**自我介绍：**

您好：

我是xxx，从事java开发已经x年多了，之前在xxx有限公司工作。在这几年的工作中基本的主流框架比如ssh、ssm、dubbo+zookeeper、springboot都有搭建和使用过，为了方便项目的开发自己也会研究一些技术，如进行异构平台之间的数据传输的webservice，shiro安全框架，solr全文检索，redis缓存框架，此外还有一些简单的技术如poi多线程导入导出，highcharts报表，httpclient远程调用，powerdesigner建模，spring定时器，javamail，对数据库的设计也有一定的经验，常用的数据库如oracle、Mysql关系型数据库如mongodb和redis非关系型数据库。在项目中我比较关注性能优化以及安全方面的事项，比如sql优化，数据库优化分库分表读写分离，jvm优化，如通过nginx+tomcat负载均衡来提升服务器的性能而且在做项目的过程中还通过junit进行单元测试并和团队中的其他成员以及测试人员进行bug方面的交流，为了方便项目的打包部署，一般采用maven进行相关jar包的操作，关于web页面的设计，我还使用过像bootstrap、jquery easyui这些RIA富客户端技术。

平时呢还喜欢上一些技术网站比如csdn、开源中国、博客园等这些网站来提高自己的知识量和扩展知识面，毕竟时代在发展，科技在进步。要跟上时代的步伐，避免被淘汰啊！

这就是我个人的一个基本情况。

**项目介绍：**xxxx课堂

这个项目我上家公司从今年3月份开始做，当时这个项目采用两期来做。一期完成后台的运营系统管理；二期主要实现web前端网站的完成。

技术架构这块：当时框架用的是 springmvc+mybatis ,数据库做了一个mysql的主从同步，考虑到后期项目的并发量的增加，

项目中我们也用到了 dubbo+zookeeper 这种分布式技术。项目中web网站首页，二级页面，像这种偶尔会更新的模块都做了redis缓存处理。网站的详情页都做了静态化。 视屏里面的大量的留言，学习提问，回复等等都用mongdb 来存储，站内检索当时用的solr 来实现的.后台用户管理呢用的是shiro安全框架把权限控制到按钮级别，当时视频存储这块，一开始用的 FTP服务器存储，后来换成了阿里云视频服务器，测试服务器配了一个nginx+tomcat 集群配置。

**项目的架构：**

后台运营系统管理项目用 springMVC+mybatis+spring+maven+shiro+redis

前端采用easyui实现页面布局。

二期的的web前端网站

项目：springboot+dubbo+zookeeper+redis+mongodb+注解版mybatis(生产者，只提供service+dao)

项目中我们把网站页面+ation层作为消费者，service+dao 层作为生产者，消费者和生产者之间的调度用的是zookeeper 。

在上家公司，我们项目的初期的后台运营系统项目，给运营系统部署一台服务器、mysql部署一台服务器、对于存放的文件（视频和图片等）部署一台阿里云服务器。

在项目的中期，我们把web前端网站的项目加入进程，所以架构的部署为、前台消费者项目部署一台服务器、生产者和消费者之间的调度zookeeper部署两台服务器、Redis部署一台服务器、mongodb部署一台服务器、solr全文检索部署一台服务器。后期随着项目的逐渐成熟，我们在第二版上线的时候进行集群部署，对于dubbo的消费者计划多出一台服务器，配置nginx代理服务器来实现对两个消费者的调通、对于dubbo的生产者项目我们计划每个项目多出3-5台服务器来部署，redis集群部署（3-5台集群），实现哨兵机制、mongodb副本集部署、mysql服务器主从配置，一主两从，实现主从同步，读写分离。。

业务模块的话，你像这个客服运营中心主要涉及到：权限管理，讲师审核，机构审核，课程审核，网站用户管理，评论管理，热销和精选课程推广，合作企业管理，订单管理，订单审核，目录管理，课程目录管理，投诉日志，同时呢，项目中还有大量的报表需要做统计

Web视频网站主要涉及：课程分类展示，课程的点赞，收藏功能，热门课程的推广，我的足迹，讲师入驻，机构入驻，个人信息维护，余额充值，课程购买，积分兑换，投诉建议，精选课推荐，名师讲堂，企业合作，购物车模块，分销模块。

就整个项目的业务功能来讲，前台视频网站主要有：

首页展示模块：

目录展示、轮播图展示、名师大咖、微专业、免费排行、付费排行等子模块。

机构模块：

机构管理、公告管理、教务教学等子模块。

讲师模块：

个人管理、课程管理等子模块。

用户模块：

登陆注册、个人信息管理等子模块。

课程模块：

课程体系、课程目录、课程视频。

购买课程模块：

已订阅课程，已订阅付款课程，已订阅未付款课程

后台的运营系统主要有：

审核模块：

讲师入住审核、机构入住审核，课程视频的审核。

广告管理模块：

广告视频、广告图片。

推广管理模块：

首页轮播图推广、类型推广、微专业、名师大咖推广、免费推广、付费推广

订单管理模块：

订单查询，付款订单，未付款订单。

**负责模块：**

1.

参与SSM框架搭建，考虑到项目在线的高并发用的是ssm+dubbo+zookeeperdubbo+zookeeper搭建，为了实现负载均衡还进行了nginx+tomcat集群部署 当时为了降低主服务器压力，避免一台数据库服务器挂了出现问题还参与了mysql服务器部署主从同步的部署；  
2.用户管理模块：（用户的信息管理，POI导入导出 highchar图表统计比如：统计每个月的注册人数，男女比例，注册用户的年龄段等,shiro安全框架将权限控制到按钮级别

**权限的管理shiro：**

对于权限的管理，我们用到了shiro权限控制，代替拦截器将权限控制到了按钮级别。实现了不同系统人员的登陆后操作不同权限的目的。

为了在项目中使用Shiro我们在web.xml中配置了shiroFilter使其对所有请求都进行安全控制。

之后我们在shiro的配置文件中配置一个id为shiroFilter的bean，这点要保证和web.xml中filter的名字一致。

在进行权限管理时整体上来说分为认证和授权两大核心。

认证就是只有用户经过了登录页面的验证才能成为合法用户继而访问后台受保护的资源。

我们在shiro的配置文件中配置基于url路径的安全认证。对一些静态资源如js/css等，

包括登录以及验证码设置为匿名访问。对于其他的url路径设置为authc【认证】即只有经过正常

的登录并且验证成功后才能访问。

为了保证数据库中用户密码的安全性，我们对其密码进行了md5加密处理，又因为单纯的md5加密

比较容易破解，所以我们使用了密码+盐【salt】的方式，这里面的盐是由用户名+随机数构成的，并且

还进行了2次迭代即md5(md5(密码+盐)))这样就更增加了安全性。

在用户添加和重置用户密码时

调用PasswordHelper将加密后的密码以及生成的盐即salt都存储到数据库的用户表中。

Subject、SecurityManager、Realm。接下来，分别介绍这三者有何用：

Subject：表示“用户”，表示当前执行的用户。Subject 实例全部都绑定到了一个 SecurityManager 上，当和 Subject 交互时，它是委托给 SecurityManager 去执行的。

SecurityManager：Shiro 结构的心脏，协调它内部的安全组件（如登录，授权，数据源等）。当整个应用配置好了以后，大多数时候都是直接和 Subject 的 API 打交道。

Realm：数据源，也就是抽象意义上的 DAO 层。它负责和安全数据交互（比如存储在数据库的账号、密码，权限等信息），包括获取和验证。Shiro 支持多个 Realm，但是至少也要有一个。Shiro 自带了很多开箱即用的 Reams，比如支持 LDAP、关系数据库（JDBC）、INI 和 properties 文件等。但是很多时候我们都需要实现自己的 Ream 去完成获取数据和判断的功能。

****登录验证的过程****就是：在用户进行登录认证时我们会在登录方法中传入用户名和密码并创建一个Token,Subject 执行 login 方法，传入登录的「用户名」和「密码」，然后 SecurityManager 将这个 login 操作委托给内部的登录模块，登录模块就调用 Realm 去获取安全的「用户名」和「密码」，然后对比，一致则登录，不一致则登录失败。

）,  
3.视频管理(mongodb+freemarker存放视频详情页

首先导入springMVC-webmvc所需的JAR包。

配置web.xml（启动mvc框架）

<!-- 设置freeMarker的配置文件路径 -->

 <!-- 配置freeMarker的模板路径 -->

<!-- 配置freeMarker视图解析器 -->

新建文件：freemarker.properties，放conf目录下

auto\_import="/WEB-INF/templates/index.ftl" as do这个表示每个freemarker的视图页面都会自动引入这个ftl文件。里面定议的就是一些宏，如text文本框，各种form元素

1. 后台新建测试Control类：FreemarkerController

因为有大量的视频详情页，我们采用freemarker来生成静态页面。当时我们在生成静态页面的时候，我将当前数据的id+html作为生成静态页面的名称，

以此来防止静态化页面的相互覆盖，我们封装的freemarker模板工具类是传四个参数，第一个是模板的名称、第二个是模板的路径、第三个是传递的当前数据对象，放到map里、第四个是来指定生成的静态化页面的存储位置。这个生成的静态化的页面当时涉及到一个同步的问题，怎么保证数据库的数据更新的同时静态化页面的数据也更新呢？我们想到了spring的aop组件，在项目中做的时候，我们把所有在service层开启的事物的方法都统一命名。配置aop的切面规则，配置切面代理类，在代理类里面我们用httpcilent远程调用生成静态页面的方法

# **[断点续传原理与实现](http://blog.csdn.net/zhaowen25/article/details/41779221" \t "http://blog.csdn.net/zdy0_2004/article/details/_blank)**

   断点续传主要是使用http协议中range的属性来取得资源的部分内容，由于一般服务是不对外直接提供url访问的，一般都是通过id，在servlet中输出byte[]来实现，所以要想实现断点续传一般要自己实现服务端和客户端，客户端保持文件的下载或上传状态，（保存在本地或者数据库中）。再进行中断时保持中断状态，在进行续传时，首先读出文件的状态，然后设置range属性信息发送续传请求。服务器收到续传请求，读取range属性值，从文件中读取数据，发送到客户端。上述是基本的原理，上传下载原理相同。

 下面模拟两次从服务器中现在文件。

客户端：分多次取得部分文件内容，利用RandomAccessFile文件中读取数据，写到输出流中。服务端：获取分析range属性，从收入流读取内容字节流，利用RandomAccessFile输出到文件中。其中还解决了中文文件名乱码的问题，由于http协议中规定，当在网络中传输时，setHeader方法中的字符只能按ISO8859-1传输，所以这时候就要把Unicode字符转换成了ISO8859-1的编码传到客户端，客户端进行解码。否则会出现乱码。

）  
4.讲师和机构审核模块。（

讲师审核：（推荐用这个模块）

讲师审核模块，主要的作用是对提交成为讲师的人的审核，为什么要进行审核呢，因为不是所有的人都可以成为讲师，必须要是对这方面有相关经验的人，才可以申请，比如：我现在想成为语文的讲师，我必须要有语文的讲师资格证书

或者其他的可以证明你能力的证书，等等。此外，当你在前台页面填写各种信息，进行提交的时候，后台，会去调阿里云的oss文件服务器，去将你的身份证图片，你的资质的图片，上传到阿里云的服务器上，同时，返回一串地址字符串，可以存入数据库。此外

后台还会调用webservice 去调用公安部的第三方接口，支付宝的芝麻信用等等，查询该用户是否是在逃人员，信用记录不好的人员，或者是其他会发布危险信息的人员。然后，将申请者的信息，写入到数据库中，然后，将其状态置为WAIT\_AUDIT.

后台根据状态，将这些数据查出，展示出来，后台会专门的设置审核人员，人工进行审核，比如，查看该用户的些基本信息是否正确。看看该用户之前是否有不良的订单记录，等等。如果，一切ok。就将该申请的人的状态置为审核成功AUDIT\_SUCESS.

然后，该讲师就有了，发布课程，设置价格等等权限了。

机构申请的话主要是进行企业入驻申请，都必须填写正确的信息，如身份证，绑定银行卡帐号）

1. 课程推广（

使用redis的list数据类型存放需要展示的推广ID，hash数据类型存放推广详细信息。然后前台的web网站可以可以去redis里取出来，在页面展示；每增加一条推广展示，在 列表“home:ad:list” 末尾使用 “RPUSH” 命令增加一个值（例如：ID值为5的广告，值为“ad:hash:5”）。同时使用 “HMSET ”命令存储一条 “hash ”类型的广告数据。每删除一条广告信息，使用“ LREM”命令从列表“home:ad:list” 中删除对应的推广ID（例如：“ad:hash:5”），同时使用“DEL”命令删除本条广告对应的“hash”类型数据。获取推广展示列表，先使用“LRANGE”命令从列表“home:ad:list”中获取广告ID集合（例如：LRANGE(“home:ad:list”, 0, 9)）。循环遍历，使用“HGETALL”命令获取推广详细信息。

在项目中我们是通过jedis来操作redis的，jedis是Java提供的用来操作redis的API,导入相关的jar包、配置连接池，设置最大连接数，最大空闲数，然后用静态化语句块初始化连接池信息，获得jedis实例；配置jedis connectionfactory ，通过ip和端口号来连接相对的redis数据库。我们封装了相应的工具类，通过操作模板来调用其中的方法，进行增删改查。

）

前台：6.购物车订单管理(redis缓存

## 购物车在redis中是怎么存的？

我们的购物车，用的是redis 来实现的。 当加入购物车的时候 用户id作为redis 的key，产品集合作为redis的value。商品存的是 ，商品id 商品名称，和商品购买数量。

当加入商品到购物车的时候，首先判断当前用户id对应的的产品集合里面是否含有当前产品，有则数量加一。没有则新添加该商品。

订单支付问题

当用户点击“提交订单”时会将redis中的信息取出来放进jms消息队列的生产者里,根据不同的商家,将订单进行拆单处理路由到不同的队列里.消费端通过获取队列里的消息后操作数据库 插入到数据库的订单表中，并清除reidis中该购物车的信息，同时将订单表中该条记录的状态设置为1（未付款），设置过期时间，在规定时间内用户可以选择进行支付，支付成功后会将订单的状态改为2，并将财付通返回的交易信息存入数据库的交易记录表中，同时调用webservice接口发送短信通知用户支付成功，此时的订单已进入后台去处理，前台再查看订单时，

如果在规定时间内，用户没有支付，我们会取消订单，这样是避免其他用户不能购买到商品，我们会根据状态显示不同的信息，比如：状态1是未支付订单，状态为2的是支付成功等待审核状态，当后台审核通过状态会改为3，当发货时状态会改为4，这时客户就可以确认收货，确认收货后状态改为5。注意:订单支付采用的是 微信（腾讯），支付宝(淘宝)

## 订单支付：

（支付我们提交企业营业执照等信息，需要申请支付宝商户号和密钥，支付宝那边会给出示例接口代码，具体的我这这， 点击订单需要提交价格到我们的项目做价格校验，我们需要调用支付宝接口，传递我们的订单编号，价格，支付宝密钥，我们项目的回调的URl给支付宝接口，完了网站页面就会支付宝页面，客户就可以完成一系列支付动作，当支付完成后，我们网站通过这个URl回调，获取到支付的，订单编号，流水号，价格后，进行校验存储交易日志等动作后，返回SUCCESS给银行那边银行的支付操作事物完成）。

支付安全问才说：支付的时候，很多数据都是用MD5加密过的，在一个请求先进入我们的 项目，我们会先校验，价格是否被F12 修改过完了调用支付宝接口，转到支付宝页面，后续的客户输入账号密码，或者网银转账都是在支付宝，和银行之间完成的，钱在银行数据库，支付宝的密码，在支付宝数据库他们有自己的安全措施，而且都走的是https协议都是经过加密的。我们项目再整个过程中只是通知支付宝交钱，具体的交钱动作都有支付宝，和银行系统来完成。所以安全性没问问题。及时是网络抓包工具也只是类似知道客户要付钱给谁。具体赋的过程任何操作都是支付宝，和银行的交易。所以没有安全问题。

**订单号生成：**

即时生成‘日期+6位随机数字’把明天要用的订单号先随机好，放进redis之类的缓存里里随用随取，这样就不会有性能和并发的问题了

)，

7.个人目录管理（主要是用户登陆后可以查看自己购买课程后学习的进度和用户收藏的课程），

1. 课程类型管理（

当项目启动的时候把所有商品信息加载到solr服务器里面进行分词(IK分词器)，创建索引。当用搜索框进行搜索的时候直接调用solr服务器里面的数据，不需要进入数据库查询，降低了服务器压力，Solr还提供了：地理位置检索(特色的功能)、拼写检查、自动补全、关键词高亮显示、检索结果分页、缓存等。

在从数据中取数据生成索引的时候，因为表中的数据量比较大，防止一次取出所导致内存溢出问题，我采用了分段批量提取的方式进行，除此之外我们对后续增加的数据根据优先级的不同采取不同的策略，对于那些需要及时显示的数据我们通过spring定时器  在短时间内(30分钟)进行增量索引的生成，对于那些不需要及时展示的数据,我们通过spring定时器在服务器相对空闲的时候（比如每天晚上凌晨）进行索引的重新生成。此外我们为了提高solr搜索的性能对其进行了solr集群的搭建.

solr 当时用的solr是4.10.4版本的因为这个版本比较好用，稳定，5.以上的版本bug比较多，配IK分词器用的是2012FF\_u1版本，加入分词器的schema文件要把version改为1.5基本的CRUD操作都可以。

导入相应的java包，在schema.xml中配置field标签中配置表的映射信息，name值要和数据库中的字段一致，并且在需要的字段配置分词，并且将version版本改为1.5，在**：**data-config.xml中设置好你的数据库连接还有用于查寻的查询语句

删除：

手动删除也不是不可以，但这人工操作未免太扯了，到时间人工去solr中删除这条数据耗时又耗力，也不太现实。

解决办法就是：

在data-config.xml的entity中添加：

|  |  |
| --- | --- |
| 1 | deletedPkQuery="select id from shops  where endTime &lt; NOW()" |

这一句，这句和entiy的query同级，目的是查询出所有数据中结束时间小于当前时间的课程id,

这个id会和deltaQuery中收集到的id一并执行deltaImportQuery操作，只不过一个是添加数据，一个是删除数据

更新：

1、下载jar包：solr-dataimportscheduler-1.1.jar [http://pan.baidu.com/s/1hsySs2S](http://pan.baidu.com/s/1hsySs2S" \t "http://blog.51cto.com/simplelife/_blank)

 2、新建文件：dataimport.properties，文件复制下面的就好，具体配置含义已给出注释：

1. 在\solrhome目录下新建一个conf文件夹（与simple文件夹同目录），将dataimport.properties文件放在conf文件夹中。

springmvc.xml配置如下  <constructor-arg value="http://localhost:8080/solr/products"></constructor-arg>         -----索引库在solr服务器的地址

分词器的词典编码格式为UTF-8无BOM格式

在从数据中取数据生成索引的时候，因为表中的数据量比较大，防止一次取出所导致内存溢出问题，我采用了分段批量提取的方式进行，

此外我们为了提高solr搜索的性能对其进行了主从配置。

1. 我们solr使用的是solr4.7版本

2. 通过修改schema.xml来添加要进行索引的字段以及增加ik分词器

3. 通过solrj将数据库中的数据生成solr中的索引文件，注：solrj是java程序调用solr服务所用的jar包。

4. 通过在solrconfig.xml中配置requestHandler name=“/Replication”来进行主从同步的配置，在从solr中通过masterUrl指明要从哪些主solr服务器中同步数据

）

**dubbo和zookeeper的配置：**

新建dubbo-provider.xml配置文件，通过dubbo:application配置提供者应用名，通过dubbo:registry配置注册中心的地址，通过dubbo:protocol配置协议，以及通过dubbo:service来暴露要发布的接口。

最后我们在需要使用dubbo接口的项目中配置消费者信息，新建dubbo-consumer.xml文件，通过dubbo:application配置消费者应用名,通过dubbo:registry指明要订阅的注册中心地址，通过dubbo:reference指定要订阅的服务接口。除此之外考虑到dubbo的健壮性和性能我们对它的参数项进行的调优。通过在dubbo:protocol中threadpool="fixed" threads="200"来启用线程池，通过在dubbo:service中connections=5来指定建立长连接的数量。

配置dubbo集群来提高健壮性以及可用性。dubbo默认的集群容错机制是Failover即失败自动切换，默认的重试次数为2，可以通过retries调整。dubbo默认的负载均衡策略是Random随机，可以按权重设置随机概率。我们在写完dubbo提供者之后，为了测试接口的正确性，我们会进行直连测试。首先会在提供者端，通过将dubbo:registry的register设置为false,使其只订阅服务而不注册现在正在开发的服务；

在消费者端，通过设置dubbo:reference的url，直连提供者进行测试。被动说：

所谓dubbo集群就是将dubbo的提供者部署多份，在不同的机器上或者说在同一台机器上用不同的端口号。从而在启动时可以向注册中心进行注册，这样结合dubbo的集群容错策略以及负载均衡策略可以提高可用性。

dubbo负载均衡策略：随机,轮询，最少活跃调用数。

dubbo的集群容错:失败自动切换,快速失败,失败安全。

## dubbo+zookeerper怎样实现session共享（在消费端）：

我们的消费者只有一个模块，所有的请求首先都是进入这个模块里面，生产者有多个模块，session都在消费者这个action里面，所有的请求都是首先进入这个项目里面，我们的生产者有多个模块。如果有多个消费者的情况下，会存在session共享问题，我们可以将session的id作为key值，用户对象作为value值存储到redis里面。当每次发送的请求的时候，拿着浏览器的session的id去redis里面取，如果能取到，证明用户已经存在，如果不能取到，就重新登录。

## 为什么用dubbo+zeekeeper?介绍一下？

实现了分布式部署，如果不用的话，传统的项目就是action>servcie>dao,有三部分的请求，单个儿tomcat，用了dubbo+zookeerper后，分成了消费者和生产者，划分成不同的模块，有几个模块就有几个tomcat，大大降低了tomcat的压力，而且后期随着访问量的增加，我们可以不断的增加生产者，每个单个web服务节点者所受到的压力明显降低。

## 生产者和消费者是怎么交互的？

dubbo首先向zookeerper暴露端口，消费者向它们这边订阅服务通过zookeerper，消费者发送请求，zookeerper用来调度，调度的方式有三种，轮循，随机，把这些请求都拿给生产者，如果其中一个生产者挂掉，如果有新的生产者，会把请求分发给新的生产者。

## 你看过dubbo底层吗？

生产者跟消费者之间相当于长连接长请求，传统的request请求，response响应，请求一次，响应一次，开关一次，dubbo是基于 tcp/ip协议的，其他的源码闲的时候，基本上通过maven来看过没有刻意的去了解，因为平时这个业务开发状况很繁琐。

## zookeeper的作用：

dubbo是把项目实现分离，分为消费端跟服务端，在消费者这边的配置文件中添上zookeeper地址，让它连上zookeeper，每个服务端都暴露一个端口给zookeeper，这样就实现服务端在zookeeper中注册，消费者发送请求给zookeeper，zookeeper接收请求，zookeeper再分发给这些服务端。

**这个生成的静态化的页面同步的问题：**

怎么保证数据库的数据更新的同时静态化页面的数据也更新呢？我们想到了spring的aop组件，在项目中做的时候，我们把所有在service层开启的事物的方法都统一命名。配置aop的切面规则，配置切面代理类，在代理类里面我们用httpcilent远程调用生成静态页面的方法。

## 项目安全性怎么解决？

是指攻击安全还是访问量过大安全还是数据安全？

如果是攻击性的话，input框都采用了js验证，用java代替重复效验，避免输入不必要的空格或者sql注入。

如果是访问的安全的话，页面做了大量的页面静态化，缓存，通常访问量不会增加服务器造成负担。

如果是数据安全这一块的话，系统用的是linux，数据库的账号密码都做了加密处理。

如果是数据遗失或是自然灾害，数据库做了读写分离，而且一个主库，两个从库，其中一个从库做了备份，而且是定时备份。

在登录过程中，为了防止程序直接登录，也做了一些验证码，包括支付的资金账户安全，我们的系统没有钱，主要是通过超链接将钱存储到网络支付宝qq账号，最主要还是支付宝的安全和网民的安全,而且这个账号只能出不能进。

**充值和购买：**

新用户注册登录后，默认余额字段为0，如果用户想要看付费的课程，就得购买，购买时先判断用户的余额是否可以购买,购买的话就要充值学币，当时一学币就是一元，按照充值的金额+用户本来的余额减去课程的价钱，就可以购买了！这个可以直接购买也可以加入购物车购买互不影响；唯一区别就是加入购物车要先放到redis中；

## 问购物车的商品会存到redis里面，假如第二天老板将数据库价格改了，那购物车里面的价格会变化吗？

答：会变化。我的购物车用户id做为key，value是产品集合，而且默认过期时间是7天，产品集合只存储产品id ，数量，名称。 当显示该商品的时候我需要用产品id 去其他redis 库里面获取价格。其他的redis库里面已经存储了所有产品id作为key，value是商品 价格数量。当老板变动数据库价格的时候我只需要更新，单独存储的这边redis产品。

**dubbo 生产者挂掉怎么办。消费者只有一个挂掉怎么办？**

如果IP 请求无响应，则IP转发到另外一个消费者服务器上。

况且我们的消费者是，nginx+tomcat 集群来实现的相当于一个超大容量的tomcat 所以一般也不会那么容易挂掉，在一个既然项目要上线肯定对，访问量有一个预判，肯定要做性能测试的。对应的服务器配置能扛得住上线的预计访问量。真要出了问题那也没办法，就跳到应急页面显示服务器繁忙稍后再试。

## solr搜索引擎：

在整个项目中实现课程搜索功能，因为用户有时候不是多么清楚他所需要的课程名称，又或者是有多个要求，为了满足用户的诸多需求更准确的查询到用户所需要的课程，所以就用到了Solr搜索引擎

solr是一个基于Lucene的全文搜索服务器，相比Lucene更高效、使用更便捷，在进行模糊匹配的时候，他可以 用来替代数据库中的like ,从而在匹配准确性以及性能进行大幅度的提高。因为官方的分词器对中文支持不好所以使用了第三方的IK分词器来进行分词。在建立索引的时候我们通过在schema.xml配置IK分词器来完成中文分词。从而实现了高亮显示关键词，分页，排序，多字段，多条件的高性能搜索。

其实目前有很多优秀的中文分词组件 ：像mmseg4j，IK Analyzer ，Paoding

通过查询资料发现用 IKAnalyzer比较好IK 比其他中文分词维护的勤快，和 Solr 集成也相对容易。

其中我们在配IK分词器的时候，当时遇到了三个问题：

分词器版本与solr服务版本的匹配：

**当时用的solr是4.10.4版本的因为这个版本比较好用，稳定**，5.以上的版本bug比较多，**配IK分词器的2012FF\_u1版本**，开始时用的是u6,然后配完效果出不来，上网查了好多资料，又问了之前加的一个搜索引擎群，才知道是版本问题，u3,u5,u6的都不好用，只有u1的和solr4.10版本的合适，改完之后问题确实解决了

加入分词器的schema文件要把version改为1.5基本的CRUD操作都可以。

但是搜索却只能全字匹配或者只能单个字符匹配出结果。这是绝对不能容忍的。定位问题接近了一天，找有经验的同事给 排查也没排查出来问题。最后我自己一点一点比对multicore文件夹下的配置文件跟F:\solr\solr-4.6.0\example\solr \collection1这个文件夹下的配置文件的配置区别。当我把schema.xml的version属性从1.1升到跟collection1下的相同文件的1.5一致之后。重启服务器，问题解决了！、

分词器的词典编码格式为UTF-8无BOM格式

在从数据中取数据生成索引的时候，因为表中的数据量比较大，防止一次取出所导致内存溢出问题，我采用了分段批量提取的方式进行，

此外我们为了提高solr搜索的性能对其进行了主从配置。

1. 我们solr使用的是solr4.7版本

2. 通过修改schema.xml来添加要进行索引的字段以及增加ik分词器

3. 通过solrj将数据库中的数据生成solr中的索引文件，注：solrj是java程序调用solr服务所用的jar包。

4. 通过在solrconfig.xml中配置requestHandler name=“/Replication”来进行主从同步的配置，在从solr中通过masterUrl指明要从哪些主solr服务器中同步数据

我们为了提高用户体验度还使用了solr的spellCheck可以用于查询输入的自动完成功能auto-complete。他是基于动态代码方式建立内容，suggestion可通过读文件方式建立内容，并有点击率排序。使用facet在查询结果上根据分类添加了count信息, 然后用户根据count信息做进一步的查询, facet的主要好处就是可以任意对搜索条件进行组合, 避免无效搜索, 改善搜索体验.

**登录状态下购物车功能的实现**

功能分析：   
 1、购物车数据保存的位置：

登录状态下，需要把购物车数据保存到服务端。需要永久保存，可以保存到数据 库中。可以把购物车数据保存到redis中。   
 2、redis使用的数据类型   
 a) 使用hash数据类型   
 b) Hash的key应该是用户id。Hash中的field是商品id，value可以是把商品信息转换成json   
 3、添加购物车   
 登录状态下直接把商品数据保存到redis中。   
 未登录状态保存到cookie中。   
 4、如何判断是否登录？   
 a) 从cookie中取token   
 b) 取不到未登录   
 c) 取到token，到redis中查询token是否过期。   
 d) 如果过期，未登录状态   
 e) 没过期登录状态。

实现：

1、登录拦截器   
 几乎在购物车所有功能执行 都要判断用户是否登录。利用aop思想，应该编写个拦截器，来判断用户是否登录。登录的话用户信息需要存在request域中   
 （1） 从cookie中取token   
 （2） 判断token是否存在   
 （3） 不存在，说明用于未登录，放行   
 （4） 如果token存在，调用服务，根据token从redis中取用户信息   
 （5） 取不到用户信息，说明已经过期，放行   
 （6） 取到了用户信息，说明用户已经登录，用户信息存到request中   
 （7） 放行

2、登录状态下，商品添加功能实现

向redis中添加购物车

比未登录多一个登录判断而已

3、登录状态下，商品列表展示   
分析：   
（1）从cookie中取购物车列表   
（2）判断用户是否登录   
（3）用户已经登录的话，则调用服务层，合并cookie中的列表和redis中的列表。存入到redis中。   
（4）同时删除cookie中的购物车列表   
（5）根据用户id，调用服务查询redis中所有的商品，返回购物车列表。   
（6）未登录状态还是跟前面一样   
（7）将列表绑定到参数，返回购物车列表页面

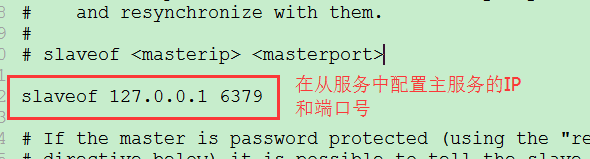
4、登录状态下修改购物车商品数量   
分析   
单击”+”,”-”修改商品的数量的时候,要求redis中该商品的数量发生改变   
（1） 根据用户id，商品id从redis中取出对应的商品   
（2） 设置商品的数量   
（3） 该商品更新到redis中

5、登录状态下，删除购物车商品   
分析   
单击删除的时候，删除redis中该商品。重定向到列表页面   
（1） 直接用jedisClient的del的方法根据用户id跟商品id 商品   
（2） 返回成功

## Redis主从同步：

我们还可以对redis服务器配置主从同步，来减轻redis服务器的压力，

修改从服务对应的主服务地址（修改时可以直接搜索slaveof关键字）



这样访问时会读写分离，查询的时候会走从服务器，而更新操作的时候走主服务器；

与memcached的区别 （问到再说）

1，redis并不是把所有的数据都存放在内存，还可以将一些长期不使用的值存放在磁盘当中。而memcached只能存放在内存当中的；

2，redis的数据类型比memcached的数据类型丰富，除了string外，还有set,list,sortd set，hash等；

3：redis是可以持久化数据的，在一定程度上避免了数据的丢失，而memcached的安全性较低，一旦遇到停电，宕机的情况数据就会丢失；

4设置过期时间，memchched是在set key的时候直接设定时间（memcache.set(key,value,new Date(50)) 设置为50秒过期），而redis设置过期时间是通过expire来设置的；

## 如何在项目中使用redis的：

我们的项目采用的是maven来进行管理的，可以通过maven来进行包的依赖管理，保证jar包版本的一致性，所以只需要在pom.xml里添加对应的<dependency>信息即可自动下载jar包，

1：在项目中我们是通过jedis来操作redis的，jedis是java提供的用来操作redis的API

2：我们需要导入一些相关的jar包，目的是为了一些工具类的使用；

3：配置redis.xml文件,

1）我们配置了连接池；设置了最大链接数，最大空闲数；然后用静态语句块初始化连接池信息，在获得jedis实例

2）配置Jedis ConnectionFactory，通过IP和端口号来链接相对应的redis数据库；

3）配置了模板，并且在模板中引入了redis

的相关jar包；

4：在web.xml文件中引入了 redis.xml文件；

5：我们封装了相对应的工具类；通过操作模板来调用其中的方法，进行增删改查；在service层中获得jedis的实例，通过jedis实例来调用Redis中set类型对应的方法，就可以实现在Redis缓存中放入和取出查询到的商品的信息了，需要注意的是redis里只能存储字节，所以相对应的实体类要实现serializable这个接口；

## redis常见性能问题和解决方案：

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

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

还有就是Redis的秒杀场景

使用的是Redis的事务功能，先通过watch（）监控一个字段，且这个字段为0，之后对于秒杀数进行判断。当监控字段小于秒杀数时在IF条件内部内开启事务通过multi,当有人抢购成功的时候这个监控字段加1，且通过Exec执行事务，事务执行成功则抢购成功，否则抢购失败。抢购成功之后将相应的用户也存储到数据库中然后进行相应的订单处理。当监控字段大于秒杀数的时候也就是抢购完毕。无法进行抢购。

缓存的策略是每次数据在增删改之后就更新，主要是为了避免修改之后缓存更新不及时造成脏数据的出现