**第二天：**

今天主要完成后台商品列表查询功能的开发。

由于分布式商场是基于SOA架构的，后台管理的表现层和服务层是不同的工程，所以要实现商品列表的功能，就需要两个系统之间进行通信。

在这里，本项目使用了Dubbo进行通信，由于dubbo直接使用socket进行通信，所以传输的效率很高，并且可以进行系统间**调用关系和次数的统计**。

**下面对dubbo进行简单的介绍**

当项目中的服务越来越多，需要增加一个调度中心，基于访问压力实时管理集群的容量，提高集群的利用率，而**dubbo就是资源调度和治理中心的管理工具。**

Dubbo采用全Spring配置方式，透明化接入应用，对应用没有任何的API侵入，只需要用Spring加载Dubbo的配置即可。

配置如下

发布服务：

|  |
| --- |
| <!-- 和本地服务一样实现远程服务 -->  <bean id="xxxService" class="com.xxx.XxxServiceImpl" />  <!-- 增加暴露远程服务配置 -->  <dubbo:service interface="com.xxx.XxxService" ref="xxxService" /> |

调用服务：

|  |
| --- |
| <!-- 增加引用远程服务配置 -->  <dubbo:reference id="xxxService" interface="com.xxx.XxxService" />  <!-- 和本地服务一样使用远程服务 -->  <bean id="xxxAction" class="com.xxx.XxxAction">  <property name="xxxService" ref="xxxService" />  </bean> |

官方推荐使用zookeeper注册中心。注册中心提供服务地址的注册于查找，相当于目录服务。

服务层和表现层都必须添加dubbo依赖的jar包，在spring和springMVC的配置文件中药加入dubbo命名空间

服务层和应用层的配置文件中都需要先配置注册中心，然后发布服务和订阅服务

配置如下：

!-- 使用dubbo发布服务 -->

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"e3-manager"* />

<dubbo:registry protocol="zookeeper"

address="192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183" />

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"20880"* />

<!-- 声明需要暴露的服务接口 -->

<dubbo:service interface=*"cn.e3mall.service.ItemService"* ref=*"itemServiceImpl"* />

<!-- 引用dubbo服务 -->

<dubbo:application name=*"e3-manager-web"*/>

<dubbo:registry protocol="zookeeper" address="192.168.25.154:2181,192.168.25.154:2182,192.168.25.154:2183"/>

<dubbo:reference interface=*"cn.e3mall.service.ItemService"* id=*"itemService"* />

由于在web.xml中定义的url拦截形式为“/”表示拦截所有的url请求，包括静态资源例如css、js等。所以需要在springmvc.xml中添加资源映射标签：

需要在springmvc.xml中添加资源映射标签

<mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"*/>

<mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"*/>

至于商品列表查询，使用easyui实现，可以从静态页面中得到请求路径、请求参数（page,rows）以及返回到页面的数据的格式。

格式为：{total:”2”,rows:[{“id”:”1”,”name”:”张三”},{“id”:”2”,”name”:”李四”}]}

可以新建一个pojo（EasyUIS=DataGridResult，total对应Integer字段，rows对应一个List集合）

然后再控制上加入responsebody即可

编写相应的controller（应该从底往上）

首先dao层已经通过mybatis逆向工程生成了mapper类和mapper配置文件，可以自己手写（手写的好处是可以使用方言，limit）

此时要新建一个Page类，里面保存了从数据库查询的信息，包括当前页面，每页记录数，总记录数、总页数和所查询到的List

在service层要新建一个EasyUIDataGridResult类，里面包含两个变量，int total，和查询到的记录List，需要将dao层查询的结果Page对象中对应的字段封装到EasyUIDataGridResult中，使用SpringMVC的@controller注解，可以返回JSON格式的数据。

最后服务层需要在配置文件中发布服务，表现层需要引用服务

**第一天结束**

--------------------------------------------------------------------------------------------------------------------------------

**第二天**

实现了商品列表查询，接下来就是实现后台商品的增删改查

以增加商品为例

商品添加页面。添加商品第一就是需要选择商品的类型，前端页面使用EasyUI的tree来实现。初始化tree的时候只要把第一级节点展示，子节点通过异步的方式加载，当展开父节点的时候触发事件，到后台进行处理。处理的方式很简单，在前端页面中需要获取请求的url，通过controller的requestmapping进行拦截，再获得请求的参数，其实就是父节点的ID，然后到数据库中进行查询，父节点为xxx的都访问。最后，还要获得返回数据的JSON格式，为了方便展示，建立一个EasyUITreeNode类对查询结果进行封装，在这里，包含三个字段，分别是id，闭合状态state和名字text。最后返回的是一个EasyUITreeNode集合，注意的是，由于需要在网络中进行传输，必须实现序列化接口Serializable。

开发顺序一样是先dao层，service（service层要先接口再实现类），再controller。别忘了，服务层需要发布服务，应用层需要引用服务。

接下来要介绍一下nginx，它是一个轻量级、高性能的web server服务器（5w并发），主要可以做两件事：

一是作为http server，代替tomcat，后面静态页面的访问就是使用nginx进行。

二是作为反向代理服务器实现负载均衡。

nginx仅仅处理静态页面，动态的页面(php请求)统统都交付给tomcat服务器来处理。

也就是说，可以把我们网站的静态页面或者文件放置到nginx的目录下；动态的页面和数据库访问都保留到后台的apache服务器上。（在网页静态化中，生成的静态页面就放在配置的nginx路径下）

Nginx负载均衡的5中策略

1. 轮询（默认）。每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，自动删除。
2. 指定权重。指定轮询的几率，weight值和访问比率成正比，用于后端服务器性能不均的情况。upstream backserver { server 192.168.0.14 weight=10; server 192.168.0.15 weight=10; }
3. IP绑定 ip\_hash。每个请求按访问ip的hash结果分配，这样每个方可固定访问一个后端服务器，可以解决session共享的问题。
4. Fair（第三方）。按后端服务器的响应时间来分配，相应时间短的优先分配。
5. url\_hash（第三方）。按访问url的hash结果进行分配，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

图片上传功能，需要建立一个图片服务器，用的是FastDFS。上传图片以后，会返回文件的索引信息。格式如下：



文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

添加商品的功能：访问的uri为：/item/save。与之前的处理方式类似，使用一个pojo来接受表单的数据，需要注意的是，pojo的属性和input标签的name属性名要一致，这样才会自动进行注入。这里使用TbItem为接受数据的pojo

再看返回的数据类型，依旧为JSON，而且必须包含一个status属性。这里我们新建一个E3Result的pojo，用来保存各种数据处理结果的返回。

需要注意的是，商品ID不采用自增长，因为要考虑到数据库分片和合并，生成的方式是日期+时间+随机数。

注意，商品信息要插入到两个表中（TbItem和TbItemDesc），所以controller的入参应该为(TbItem item, String desc)，然后传入到service，在service中再建立商品详情对象，注入相关属性，要调用两个mapper方法插入数据库。

**第四天结束**

---------------------------------------------------------------------------------------------------------------------------------

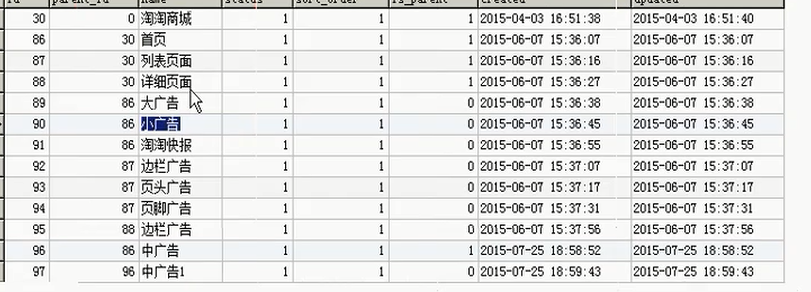
**第五天**

今天主要是前端页面的展示，工程名e3-portal-web（war包）

有一个小细节是，前端页面最好是以html结尾，这样容易被搜索引擎抓取到。所以在web.xml中配置springMVC的拦截后缀是\*.html，伪静态化。

门户网站中的所有图片展示（不仅仅是图片，包括首页的新闻，链接，公告，所有东西都是分组的）都是由组来控制的，每一个模块都是一个组，组内可能还包含小组，每个组都有对应的组ID

看下图，分布式商城中的主页就包含这么多的组



要进行前台轮播图的展示，必须要有一个后台工程来管理各个模块（组），进行维护，让前台页面展示什么都是由后台管理系统来控制的。 （本质上是通过数据库来进行通信的，后台决定需要在哪些模块添加什么内容，什么图片，然后把信息保存到数据库中，然后打开门户网站的自动查询数据库中，把信息展示出来）

现在考虑数据库中应该保存什么信息呢？首先！必须有一张图片，这个图片在数据库中是一个连接地址，指向图片服务器。其次，还有有一个跳转链接，点击图片必须要跳转到商品详情页面。最后可能还需要一些额外信息，比如商品标题啊什么的，但是最核心的肯定是前两个。

主页所有信息的展示都是由内容管理系统解决，本项目中为e3-content

内容管理首先需要展示所有分组之间的管理，使用的一样是EasyUI的Tree控件，需要注意的是，第一次访问的时候是没有带参数的，所以在Controller要设置参数的默认值为0.

以后每次提交请求都是带参数的，驱动时间为每次展开tree或者关闭tree

明白这个以后，就可以根据前端静态页面来获取我们需要的信息

1.请求的url；2.请求带的参数，这里是父节点的id，第一次访问的时候没有父节点，所以默认值为0；3.返回信息的格式（这里依然是EasyUITreeNode）。

控制器的参数绑定格式：使用@RequestParam(value=”id”,defaultValue=”0”)

当然我们还要在工程中能够对这些组（节点）进行管理，即增删改查，增加节点的返回对象是E3Result，插入表tb\_content\_category

插入节点后，还要将该节点的父节点的isParent字段设为true；

重命名、删除一样做，注意的是，删除的时候，如果是删除一个父节点，建议是返回错误信息，不允许删除。

接下来，对每一个组中的内容进行维护，可以增删改查。

还是一样的套路，获得请求路径、参数、返回信息格式，就是在把信息插入到数据库的时候，需要带上商品所属组的组id，这样前端页面在查询数据库的时候，才能获得各自的组所拥有的数据信息。

还是要把消费者和供应者进行分离。这里提供服务端是e3-content，而消费者是e3-manage-web。所以还是要发布和订阅服务。

前端页面展示：根据组id去查询刚刚的数据表，一样要从前端页面中获得url，参数（组ID），返回格式。如果没有参数，则参数使固定的，也要写在配置文件中！！

**第五天结束**

--------------------------------------------------------------------------------------------------------------------------------

**第六天**

使用Redis做缓存

主页内容展示的分析（以轮播图为例）

根据分类id查询内容列表，把内容展示到首页。

内容分类id需要是固定的。可以配置到属性文件中。

展示首页之前，先查询内容列表，然后展示到首页。

使用Redis做缓存，redis的持久化方案有两个：Rdb和aof

**Rdb**是Redis默认开启的，快照形式，**定期把内存中当前时刻的数据保存到磁盘**。Redis默认支持的持久化方案。

定期的栗子：（在配置文件中）

Save seconds changes

Save 900 1

Save 300 10

Save 60 10000

运行原理是这样的：60秒的时候先检查一次，如果修改的记录是达到10000，则同步到磁盘。否则，在300秒的时候检查，如果修改的记录超过10个，同步。否则，900秒的时候检查，如果超过一个，就同步。

**Aof**：默认是不开启的。append only file。把所有对redis数据库操作的命令，增删改操作的命令。保存到文件中。数据库恢复时把所有的命令执行一遍即可。为什么默认不开启呢？因为它默认的配置是**每秒钟**操作一个更新

**Redis集群暂时跳过，以后继续复习**

操作单机版Redis，有专门的Java类，即Jedis，创建的时候输入redis服务器的地址

如：Jedis jedis = new Jedis("192.168.25.153", 6379);

然后就可以操作缓存了

注意：单机版和集群版不能共存，使用单机版时注释集群版的配置。使用集群版，把单机版注释。

前端你的首页访问量是非常大的，如果每次访问首页都去查询数据库，非常浪费资源。而且首页的内容并不经常改变，所以可以把首页各个组的内容放入缓存（content数据表）

如何使用redis对首页各个组的内容进行缓存呢？

使用redis的hash结构，hash的key就是组id，内容中以key-value保存了数据表中的内容信息。

String json = jedisClient.hget(CONTENT\_KEY, cid + "");

if (StringUtils.isNotBlank(json)) {

//把json转换成list

List<TbContent> list = JsonUtils.jsonToList(json, TbContent.class);

return list;

jedisClient.hset(CONTENT\_KEY, cid + "", JsonUtils.objectToJson(list));

用到了JsonUtils工具包

一般在服务层加缓存会比较好，这样，每一个调用这个服务的调用者，都可以从缓存中获得数据

在e3-portal-web中查询主页信息时，实际上调用还是e3-content中的服务，所以需要修改e3-content中查询列表方法，如果缓存中有数据，就从缓存中直接获取；如果没有，则查询数据库，同时，把查询到的东西保存到数据库中。

此时还有一个问题，如果后台管理系统通过e3-content服务，对某一个组中的内容进行了增删改，会发生什么呢？很简单，由于不会同步缓存，前端页面看到的还是修改之前的页面，怎么解决这个问题呢？

其实很简单，只要通过e3-content对组中内容进行增删改的时候，把对应组（redis中hash结构，是以组id区分的）的hash结构完全删除，当下一次有用户访问前台的时候，调用查询后台服务，因为缓存中的数据被删除，只能重新从数据库中获取所需数据并且再次同步到缓存中，就解决了缓存同步的问题。

**第六天结束**

---------------------------------------------------------------------------------------------------------------------------------

**第七天**

今天主要完成了前端页面的商品搜索功能，既然是一个新功能，那么久要建立一个新的聚合工程。E3-search工程，一样可以为其他模块提供服务

要使用solr完成搜索功能，首先需要搭建solr服务，然后将数据库中的数据导入索引库。

想要把数据导入到solr中，必须先配置相应的业务域，业务域必须先配置再使用

由于商品信息时中文的，所以不能使用solr默认的分析器，需要配置中文分析器，我们使用IKAnalyzer。把IKAnalyzer的扩展词典、配置文件放到solr工程的WEB-INF/classes目录下。

jar放到配置solr的服务器的tomcat下工程的WEB-INF/lib目录下。

配置业务域的时候要注意，需要两个表的字段，一个是商品信息表，然后商品信息表中有一个字段是cid，表示所属类别id，所以我们还要获得类别的名称，所以还需要类别表。

商品id、标题、卖点、价格、图片、分类名称

需要在schema.xml中配置，先定义后使用

在schema.xml配置文件中要先定义分析器，然后业务域（标题、卖点）要引用这个分析器（使用配置的分析器对每个域进行分词）

E3-manege-web后台管理可以实现数据库导入，但这只是表现层，不能在表现层实现业务逻辑，所以需要新建一个e3-search服务层工程，提供服务。

SolrJ API的使用

@Test

**public** **void** addDocument() **throws** Exception {

// 第一步：把solrJ的jar包添加到工程中。

// 第二步：创建一个SolrServer，使用HttpSolrServer创建对象。

SolrServer solrServer = **new** HttpSolrServer("http://192.168.25.154:8080/solr");

// 第三步：创建一个文档对象SolrInputDocument对象。

SolrInputDocument document = **new** SolrInputDocument();

// 第四步：向文档中添加域。必须有id域，域的名称必须在schema.xml中定义。

document.addField("id", "test001");

document.addField("item\_title", "测试商品");

document.addField("item\_price", "199");

// 第五步：把文档添加到索引库中。

solrServer.add(document);

// 第六步：提交。

solrServer.commit();

}

学会使用API以后，就可以进行商品信息导入了，由于要用到分表查询，不能使用逆向工程生成的mapper和配置文件，所以需要自己定义一个pojo，包含了导入solr库需要的信息。注意，这里需要用到多表查询。

接下来实现service层，定义接口，再定义实现类。（需要将solrJ的jar包添加到工程）

通过spring类配置HttpSolrServer，然后注入到工程中

将查询的到的结果集通过foreach循环，利用SolrInputDocument写入索引库。

编写完逻辑要进行对服务进行发布。

然后再分析静态页面，获得请求的url，是否带参数，以及相应的结果格式。这里相应结果可以使用E3Result。

这样就完成数据库导入solr库惹！接下来就是搜索功能的实现。搜索使用的solr的api—solrJ

首先需要根据界面所需数据的格式，首先建立一个封装数据的pojo，即SearchItem，然后建立一个前端页面所需的SearchResult的pojo，包含的字段有查询结果List<SearchItem>总页数和总记录数。

Service层需要做的事情有：

1. 根据掐你段参数创建一个查询条件，需要制定默认的搜索域
2. 根据dao查询得到一个SearchResult对象
3. 计算查询的总页数，每页侠士的记录数就是rows参数

之后进行服务的发布

这里要注意乱码：

Post乱码可以在web.xml中进行配置，get乱码有两种解决方法：1.修改tomcat配置文件，设置编码格式为utf-8；2.手动转码keyword = **new** String(keyword.getBytes("iso8859-1"), "utf-8");

**第七天结束**

--------------------------------------------------------------------------------------------------------------------------------

**第八天**

今天是要是使用ActiveMQ消息队列

现在考虑一个问题，由于我们使用solr进行商品的搜索功能，我们之前使用的是一键导入solr库的功能，那如果此时我在后台添加一个商品，solr要怎么操作呢？难道重新调用一键导入吗？这样显然是非常浪费资源的，还是当后台添加商品的时候，同时把这个商品加入到solr库中呢？这显然也不行，因为搜索功能是在e3-search工程中实现的，你不能在e3-manage工程进行搜索功能的相关操作，这样会造成系统间的耦合。

如何同步索引库：

方案一：在taotao-manager中，添加商品的业务逻辑中，添加一个同步索引库的业务逻辑。

缺点：业务逻辑耦合度高，业务拆分不明确

方案二：业务逻辑在taotao-search中实现，调用服务在taotao-manager实现。业务逻辑分开。（为什么呢？因为这两个模块都是服务层，服务层之间理应没有谁先启动谁后启动这一说，如果两个服务层模块互相调用，就必须保证search模块要先于manage模块启动，其实这也是一种耦合）缺点：服务之间的耦合度变高。服务的启动有先后顺序。

方案三：使用消息队列。MQ是一个消息中间件。

同步索引库

Consumer

使用ActiveMQ：先添加jar包依赖，然后使用API操作

**第八天结束**

--------------------------------------------------------------------------------------------------------------------------------

**第九天**

今天内容：

1. Activemq整合springMQ的应用场景
2. 添加商品同步索引库
3. 商品详情页面动态展示
4. 展示详情页面使用缓存

解释一下消费者的原理，使用topic（实际应用比较多的使用topic）

创建一个MyMessageListener implements MessageListener，重写onMessage方法

最重要的是要在springmvc.xml中配置监听器，即我们的实现类MyMessageListener

然后！！配置消息监听容器

<!-- 消息监听容器 -->

<bean class=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>

<property name=*"connectionFactory"* ref=*"connectionFactory"* />

<property name=*"destination"* ref=*"queueDestination"* />

<property name=*"messageListener"* ref=*"myMessageListener"* />

</bean>

当有生产者发布消息的时候，消费者监听器坚挺到这个消息，会自动的运行onMessage方法。

知道上面的原理，当我们在后台管理系统对系统进行增删改的时候，就能同步索引库了。

具体的实现方式是：

e3-manager-server工程中发送消息。

当商品增删改完成后发送一个TextMessage，包含一个商品id。直接在原有的方法上发送消息（这里发送的消息是商品的ID）

当消费者端，即e3-search工程，当监听到有商品操作信息了，马上就行相关的操作，同步索引库。

搜索到结果以后，我们点击相关商品，那么就会进入到商品详情页面。

商品详情页面是通过e3-item-web展示的

请求

请求的url：/item/{itemId}

参数：商品id

返回值：String 逻辑视图

添加商品详情的缓存，使用redis

业务逻辑：

1. 根据商品id到缓存中命中
2. 查到缓存，直接返回。
3. 差不到，查询数据库
4. 把数据放到缓存中
5. 返回数据

缓存中缓存热点数据，提供缓存的使用率。需要设置缓存的有效期。一般是一天的时间，可以根据实际情况跳转。

需要使用String类型来保存商品数据。

可以加前缀方法对象redis中的key进行归类。

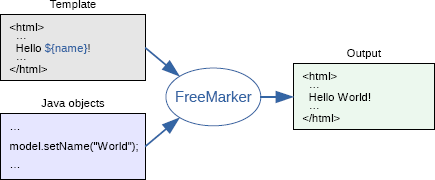
ITEM\_INFO:123456:BASE

ITEM\_INFO:123456:DESC

**第九天结束**

主要用freemaker做静态页面或是页面展示

原理如下图所示：



使用步骤：

第一步：创建一个Configuration对象，直接new一个对象。构造方法的参数就是freemarker对于的版本号。

第二步：设置模板文件所在的路径。

第三步：设置模板文件使用的字符集。一般就是utf-8.

第四步：加载一个模板，创建一个模板对象。

第五步：创建一个模板使用的数据集，可以是pojo也可以是map。一般是Map。

第六步：创建一个Writer对象，一般创建一FileWriter对象，指定生成的文件名。

第七步：调用模板对象的process方法输出文件。

第八步：关闭流。

在spring整合freemaker的时候，需要配置：1.模板文件路径；2.字符集

商品详情页面静态化的方案：

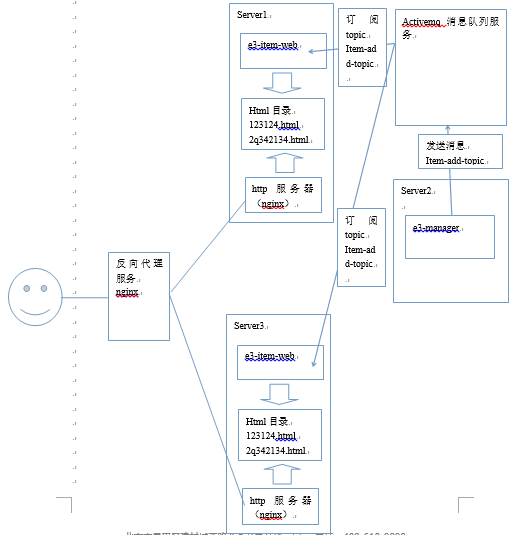
输出文件的名称：商品id+“.html”

输出文件的路径：工程外部的任意目录。

网页访问：使用nginx访问网页。在此方案下tomcat只有一个作用就是生成静态页面。

工程部署：可以把e3-item-web部署到多个服务器上。

生成静态页面的时机：商品添加后，生成静态页面。可以使用Activemq，订阅topic（商品添加）



所以一开始用的是Redis缓存，把商品详情信息放在redis的string中，设置过期时间60分钟等。但还是动态jsp页面

后来发现其实可以用freemaker生成静态页面，进一步优化

接下来要解决商城的登陆问题。

本质上是为了解决session共享的问题

新建一个工程用来解决登陆功能，这就是单点登陆系统。

由于session同一进行管理，就不能使用tomcat自带的session，需要用redis的string模拟session，并设置过期时间。

Key是uuid，代表每个用户，这里成为token（令牌）。value就是用户信息（Json格式）

然后还要把token保存到浏览器的cookie中去。

添加购物车的时候需要考虑：用户是否登陆

如果用户已经登陆，那么直接把购物车写入redis缓存（使用hash数据结构）

如果用户没有登陆，还是写入cookie

判断用户是否登陆需要用到springmvc的拦截器，具体的配置见博客：<http://blog.csdn.net/zjkc050818/article/details/78549420>

在拦截器的preHandle方法中进行以下操作：

1. 通过获取cookie中的token，然后使用token是seesion中获得用户信息
2. 只要有一步失败，说明没有登陆
3. 如果用户登陆了，把用户信息写入request
4. 在控制器中就可以通过判断request中是否有user，来判断是否登陆
5. 该功能在购物车结算模块一样可以使用。