算的时候在提醒用户需要登录。为什么？过早提醒用户需要登录才能购买，会打断用户浏览的流程（用户可能还要购买其他物品好吗？）这样的设置会让部分用户避而远之。

这里涉及到的一个点是在APP端需要记忆用户加入购物车的信息，与登录后的购物车信息合并（如果一开始没有这样考虑好，技术那可能会有难度）

②自动勾选用户本次挑选的商品

用户使用购物车有一个大的作用就是收藏，所以你要知道很多用户在购物车中积累了很多物品，当每次挑选加入购物车的商品，用户每次来到购物车要重新把本次的购买商品选上是很不好的体验。

所以这里一般是自动勾选本次挑选的商品，同样这里也要储存用户的勾选信息。

③陈列展示，注意沉底商品

让用户看见当前想买的商品就好了，把一些时间久远的，已经卖完的沉底显示。这样做的好处是能让用户看见之前的选择但没购买的商品，提醒一下说不定就又勾上买了哦！

④归类展示，可能增加购买

考虑如何进行归类展示，C2C可以按照商家分类，B2C可以按照品牌分类。

⑤价格和优惠的提醒

消费用户会关系自己每一次的消费价格，为避免商品列表过长隐藏价格信息，APP端一般会把总价固定底部提示。同时在合计信息中，展示优惠价格，能够促进消费者购买。

哪些细节要推动用户继续购买？

①还差一点就可以有优惠啦！

凑单，常用的手段包括运费见面或是满减促销，一般在网站底部会展示一些适合凑单的商品；在APP端可以给链接（不过需要权衡用户跳转会不会再跳回来哦！）

②提醒用户有些商品你真的可以买了

有关调查显示，加入购物车而没有购买的，在4小时以内提醒用户，会有27%的唤醒率哦！

所以需要提醒的几个点有：

生成订单但是还没支付的

商品有优惠信息

商品库存不足的

这些信息可以促进消费者购买，注意提醒的时间段，早上9点至晚上8点为宜，其他时间段就可能打扰用户咯（当然也要视产品类型而定啦，只不过大半夜提醒用户买东西确实不好，不是？）

八 传统项目

8.1 什么是 BOS ？

BOS是ERP的集成与应用平台。BOS遵循面向服务的架构体系，是一个面向业务的可视化开发平台；是一个ERP和第三方应用集成的技术平台。它有效的解决了ERP应用的最主要矛盾－－－用户需求个性化和传统ERP软件标准化之间的矛盾。

BOS与ERP是什么关系？

ERP是企业管理信息化的全面解决方案，ERP是基于BOS构建的。ERP满足企业全面业务的标准应用；BOS确保了企业ERP应用中的个性化需求完美实现。基于BOS的ERP，可以为不同行业不同发展阶段的企业构建灵活的、可扩展的、全面集成的整体解决方案。

8.1.1 项目开发流程

需求 --- 需求分析写文档（3个月） --设计 ---- 编码 ---- 测试 ---- 实施

分析需求1-2个月+代码编程3个月+初步上线3个月+系统运维2个月

需求分析2人+代码编写4人+测试2人+项目经理2人

8.1.2Bos中所使用的技术点

Struts2 表现层框架 Poi-office 文档读写组件

Hibernate 数据层持久化框架 Lucene、hibernatesearch 全文索引库技术

Spring,springData业务管理 IOC和aop框架 Hessian、CXFRMI远程调用

Junit	元测试	Activiti	工作流框架
Jquery	js框架	Ehcache	缓存技术
Jquery easy UI	js前端UI框架	Highcharts	JS报表框架
Ztree	js树形菜单插件	Apache shiro	权限管理框架

8.1.3 系统主页的设计

Java工程师大多不善于页面开发设计 ----- 完成门户站点页面开发

门户站点开发步骤： 设计工程师，设计站点页面整体效果 大图片 ---- 切图 ---- 前端页面开发工程师使用HTML+CSS 将页面开发出来 ---- java工程师将页面数据换为动态数据。

企业内部管理系统，对页面要求不高，公司不需要请页面设计工程师，java工程师开发看得过去的页面，使用前端开发框架 开发页面 jquery easyui。

Jquery Easy UI

layout 页面布局、accordion 折叠菜单、tabs 选项卡菜单

Ztree 树形菜单

Standard Data Tree 、Simple Data Tree

Easy-ui 各个目录

demo 框架使用案例

locale 国际化

plugins 框架包含控件包

src 框架包含源码

themes 样式css文件和icon图标

easyloader.js 框架核心加载器

jquery-1.8.0.min.js JQuery框架压缩文件

jquery.easyui.min.js JQueryEasyUI框架压缩文件 === easyloader + plugins/*

8.2 什么是 EasyUI?

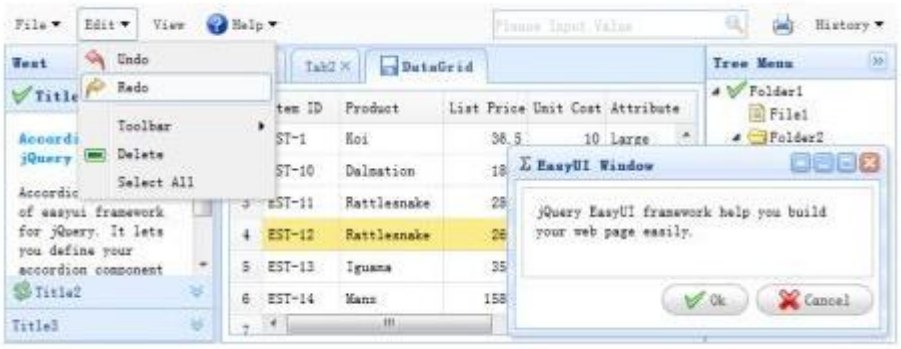
jQuery EasyUI是一组基于jQuery的UI插件集合体，而jQuery EasyUI的目标就是帮助web开发者更轻松的打造出功能丰富并且美观的UI界面。开发者不需要编写复杂的javascript，也不需要对css样式有深入的了解，开发者需要了解的只有一些简单的html标签。

8.2.1 市面上的常见前端框架

8.2.2easyui

easyui帮助你构建你的web应用更加容易。

它是一个基于jquery的插件，开发出来的一套轻量级的ui框架，非常小巧而且功能丰富。



官方网站是：http://www.jeasyui.com/

但是她有一个最大的问题就是代码只能找到以前的开源的版本，到了1.2以后的版本源代码都是经过混淆的，如果遇到问题修改起来会非常麻烦！不过一个比较大的优势是开源免费，并且界面做的还说的过去！

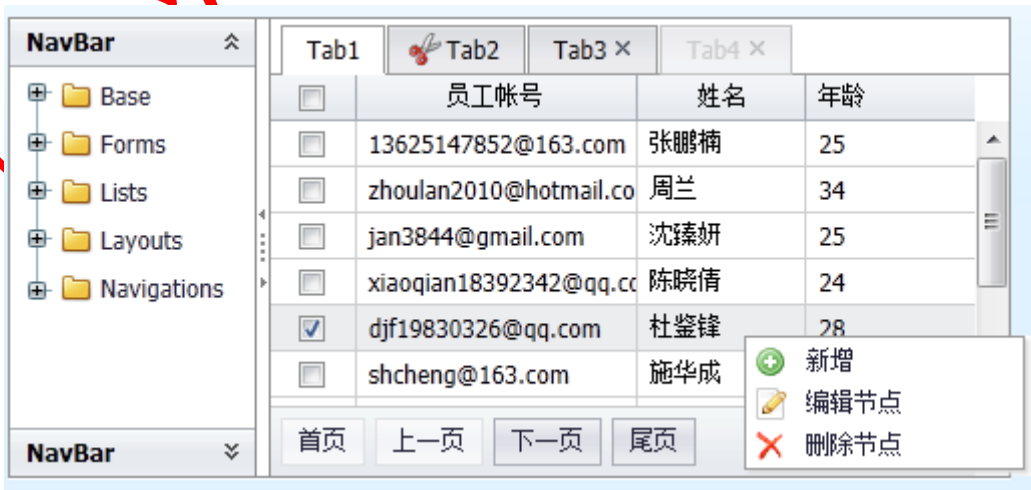
8.2.3MiniUI

又一个基于jquery的框架，开发的界面功能都很丰富。

jQuery MiniUI - 快速开发WebUI。

它能缩短开发时间，减少代码量，使开发者更专注于业务和服务端，轻松实现界面开发，带来绝佳的用户体验。

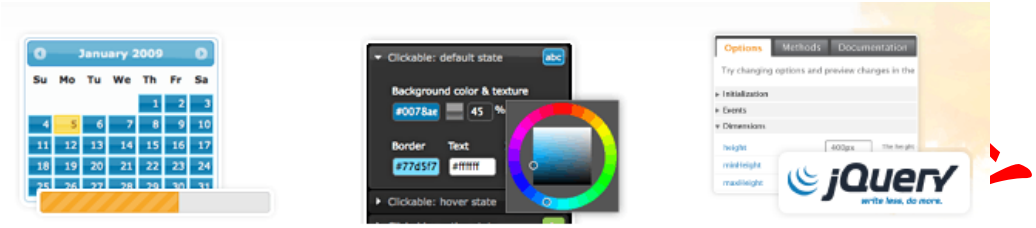
使用MiniUI，开发者可以快速创建Ajax无刷新、B/S快速录入数据、CRUD、Master-Detail、菜单工具栏、弹出面板、布局导航、数据验证、分页表格、树、树形表格等典型WEB应用系统界面。



界面做的挺不错，功能也挺丰富，但是有两个比较大的问题，一个是收费，一个是没有源码，说白了，不开源！基于这个开发如果想对功能做扩展就需要找他们的团队进行升级！

8.2.4jQuery UI

jQuery UI 是一套 jQuery 的页面 UI 插件，包含很多种常用的页面空间，例如 Tabs（如本站首页右上角部分） 、拉帘效果（本站首页左上角）、对话框、拖放效果、日期选择、颜色选择、数据排序、窗体大小调整等等非常多的内容。



功能非常全面，界面也挺漂亮的，可以整体使用，也可以分开使用其中的几个模块，免费开源！

如何使用 jQuery EasyUI 框架创建应用。

1. 直接在 HTML 声明组件。

```
<div class="easyui-dialog" style="width:400px;height:200px"
    data-options="title:'My Dialog',collapsible:true,iconCls:'icon-ok',onOpen:function()
{}">
        dialog content.
</div>
```

2. 编写 JavaScript 代码来创建组件。

```
<input id="cc" style="width:200px" />

$('#cc').combobox({
        url: ...,
        required: true,
        valueField: 'id',
        textField: 'text'
});
```

然后你就可以使用 EasyUI 提供的各种便捷的应用。

8.3Activity 工作流

8.3.1 什么是工作流

以请假为例，现在大多数公司的请假流程是这样的

员工打电话（或网聊）向上级提出请假申请——上级口头同意——上级将请假记录下来——月底将请假记录上交公司——公司将请假录入电脑

采用工作流技术的公司的请假流程是这样的

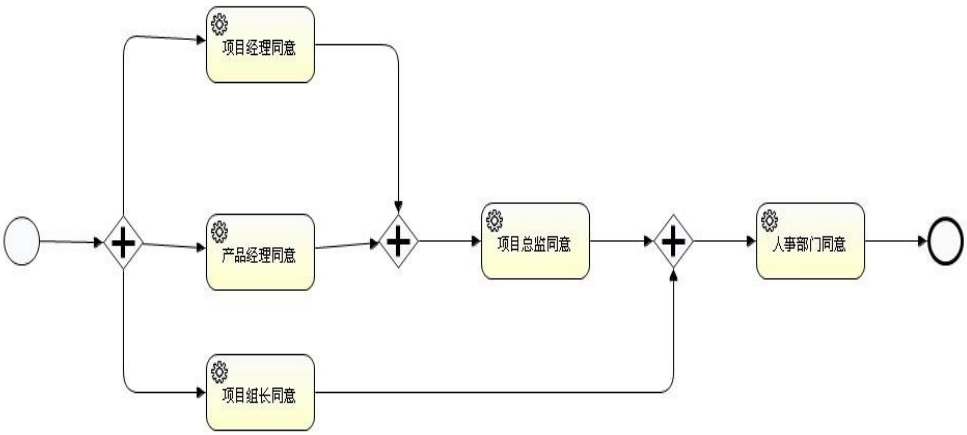
员工使用账户登录系统——点击请假——上级登录系统点击允许

就这样，一个请假流程就结束了

有人会问，那上级不用向公司提交请假记录？公司不用将记录录入电脑？答案是，用的。但是这一切的工作都会在上级点击允许后自动运行！

这就是工作流技术。

Georgakopoulos给出的工作流定义是：工作流是将一组任务组织起来以完成某个经营过程：定义了任务的触发顺序和触发条件，每个任务可以由一个或多个软件系统完成，也可以由一个或一组人完成，还可以由一个或多个人与软件系统协作完



8.3.2 工作流技术的优点

从上面的例子，很容易看出

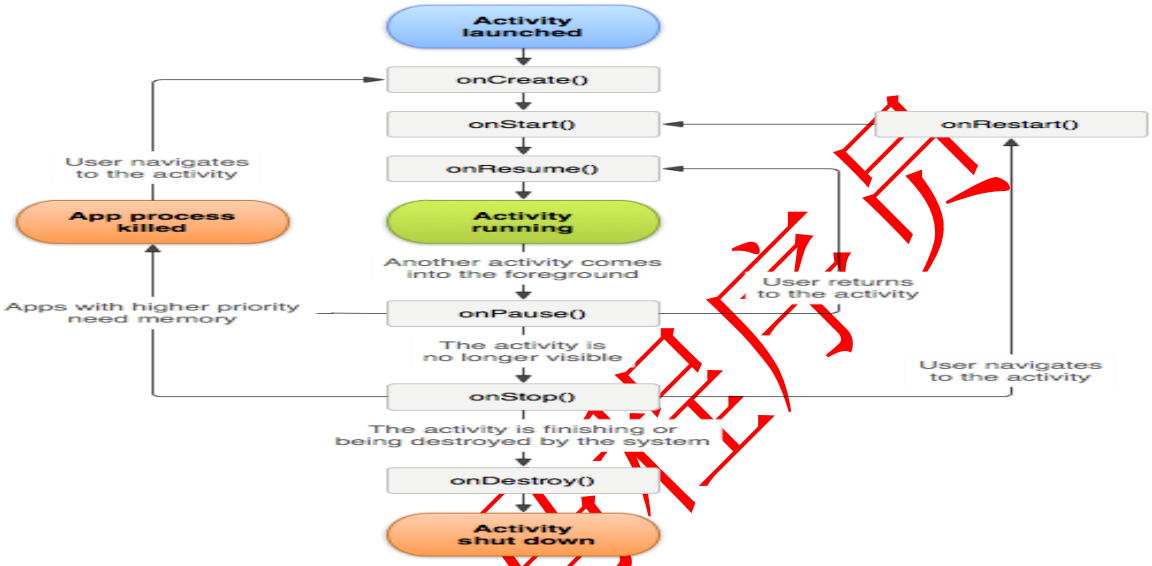
工作流系统，实现了工作流程的自动化，提高了企业运营效率、改善企业资源利用、提高企业运作的灵活性和适应性、提高量化考核业务处理的效率、减少浪费（时间就是金钱）。

而手工处理工作流程，一方面无法对整个流程状况进行有效跟踪、了解，另一方面难免会出现人为的失误和时间上的延时导致效率低下，特别是无法进行量化统计，不利于查询、报表及绩效评估。

8.3.3 生命周期

除了我们自行启动（start）或者结束（finish）一个Activity，我们并不能直接控制一个Activity 的生命状态，我们只能通过实现Activity生命状态的表现——即回调方法来达到管理Activity生命周期的变化。

具体如下图所示：



一个Activity本质上只有三种状态：

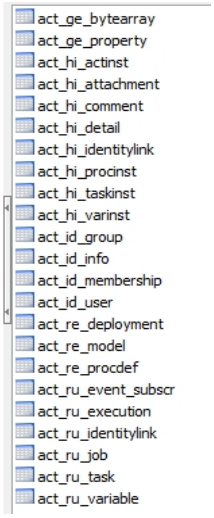
Resumed（运行）、Paused（暂停）、Stopped（停止），因为从Activity被创建之后，它只可能在这三种状态保持长久的停留，其他的回调方法结束后的状态都只能称之为过渡状态。比如进入到onStart方法后，执行完该方法，会立即进入到OnResume方法。（这里所说的状态都是指对应的某个方法返回之后）

即使一个Activity进入到Paused或者Stopped方法，它仍然是存在的，被保存在任务返回堆栈中。它仍然保持着自身的所有实例和状态，所以根本不用担心它在返回到onResume方法时，实例会变为null，或者控件的事件监听不了（我以前就担心过这个问题）。唯一需要考虑的就是，系统在内存不足的情况下，杀死在Paused或者Stopped状态下的Activity。

当一个Activity在Resumed状态下，它是不会因内存不够而被系统直接杀死（在极端的情况下也有可能被杀死，但是一般不会考虑这种情况）。只有进入Paused或者Stopped状态才会，而且可能根本就不会去调用onStop()和onDestory()方法，所以onPause()方法是我们最大程度上保证Activity在销毁之前能够执行到的方法。因此，如果你的某个Activity需要保存某些数据到数据库，您应该在onPause()里编写持久化数据的代码。但要注意，你应该选择哪些信息必须保留在onPause()，因为这个方法任何阻塞程序都会阻止过渡到下一个Activity，这样给用户体验就感觉十分缓慢。

8.3.3 23 张表

不同的表存放不同方面的数据，有流程定义表、任务结点表、流程变量表、任务历史表等等。



8.4Apache POI 报表技术

8.5.1 简介：

Apache POI 是创建和维护操作各种符合Office Open XML（OOXML）标准和微软的OLE 2复合文档格式（OLE2）的Java API。用它可以使用Java读取和创建,修改MS Excel文件.而且,还可以使用Java读取和创建MS Word和MSPowerPoint文件。Apache POI 提供Java操作Excel解决方案。

8.5.2 优点:

效率高
支持公式，宏，一些企业应用上会非常实用
能够修饰单元格属性
支持字体、数字、日期操作

8.5.3 缺点:

不成熟,代码不能跨平台，貌似不少同行在使用工程中还碰到让人郁闷的BUG（最近的项目中也是遇到了一些bug，不过目前没有查出来是代码的问题还是POI的问题，总之问题很诡异，数据替代参数总有失败的。关于不能跨平

台这一说，我也没有试验过，不过Java不是跨平台吗？POI是JAVA的一个组件，怎么就不能跨平台了呢，总之这些问题还需要在以后的项目中多多实践，才能比较出区别之处。）

POI实现模板打印

我们系统根据用户习惯，采用excel方式，作为业务输出打印。打印时POI是有问题，打印时代码量非常大，样式代码非常多，（常保存，样式和字体对象创建过多）。打印时非常难控制。我进行改造，我使用模板来开发，在模板中定义列宽（POI操作列宽时有BUG，计算非常繁琐），还可以定义静态文件，例如标题部分等静态文字的内容，打印纸质方向，打印页面标题，页眉页脚，都可以在模板中设置。首先无需记忆复杂特殊api。设置字体、样式也可以直接在模板中定义。打印时先读取模板，只处理业务数据的打印即可。

POI海量数据导出系统中备份数据和恢复数据。

操作excle主要有jxl和poi两种方式，jxl在处理数据时比早期的poi快。（poi早期对象，处理时都是将加工的数据存放在内存中，如果数据量很多，很容易造成堆溢出，同时它占用CPU和大量内存，导致其他业务也无法正常完成。）poi在新版本中改善这个性能瓶颈。对大数据量的导出做了优化。

使用ooxml技术，使用SXSSF对象，当数据创建到指定数量时，自动写缓存，将它内容输出到临时文件中。这个临时文件是一个xml，相比内存中对象的结构非常简单。只保留数据的信息。保存的数据量也非常少。这样就可以形成大数量的导出。例如：杰信项目中，购销合同业务，它有很多历史信息，积累了很多年，数据量达到近百万。导致系统变慢，我们采用poi导出数据备份到excel文件中。将这部分历史信息从当前表删除。在打印中就可以直接实现。（excel单sheet可以支持1048576）。

8.5HibernateSearch 全文搜索

Hibernate Search是在apache Lucene的基础上建立的主要用于Hibernate的持久化模型的全文检索工具。像Lucene这样的检索引擎能够给我们的项目在进行检索的时候带来非常高的效率，但是它们在基本对象的检索时会有一些问题，比如不能实现检索内容跟实体的转换，Hibernate Search正是在这样的情况下发展起来的，基于对象的检索引擎，能够很方便的将检索出来的内容转换为具体的实体对象。此外Hibernate Search能够根据需要进行同步或异步的索引更新。

Hibernate Search 项目的主要特性包含以下几个方面：

1.Lucene集成——作为强大高效的检索引擎，Lucene的美名早已久经考验了；

2.数据的自动插入和更新——当一个对象通过Hibernate添加或更新时，索引也会相应进行透明的更新；

3.支持众多复杂的搜索方式——可快速的使用通配符进行搜索，以及多关键词全文检索（multi-word text searches）和近似或同义词搜索（approximation/synonym searches），或根据相关性排列搜索结果；

4.搜索集群（Search Clustering）——Hibernate Search提供了内建搜索集群解决方案，其中包括一个基于JMS的异步查询和索引系统；

5.对Lucene API接口的直接调用——如果用户打算处理某些特别复杂的问题，可以在查询中直接使用Lucene提供的API接口；

6.对Lucene的自动管理——Hibernate Search可以管理并优化Lucene的索引，并且非常高效地使用Lucene的API接口。

ibernate Search 相关的 Annotation 主要有三个：

@Indexed 标识需要进行索引的对象， 属性 ： index 指定索引文件的路径

@DocumentId 用于标示实体类中的唯一的属性保存在索引文件中，是当进行全文检索时可以这个唯一的属性来区分索引中其他实体对象，一般使用实体类中的主键属性

@Field 标注在类的get属性上，标识一个索引的Field
属性 ：index 指定是否索引，与Lucene相同
store 指定是否索引，与Lucene相同
name 指定Field的name，默认为类属性的名称
analyzer 指定分析器
另外@IndexedEmbedded 与 @ContainedIn 用于关联类之间的索引

@IndexedEmbedded有两个属性，一个prefix指定关联的前缀，一个depth指定关联的深度

8.5.1 功能

Hibernate Search的带来的强大的全文搜索引擎的持久性域模型相结合，Hibernate核心的Apache Lucene搜索引擎的功能。

如Apache Lucene的全文搜索引擎非常强大的技术，高效的全文检索功能添加到应用程序。然而，Lucene的处理对象的域模型时， 遭受数不匹配。除其他事项外指标必须跟上日期和索引结构和域模型以及查询不匹配，这样才能避免之间的不匹配。Hibernate Search 的解决了这些缺点。索引你的域模型的一些注释的帮助，照顾数据库/索引同步，并带回常规管理的对象从自由文本查询。因此，它解决了：[4]

结构不匹配：Hibernate Search的照顾对象/索引翻译

重复不匹配：Hibernate Search的管理指标，使你的数据库的同步变化，并优化索引的访问透明

API不匹配：Hibernate搜索，让您查询的索引和检索管理对象的任何Hibernate查询会做定期

即使Hibernate Search的引擎盖下使用的Apache Lucene™，你可以随时退回到本机的Lucene的API，如有需要。

根据应用的需要，Hibernate Search的工作在非集群和集群模式，提供同步和异步索引更新，让您作出积极响应，吞吐量和索引更新时间之间做出选择。

最后但并非最不重要的一点是，Hibernate Search的可以完美地与所有的传统休眠模式，特别是使用Seam的谈话模式。

8.5.2 特点

Hibernate Search主要有以下功能特点：

1，功能强大，配置简单 - 配置只需要修改persistence.xml（JPA），hibernate.cfg.xml（Hibernate）

2，支持Hibernate，以及EJB3 JPA标准应用

3，集成全文搜索引擎Lucene - Lucene是Apache项目组下的一个功能强大的全文搜索引擎项目

4，可以简单透明索引查询过的数据

5，支持复杂检索 - 支持Wild Card（诸如*, ?等通配符号），多关键字，模糊查询，排序等

6，支持Clustering

7，支持直接访问Lucene API

8，对Lucene索引，API的高效管理。

8.6ApacheShiro 权限控制

8.6.1 首先什么是 shiro?

Shiro 是 JAVA 世界中新近出现的权限框架，较之 JAAS 和 Spring Security，Shiro 在保持强大功能的同时，还在简单性和灵活性方面拥有巨大优势。

shiro是apache下面的一个开源项目，下面是其网站上对其的一段说明：

Apache Shiro is a powerful and easy-to-use Java security framework that performs authentication, authorization, cryptography, and session management. With Shiro’s easy-to-understand API, you can quickly and easily secure any application – from the smallest mobile applications to the largest web and enterprise applications.

弱弱的翻译一下：apache shiro是一个强大且易于使用的java安全框架，使用它可以进行

1：认证

2：鉴权

3：加密

4：会话管理

通过shiro简单易懂的api，可以简单快速的为任何应用程序提供安全保护。

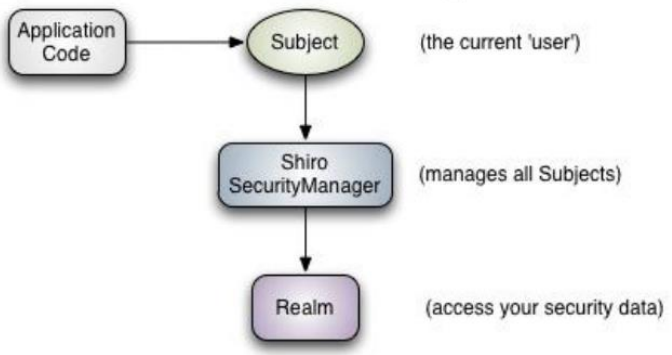
8.6.2apache shiro 能做什么？

支持认证跨一个或多个数据源（LDAP，JDBC，kerberos身份等）
执行授权，基于角色的细粒度的权限控制。
增强的缓存的支持。
支持web或者非web环境，可以在任何单点登录(SSO)或集群分布式会话中使用。
主要功能是：认证，授权，会话管理和加密。

8.6.3Shiro 主要有四个组件

• SecurityManager
典型的 Facade，Shiro 通过它对外提供安全管理的各种服务。
• Authenticator
对“Who are you ？”进行核实。通常涉及用户名和密码。
这 个组件负责收集 principals 和 credentials，并将它们提交给应用系统。如果提交的 credentials 跟应用系统中提供的 credentials 吻合，就能够继续访问，否则需要重新提交 principals 和 credentials， 或者直接终止访问。
• Authorizer
身份份验证通过后，由这个组件对登录人员进行访问控制的筛查，比如“who can do what”， 或者“who can do which actions”。 Shiro 采用“基于 Realm”的方法，即用户（又称 Subject）、用户组、角 色和 permission 的聚合体。
• Session Manager
这个组件保证了异构客户端的访问，配置简单。它是基于 POJO/J2SE 的，不跟任何的客户端或者协议绑定。

8.6.4Shiro 运行原理



Application Code:应用程序代码，就是我们自己的编码，如果在程序中需要进行权限控制，需要调用Subject的API.

Subject:主体，代表的了当前用户。所有的Subject都绑定到SecurityManager，与Subject的所有交互都会委托给SecurityManager,可以将Subject当成一个门面，而真正执行者是SecurityManager.

SecurityManage:安全管理器，所有与安全有关的操作都会与SecurityManager交互，并且它管理所有的Subject.

Realm:域 shiro是从Realm来获取安全数据（用户，角色，权限）。就是说 SecurityManager

要验证用户身份， 那么它需要从 Realm 获取相应的用户进行比较以确定用户身份是否合法；也需要从 Realm 得到用户相应的角色/权限进行验证用户是否能进行操作； 可以把 Realm 看成 DataSource，即安全数据源。

8.6.5Shiro 为程序共提供了四种权限控制方式

url级别权限控制

方法注解权限控制

代码级别权限控制

页面标签权限控制

8.7WebService

WebService是一种跨编程语言和跨操作系统平台的远程调用技术。所谓跨编程语言和跨操作平台，就是说服务端程序采用java编写，客户端程序则可以采用其他编程语言编写，反之亦然！跨操作系统平台则是指服务端程序和客户端程序可以在不同的操作系统上

8.7.1 常见远程调用技术

WebService(传统的—大Web Service)
SOCKET
Hessian
http-restful
上述技术统称 Web Service

8.7.2 各种远程调用技术效率对比

SOCKET》Hessian》http-restful》WebService

8.7.3WebService

优点：早起非常流行，大公司的弄的，标准的，很容易跨平台，跨语言。设计的时候在网络上传输。

缺点：额外使用soap协议（xml包装-消息太大---解析成本），效率不高，大公司绑架，在开源的时代，很多人抵制它。Webservice　没有集群的支持。--集群化思想。

--阿里 duble—支持集群

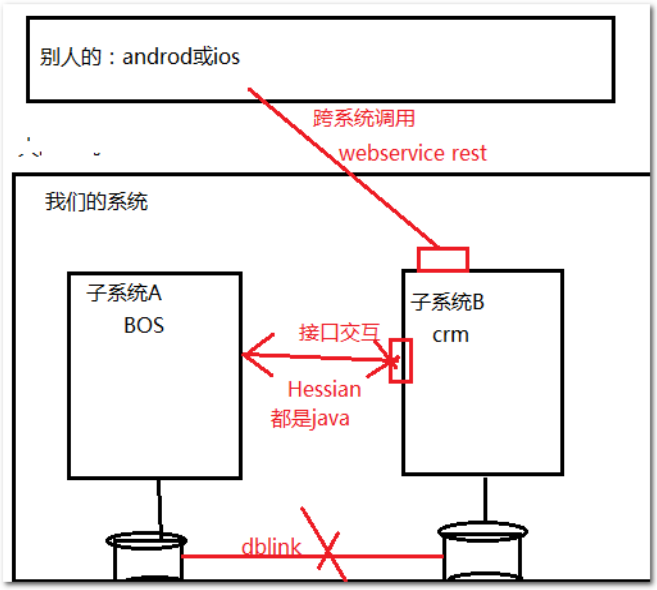
8.7.4Hessian：

优点：二进制传输，效率超高
缺点：跨平台还是有点小问题。

8.7.5 选型：

如果跨平台跨语言，不太追求效率，可以使用webservice。
如果跨平台跨语言，追求效率，http-restful.
同平台（都是java），追求效率：Hessian

8.7.6 基础架构



8.8EhCache

8.8.1EhCache 是什么

EhCache 是一个纯Java的进程内缓存框架，具有快速、精干等特点，是Hibernate中默认的CacheProvider。

8.8.2 主要的特性有

1. 快速
2. 简单
3. 多种缓存策略
4. 缓存数据有两级：内存和磁盘，因此无需担心容量问题
5. 缓存数据会在虚拟机重启的过程中写入磁盘
6. 可以通过RMI、可插入API等方式进行分布式缓存
7. 具有缓存和缓存管理器的侦听接口
8. 支持多缓存管理器实例，以及一个实例的多个缓存区域
9. 提供Hibernate的缓存实现

8.8.3EhCache 的使用

Cache的配置很灵活，官方提供的Cache配置方式有好几种。你可以通过声明配置、在xml中配置、在程序里配置或者调用构造方法时传入不同的参数。

8.9Highcharts

8.9.1 什么是 Highcharts

Highcharts是一个非常流行，界面美观的纯Javascript图表库。它主要包括两个部分：Highcharts和Highstock。Highcharts可以为您的网站或Web应用程序提供直观，互动式的图表。目前支持线，样条，面积，areaspline，柱形图，条形图，饼图和散点图类型。Highstock可以为您方便地建立股票或一般的时间轴图表。它包括先进的导航选项，预设的日期范围，日期选择器，滚动和平移等等。

8.10BOS 中的完成的功能模块

三个业务功能模块
基础数据模块
派取模块
中转流程管理
两个系统功能模块
用户管理
权限管理

基础设置模块： 管理物流信息基础元素信息 （快递员、配送区域、配送时间、排班 …）
取派模块： 客户要进行快递，系统进行业务受理、系统分单到快递员、快递员取件、打包、计费
中转模块： 管理货物在配送路程中间数据
用户权限管理： 通用权限管理系统 Apache Shiro

8.10.1 取派管理

8.10.1.1 添加取派员

我们可以在管理界面点击-取派员设置-增加，可以在弹出的窗口添加新取派员信息。通过Validatebox插件来校验表单输入的字段。对取派员编号和姓名进行非空校验，以及手机等其他信息的校验。然后通过点击保存，提交表单。

8.10.1.2 分页列表查询取派员

我们使用Datagrid+Spring Data来实现分页查询，当页面加载以后初始化datagrid的时候，自动发起一次请求，请求参数为当前页码和每页的最大记录page和rows。然后服务器获取到请求以后，将接受到的参数传入pageable接口中，然后经过DAO层的处理返回page的包装对象，从该对象中获取Datagrid所需要的json格式数据，返回给客户端，自动计算和展现分页表格。

8.10.1.3 批量作废和修改

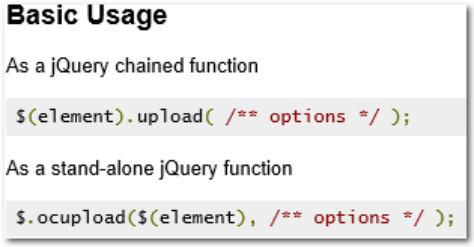
针对取派员的作废和修改,我们在删除作废取派员的时候,不会真正的删除取派员信息,因为有一些表关联的问题,所以当时考虑使用逻辑删除.勾选需要作废的取派员,点击作废之后,根据后台接收到的ID,将作废的取派员信息的deltag设置为1。同样的还有一个还原功能，反之将deltag设置为0即可。

修改的功能，给页面设置了弹窗事件Datagrid，对需要修改的某行数据进行双击操作，在弹出的窗口会有回显的数据，修改完成之后点击保存。然后当时发现一个问题，在点击添加取派员的时候发现会回显刚刚修改的数据，后来分析是因为，添加和修改用的弹窗是一个弹出层窗口，然后在弹窗JS代码里加上onClose:clearStaffForm来实现关闭弹出窗口的时候清除表单数据。

8.10.2 区域设置

8.10.2.1 批量导入

这个区域完成的是是将Excel直接导入到系统，完成一些数据的批量、快速添加。这里就用到2个技术，一个是上传年数据文件和解析excel。经过分析客户端在上传的时候，先浏览文件，将文件路径存入表单，然后再通过form表单提交，将其提交到服务端。考虑到为提升用户体验（UE），现在的系统流行一键上传，即用户点击了上传按钮或链接后，直接选择要上传的文件，点击确定就开始上传了，无需额外提交表单。而且页码无刷新。最后使用jQuery ocupload插件实现一键上传。参考官方文档，调用一键上传有两种方法，我们

这里选的第一种 

。跟着官方DEMO实现功能以后我对这块进行了一个小优化，限制文件上传的类型。添加文件格式校验，要求只能上传扩展名为.xls或.xlsx的文件。最后服务器通过Struts2 FileUpload Interceptor，给工程添加commons.io包，在Struts配置里配置临时目录以及最大上传大小等等，完成上传。

对于excel解析并入库，我们采用的市面上Apache POI来解决。通过maven引入poi，编写解析代码 当时参考资料 知道poi的解析基本流程为打开WorkBook工作簿文件—》找到Sheet工作表—》遍历读取Rows行—》读取行中的cell单元格。当时在完成这块的时候发现要更改Region实体类的主键策略为手动（因为导入的数据第一列自带主键）。当初测试的时候发现有的excel表格出现bug数据不完善乱码等现象，最后通过更改excel的单元格格式为文本格式就可以解决了。

8.10.3 分区管理

8.10.3.1 添加分区

这块分区是对区域细分，为后续指定配送区域，提供基础数据。在添加弹窗里用到 easyUI combobox 插件来实现下拉列表框。在页码加载后，自动向服务器发起 Ajax 请求获取下拉列表的 json 数组数据，页码加载后，自动向服务器发起 Ajax 请求获取下拉列表的 json 数组数据。

然后还有一个针对区域的一个自动补全的效果，类似于 jQueryUI 的 AutoComplete 效果。在 combobox 组件里具有本地加载自动补全的效果。但是仅限于本地数据，也就是说本地加载了所有数据才会有补全效果，在这里我们是把默认的设置改为远程数据加载自动补全，当修改分区的下拉框数据的时候组件会自动向服务器发送参数 q，值就是输入的值。服务器获取组件传过来的参数 q，根据参数 q 来自动匹配省市区。

8.10.3.2 列表分页条件查询

分区管理的分页和之前的分页有稍许不同以前是分页+条件查询。现在是使用 Datagrid 条件分页来完成。在查询的时候 form 表单里用户输入的查询条件会转换成 json 对象，缓存在 datagrid 内部，然后 datagrid 在向服务器发送请求时，会自动携带查询条件，查询条件会一直在 datagrid 中实现带有条件的分页查询。整个将业务条件和分页条件都交给 Datagrid，业务条件

缓存到 Datagrid 中。datagrid 通过使用 load 方法加载和缓存业务参数，在参数发送请求的时候会自动发送给服务端。

服务端通过 springdata 实现 QBC 方式查询，需要 2 个对象 Specification（条件规范）和 Pageable（分页规范）。设置多表关联条件对象，有省份条件、城市条件、区域条件，最后将条件对象汇总调用业务层，最后将数据转换成 json 需要的格式返回给客户端。

8.10.3.4 数据导出

针对查询后的数据结果进行 excel 导出,备份出来.当然我们导出的数据不仅是本页的,而是符合条件的全部数据。对导出按钮设置 form 表单提交，在服务端使用 POI 将数据转换生成 excel（.XLS 格式）根据 excel 格式创建工作簿、工作表、然后把表单里的数据写入没一个分区对应的一行里，这一块步骤会比较麻烦，大部分也是参考文档来编写。然后把文件流写入到响应中，设置好 Content-Type 和 Content-Disposition2 个参数便于游览器识别附件，最后获取文件的名字和下载方式，把他放入到响应头中。简单的说咱们就是通过客户端的同步方式提交请求，在服务端使用 stream 类型的结果集,这个结果集有封装一些自己的操作,比如输出流放入响应中,设置好 Type 和 Disposition。最后把文件下载放入到响应头中即可。

8.10.4 定区管理

8.10.4.1 定区添加

定区是针对取派员固定配送区域的管理，该区域它是动态可调整的。定区是分区的集合，定区里需要指定多名取派员来负责。在定区添加里，主要是关联取派员和分区，也就是这个定区该有哪个或者哪几个取派员来负责，该定区包含哪几个分区。在添加定区里会有一个取派员的下拉列表，列表中的取派员必须是未作废 deltag 为 0 的。在该页面里会显示未分配的分区表格，用来关联分区。最后保存定区数据,在分区中使用外键关联定区。在开发的时候当时发现了一个问题，发现提交的是表单复选框的字段，但请求参数中定区编号和分区编号的参数名称会发生冲突，分区编号提交的是也为 id 和定区编号冲突。

最后是修改了 datagrid 的复选框 field 的名字，和服务器返回的 json 的方法字段解决了问题。

8.10.4.2 定区关联客户

我们系统中划分了定区，不同的取派员被分配管理着不同的定区，每个定区中又包含多个不同分区，客户居住在某个区域的分区中，而分区又属于定区，因此，当客户下单后，会自动关联分配对应的取派员，方便了系统后期的查询和管理、自动下单等。我们系统交互必须远程访问 CRM 系统来获取客户信息，然后关联定区。在 CRM 中建立一张客户表来存放客户信息，BOS 再读取 CRM 中的客户数据。我们采用了 WebService 接口技术来连接---SOA 的思想。参考 spring 的规范对 CXF 与 Spring 进行整合。引入 CXF，参考 CXF 官方文档配置 CXF 服务、核心控制器等等，接着设计他的接口数据类型，当时考虑用的字符串，因为不同平台的字符串的设计都差不多，然后用 JSON 格式来传递。因为 json 任何平台都能识别和解析。

设计 json 的格式

请求的数据：

{‘opertype’:’101’,’param’:{‘参数 1’:’参数 1 的值’,……}}

opertype’：操作类型：你要做什么，比如，你要查询未关联的客户列表，双方约定 101，如果是查询已经关联的客户列表：102，

如果某个业务需要传参，那么 param 该参数就有效了。比如，根据客户编号，查询客户信息：{‘opertype’:’301’,’param’:{‘参数 1’:’参数 1 的值’,……}}

响应的数据：

{‘status’:1,’data’:[{},{},{}]}

Status：状态码，作用告诉客户端，你的请求的情况，1 代表正常，向下解析数据了,如果 0，说明未知异常，如果 2，您的参数格式不正确。

8.10.5 业务受理

8.10.5.1 业务通知单创建

记录客户下单的请求生成业务通知单，自动分单，生成工单。这一块我们做了 2 个单子：1.业务通知单：客户下单的信息记录；2 工单：取派员取件任务单；

当客服（业务受理员）输入客户来电后，输入框离焦后，发起 Ajax 请求到后台服务器，后台通过 WebService 调用 CRM 接口，来获取当前客户信息，返回页面，显示填充到业务通知单页面。点击“新单”，将表单数据提交到服务器。页面给按钮添加 click 事件，校验并提交 Form 表单

自动分单是在生成业务通知单的时候，系统会自己通过一定的规则，匹配到对应的派送员，将通知单绑定某取派员，并自动生成工单。我们是如何匹配到取派员的呢，根据客户的地址完全匹配（这个地址以前下过单）

自动匹配 CRM 系统中客户表中的地址，匹配成功后客户表中可以找到属于的定区，定区里之前保存着关联的取派员。或者根据客户地址中的关键字来匹配，获取客户信息中具体区域 RegionId，在分表中找到高区域的分区列表，便利分区去使用地址关键字来匹配客户的地址，然后找到定区---定区里关联的取派员。如果 2 种方法都没法匹配的话，转到人工来进行手动分单。

8.10.5.2 业务受理后的其他功能操作

工单管理等一些 CRUD 操作；

✓ 追单：当客户致电催收，客服人员记录客户的追单请求，咨询受理员查询出客户历史订单信息后，可以进行追单查询操作，修改工单类

型为追单

✓ 改单：在未上门收件前，客户需要更改收件信息，致电客服，客服人员记录客户的改单请求。

✓ 销单：在未上门收件前，客户取消下单请求，致电客服，客服人员记录客户的销单请求。

✓ 查台转单：分单分错了，修改业务通知单和工单关联取派员

✓ 人工调度：手动派单，显示通知单，手动指定取派员，后台生成工单。

8.10.5.3 工单快速录入

这块提供操作人员在货量大单子多的时候简介快速录入工作单的途径，快速录入中的信息主要是满足配载而设置的功能界面。只录入必要的信息，在其他闲暇时间对工单的其他信息进行补全、完善。通过 Datagrid 来对行内编辑，使用方法 endEdit 来保存、canceEdit 取消编辑等等。当编辑结束以后发起 ajax 请求，把编辑好的数据发送给服务器保存。

8.10.5.4 工单索引的创建和搜索

在创建好工单之后，我们也有一个工单快速录入的一个列表查询，给列表增加一个搜索功能。一个分页条件查询的列表。在前台给 datagrid 的 url 写一个请求 action，返回 json 就可以了。在条件查询的时候，要调用 datagri 的 load 方法（发请求），将业务条件缓存到 datagri 的中，datagrid 在发起请求，自动将分页和业务条件参数都发送给后台。

获取到这些参数，组装一个分页对象 pageable，条件规范对象 specification 对象，调用 findAll(pageable,specification)--->page<T>,重新组装数据，交给 struts 的 json 插件就可以了。后来我们发现这样查询大数据量的时候，导致了一个效率很低很低的情况，就引入了全文搜索的技术来解决这个问题。通过 hibernate search 创建 lucene 全文索引，然后使用了一个

开源的私服 IK 分词器项目。在 hibernate 的配置里面添加索引文件的存放目录。因为 Hibernate search 是基于 Lucence 的，所以他会自动创建索引、修改索引、删除索引，我们不需要手动编写。在保存数据的时候，Hibernate Search，会在向数据库保存数据（完整）的同时，在索引库中也保存索引数据（分词后的数据，部分数据）。后来使用了一个工具 Luke 来查询索引文件，以及确认索引的无误。

在搜索框的选择，我们使用的 searchbox 插件来实现的搜索框。这个 searchbox 会提示用户输入要查找的值。它可以把一个菜单,允许用户选择不同的搜索类别。搜索行动时将执行用户按回车键或点击搜索按钮在右边的组件。

8.10.6 权限管理

我们系统权限相关的操作，都是由超管完成的，整个系统只有一个超管，普通的管理员若干，都是可以通过超管来授权普通管理员。根据权限的不同可以操作不同的管理模块。

8.10.6.1 角色管理

8.10.6.2 添加角色并关联权限

在超级管理员角色下可以添加新角色并 sssss，针对不同的权限可以授权不同的模块，在添加角色弹窗里有一个可勾选的授权树状图。在超管勾选好授权树候前端会处理获取选中的节点，并且将其拼接为逗号分割字符串放入隐藏域，然后提交表单。服务器接收到请求后保存该角色并且关联对应的权限。

8.10.6.3 添加用户并授予角色

我们系统的用户有两种，一种是登陆用的；一种是业务用的。在用户添加界面的时候就有角色选择，在保存用户的同时就保存角色。在前端页面有授予角色的列表通过 ajax 发生请求，一个角色对一个 checkbox，后端在保存用户的时候同时关联权限。

8.10.6.4 系统菜单的生成

系统菜单是动态的菜单，根据用户的权限来显示不同的有权限的菜单。前端获取当前登录的用户通过 session 中的用户 id,在后端判断用户的性质，如果是超管就拥有所有菜单，普通用户就是根据自己的权限显示菜单 generatemenu。

8.10.6.5 动态菜单的优化

首先是首页的刷新优化，在登录的首页重定向几个。然后是减少数据库查询的优化，直接在实体类中使用@JSON(serialize=fase)。最后是对每个用户菜单进行缓存优化，这里我们使用的是 ehcache 整合 spring 的缓存。首先注入 ehcache 的管理器，配置缓存的注解驱动，它会自动到 spring 的 bean 中寻找缓存相关注解，并使其有效。然后编写 ehcache 的配置文件，添加一个新的缓存区域，最后在方法上添加注解@Cacheable 来根据用户来缓存对应的菜单。