**新巴巴运动网**

# 项目介绍

## 项目的类型

目前：项目分为：互联网项目、非互联网项目（传统项目）

总结：

互联网运行在公网、外网上的（可以上世界上所的人访问） 互联网网站

传统项目：只允许少量人或公司内容员工使用（它运行在内网）

## 新巴巴运动的类型

互联网项目（对外访问）当中的电子商务类型的项目

## 什么是电子商务？

电子商务：通过互联网的形势、进行经商、交易、买卖、并提供服务（售前、集中、售后）的一种形势称为电子商务（简称电商）

电子：通过互联网的形势

商：经商、交易、买卖

务：服务（售前、售中、售后）

售前：京东商城（提供在线资询）

售中：京东商城（提前打电话在家吗？）

售后：目前七天无理由退货、支持三包

## 新巴巴运动网是一个什么样的网站？

新巴巴运动网是一家专业的运动类销售平台、它属于电子商务类型的网站、它拥有的模块有：商品模块、品牌模块、订单模块、购物车模块、内容管理模块等等。。。

## 同类网站

例如：京东、淘宝、天猫、阿里巴巴、唯品会、聚美优品。。。。

# 导入三大框架整合需要的所有的Jar包

## 父工程

|  |
| --- |
| <!-- jdk1.7 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.7</source>  <target>1.7</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build>  <!-- 设置统一控制版本 -->  <properties>  <java-version>1.7</java-version>  <junit-version>4.12</junit-version>  <spring.version>4.1.3.RELEASE</spring.version>  <jackson.version>2.4.2</jackson.version>  <mysql-connector-java-version>5.1.8</mysql-connector-java-version>  <org.mybatis-version>3.2.7</org.mybatis-version>  <org.mybatis-spring-version>1.2.2</org.mybatis-spring-version>  <opensymphony-version>2.4.2</opensymphony-version>  <freemarker-version>2.3.18</freemarker-version>  <druid.version>1.0.9</druid.version>  <commons-collections-version>1.0</commons-collections-version>  <commons-fileupload-version>1.2.2</commons-fileupload-version>  <org.apache.commons-version>3.1</org.apache.commons-version>  <commons-codec-version>1.6</commons-codec-version>  <dom4j-version>1.6.1</dom4j-version>  <javax.servlet-version>1.2</javax.servlet-version>  <aspectjweaver-version>1.6.6</aspectjweaver-version>  <slf4j-log4j12-version>1.6.6</slf4j-log4j12-version>  <log4j-version>1.2.16</log4j-version>  <javax.servlet-jsp-version>2.0</javax.servlet-jsp-version>  <cglib-version>2.2.2</cglib-version>  <slf4j-api-version>1.6.6</slf4j-api-version>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <!-- 依赖管理 版本号 -->  <!-- 设置统一依赖管理、不强行依赖 -->  <dependencyManagement>  <dependencies>  <!-- with junit4.8.2 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit-version}</version>  <type>jar</type>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jms</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- Jackson Json处理工具包 -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>  <!--freemarker -->  <dependency>  <groupId>org.freemarker</groupId>  <artifactId>freemarker</artifactId>  <version>2.3.18</version>  </dependency>  <!-- with mybatis-spring -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${org.mybatis-version}</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${org.mybatis-spring-version}</version>  </dependency>  <!-- jdbc driver -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql-connector-java-version}</version>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>${druid.version}</version>  </dependency>  <!-- apache commons jar -->  <dependency>  <groupId>commons-collections</groupId>  <artifactId>commons-collections</artifactId>  <version>${commons-collections-version}</version>  </dependency>  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>${commons-fileupload-version}</version>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>${org.apache.commons-version}</version>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>${commons-codec-version}</version>  </dependency>  <!-- analyze xml use dom4j -->  <dependency>  <groupId>dom4j</groupId>  <artifactId>dom4j</artifactId>  <version>${dom4j-version}</version>  </dependency>  <!-- jstl标签 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  <version>1.1.2</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.6.6</version>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.6.6</version>  </dependency>  <dependency>  <groupId>javax.activation</groupId>  <artifactId>activation</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>cglib</groupId>  <artifactId>cglib</artifactId>  <version>${cglib-version}</version>  </dependency>  <dependency>  <groupId>net.fckeditor</groupId>  <artifactId>java-core</artifactId>  <version>2.6</version>  </dependency>  <dependency>  <groupId>org.json</groupId>  <artifactId>json</artifactId>  <version>20131018</version>  </dependency>  <!--page -->  <dependency>  <groupId>com.babasport</groupId>  <artifactId>page</artifactId>  <version>1.0</version>  </dependency>  <!-- solrJ -->  <dependency>  <groupId>org.apache.solr</groupId>  <artifactId>solr-solrj</artifactId>  <version>4.10.3</version>  </dependency>  <!-- jedis Java接口 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.6.2</version>  <type>jar</type>  </dependency>  <!-- FastDFS client -->  <dependency>  <groupId>fastdfs\_client</groupId>  <artifactId>fastdfs\_client</artifactId>  <version>1.20</version>  </dependency>  <dependency>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  <version>3.2.5.Final</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>dubbo</artifactId>  <version>2.5.3</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.1.41</version>  </dependency>  <!-- Zookeeper 用于分布式服务管理 -->  <dependency>  <groupId>com.github.sgroschupf</groupId>  <artifactId>zkclient</artifactId>  <version>0.1</version>  </dependency>  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  <version>3.4.6</version>  </dependency>  <!-- Zookeeper 用于分布式服务管理 end -->  <!-- Active MQ 开始 -->  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-all</artifactId>  <version>5.8.0</version>  </dependency>  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-pool</artifactId>  <version>5.8.0</version>  </dependency>  <!-- Active MQ 结束 -->  <!-- Tomcat7 servlet-api -->  <dependency>  <groupId>org.apache.tomcat</groupId>  <artifactId>tomcat-jsp-api</artifactId>  <version>7.0.47</version>  <scope>provided</scope>  </dependency>  </dependencies>  </dependencyManagement> |

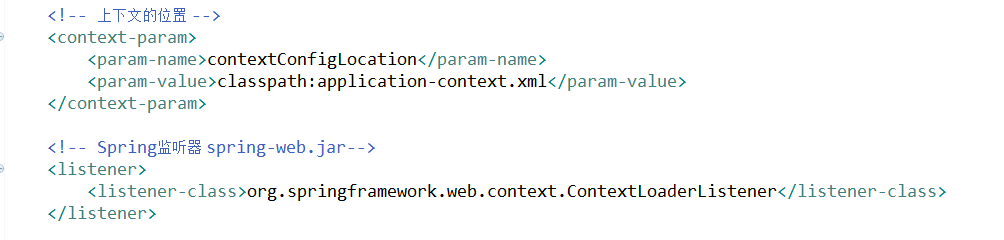
## Babasport-common

|  |
| --- |
| <!-- 依赖管理之后台 -->  <dependencies>  <!-- with junit4.12 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jms</artifactId>  </dependency>  <!-- Jackson Json处理工具包 -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </dependency>  <!--freemarker -->  <dependency>  <groupId>org.freemarker</groupId>  <artifactId>freemarker</artifactId>  </dependency>  <!-- with mybatis-spring -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  </dependency>  <!-- jdbc driver -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  </dependency>  <!-- apache commons jar -->  <dependency>  <groupId>commons-collections</groupId>  <artifactId>commons-collections</artifactId>  </dependency>  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  </dependency>  <!-- analyze xml use dom4j -->  <dependency>  <groupId>dom4j</groupId>  <artifactId>dom4j</artifactId>  </dependency>  <!-- jstl标签 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  </dependency>  <dependency>  <groupId>javax.activation</groupId>  <artifactId>activation</artifactId>  </dependency>  <dependency>  <groupId>cglib</groupId>  <artifactId>cglib</artifactId>  </dependency>  <dependency>  <groupId>net.fckeditor</groupId>  <artifactId>java-core</artifactId>  </dependency>  <dependency>  <groupId>org.json</groupId>  <artifactId>json</artifactId>  </dependency>  <!--page -->  <dependency>  <groupId>com.babasport</groupId>  <artifactId>page</artifactId>  </dependency>  <!-- solrJ -->  <dependency>  <groupId>org.apache.solr</groupId>  <artifactId>solr-solrj</artifactId>  </dependency>  <!-- jedis Java接口 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>  <!-- FastDFS client -->  <dependency>  <groupId>fastdfs\_client</groupId>  <artifactId>fastdfs\_client</artifactId>  </dependency>  <dependency>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>dubbo</artifactId>  <exclusions>  <exclusion>  <artifactId>spring</artifactId>  <groupId>org.springframework</groupId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  </dependency>  <!-- Zookeeper 用于分布式服务管理 -->  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  </dependency>  <dependency>  <groupId>com.github.sgroschupf</groupId>  <artifactId>zkclient</artifactId>  </dependency>  <!-- Zookeeper 用于分布式服务管理 end -->  <!-- Active MQ 开始 -->  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-all</artifactId>  </dependency>  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-pool</artifactId>  </dependency>  <!-- Active MQ 结束 -->  <!-- tomcat7 servlet-api -->  <dependency>  <groupId>org.apache.tomcat</groupId>  <artifactId>tomcat-jsp-api</artifactId>  <scope>provided</scope>  </dependency>  </dependencies> |

# 整合Spring + Mybatis

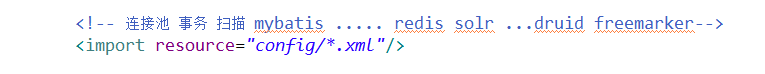
## 配置 web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">    </web-app> |



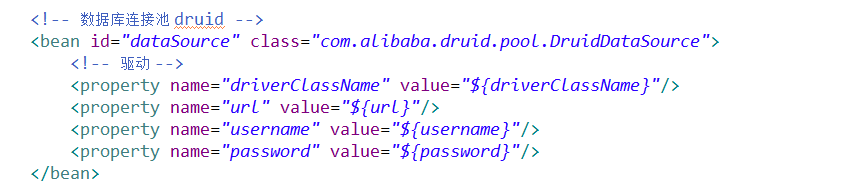
## 配置application-context.xml

|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:task="http://www.springframework.org/schema/task"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-4.0.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/task  http://www.springframework.org/schema/task/spring-task-4.0.xsd  http://code.alibabatech.com/schema/dubbo  http://code.alibabatech.com/schema/dubbo/dubbo.xsd">    </beans> |



## 配置连接池druid

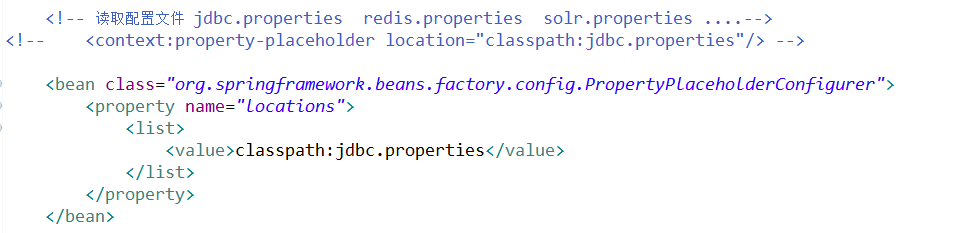
以下为一个参考的连接配置



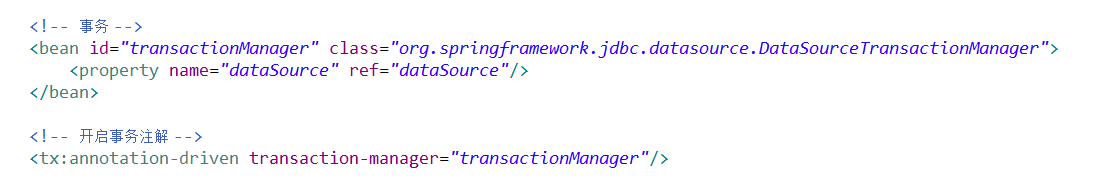
## 配置Jdbc.properties

|  |
| --- |
| driverClassName=com.mysql.jdbc.Driver  url=jdbc:mysql://localhost:3306/babasport?characterEncoding=UTF-8  username=root  password=root |

## 配置property的读取配置类



## 配置事务



## 配置mybatis



## 配置mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"> |



## Spring的扫描

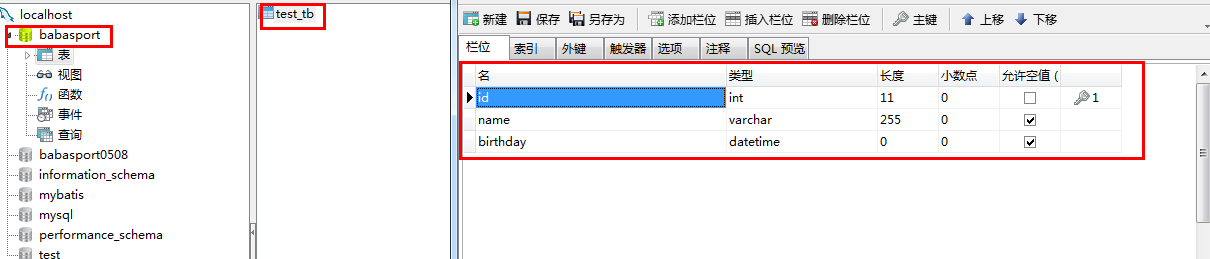
要求：Spring不要扫描Springmvc的@Controller注解



# 对Spring整合Mybatis进行测试

## 创建测试表test\_tb

|  |
| --- |
| SET FOREIGN\_KEY\_CHECKS=0;  -- ----------------------------  -- Table structure for `test\_tb`  -- ----------------------------  DROP TABLE IF EXISTS `test\_tb`;  CREATE TABLE `test\_tb` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(255) DEFAULT NULL,  `birthday` datetime DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8; |



## 创建一个JavaBean （TestTb）

|  |
| --- |
| /\*\*  \* 测试JavaBean  \* @author lx  \*  \*/  public class TestTb implements Serializable {  private Integer id;  private String name;  private Date birthday; |

## 创建一个TestTbDao.java接口

|  |
| --- |
| /\*\*  \* 测试接口  \* @author lx  \*  \*/  public interface TestTbDao {  //保存数据  public void addTestTb(TestTb testTb);  } |

## 创建一个TestTbDao.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- 正是Mybatis写Sql的标签 -->  <mapper namespace="cn.itcast.core.dao.TestTbDao">  <!-- 添加保存测试数据 -->  <insert id="addTestTb" parameterType="TestTb">  insert into test\_tb  (id,name,birthday)  values  (#{id},#{name},#{birthday})  </insert>  </mapper> |

## Junit单元测试（注解式）

|  |
| --- |
| package cn.itcast.common.junit;  import org.junit.runner.RunWith;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  /\*\*  \* Junit单元测试是基于Spring 注解式  \* @author lx  \*  \*/  @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(locations = {"classpath:application-context.xml"})  public class SpringJunitTest {  } |

|  |
| --- |
| package cn.itcast;  import org.junit.Test;  import org.springframework.beans.factory.annotation.Autowired;  import cn.itcast.common.junit.SpringJunitTest;  import cn.itcast.core.bean.TestTb;  import cn.itcast.core.dao.TestTbDao;  /\*\*  \* Junit单元测试是基于Spring 注解式  \* @author lx  \*  \*/  public class TestTestTb extends SpringJunitTest {  @Autowired  private TestTbDao testDao;    @Test  public void testAdd() throws Exception {  TestTb testTb = new TestTb();  testTb.setName("小明");    testDao.addTestTb(testTb);  }    } |

## 测试事务（注解）

TestTbServiceImpl

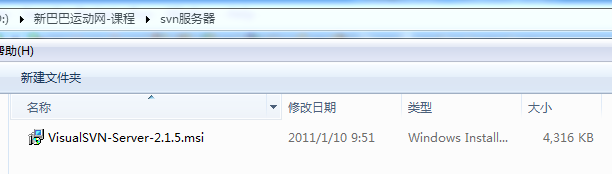
|  |
| --- |
| package cn.itcast.core.service;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import cn.itcast.core.bean.TestTb;  import cn.itcast.core.dao.TestTbDao;  /\*\*  \* 测试事务  \* @author lx  \*  \*/  @Service  @Transactional  public class TestTbServiceImpl implements TestTbService {  @Autowired  private TestTbDao testTbDao;  //添加  public void addTestTb(TestTb testTb){    testTbDao.addTestTb(testTb);    throw new RuntimeException();  }    } |

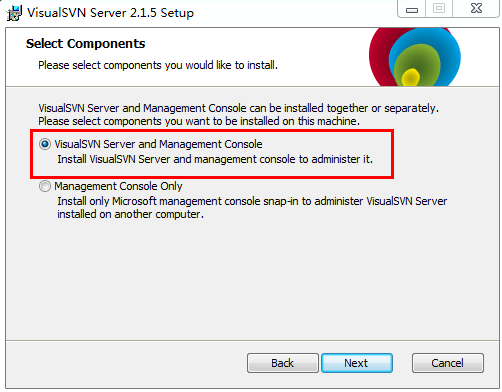
TestTestTb

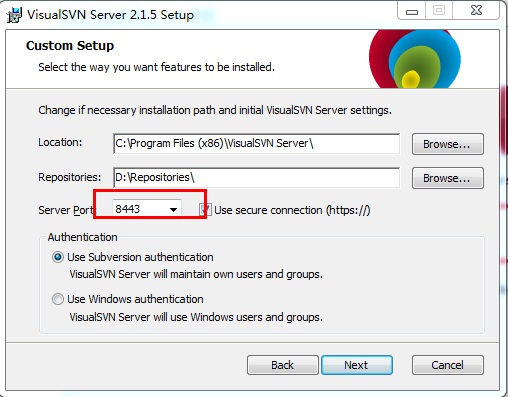
|  |
| --- |
| package cn.itcast.core.service;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import cn.itcast.core.bean.TestTb;  import cn.itcast.core.dao.TestTbDao;  /\*\*  \* 测试事务  \* @author lx  \*  \*/  @Service  @Transactional  public class TestTbServiceImpl implements TestTbService {  @Autowired  private TestTbDao testTbDao;  //添加  public void addTestTb(TestTb testTb){    testTbDao.addTestTb(testTb);    throw new RuntimeException();  }    } |

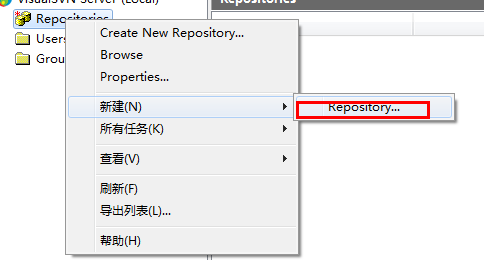
# SVN使用

注意事项：window7 8 主机名不能是中文 计算机名



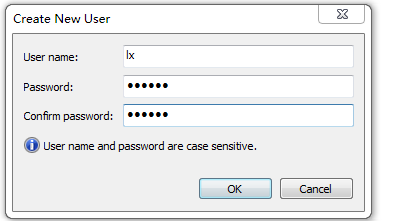




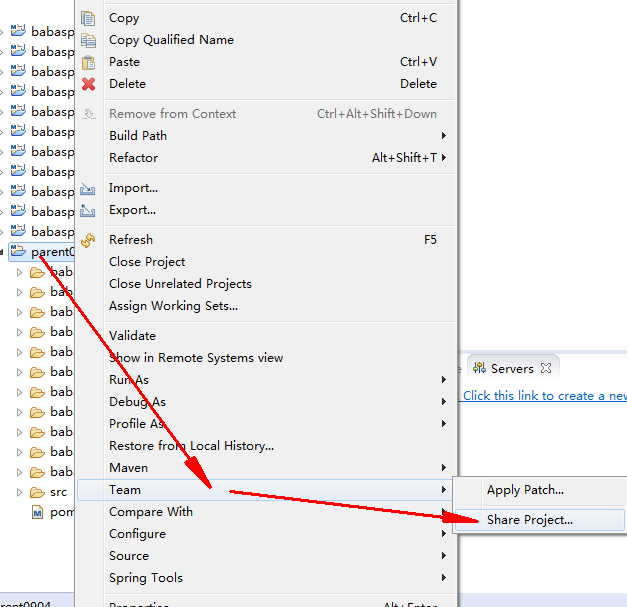


连接地址

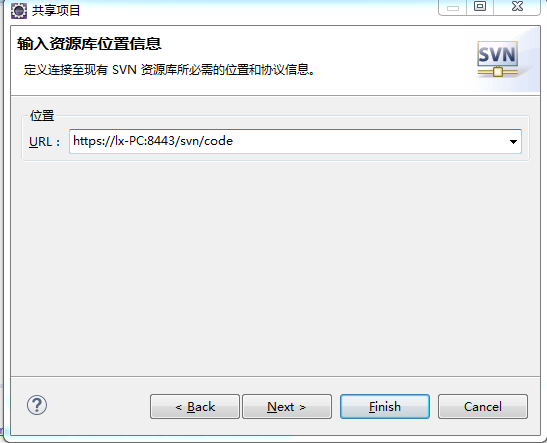




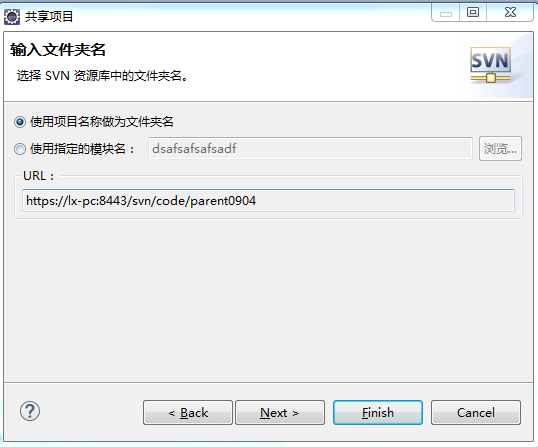
## 提交代码



第二步：



第三步：



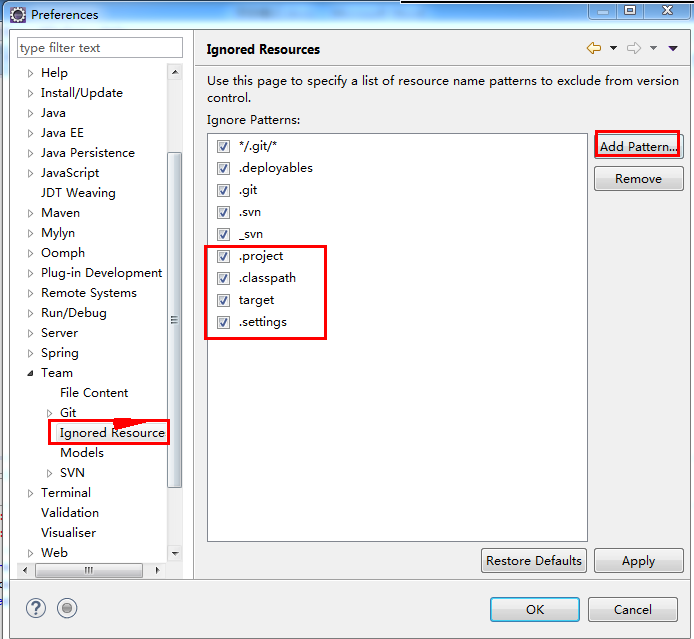
第五步：



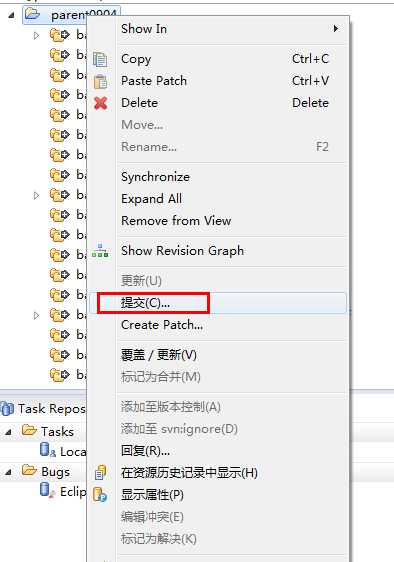
第六步  

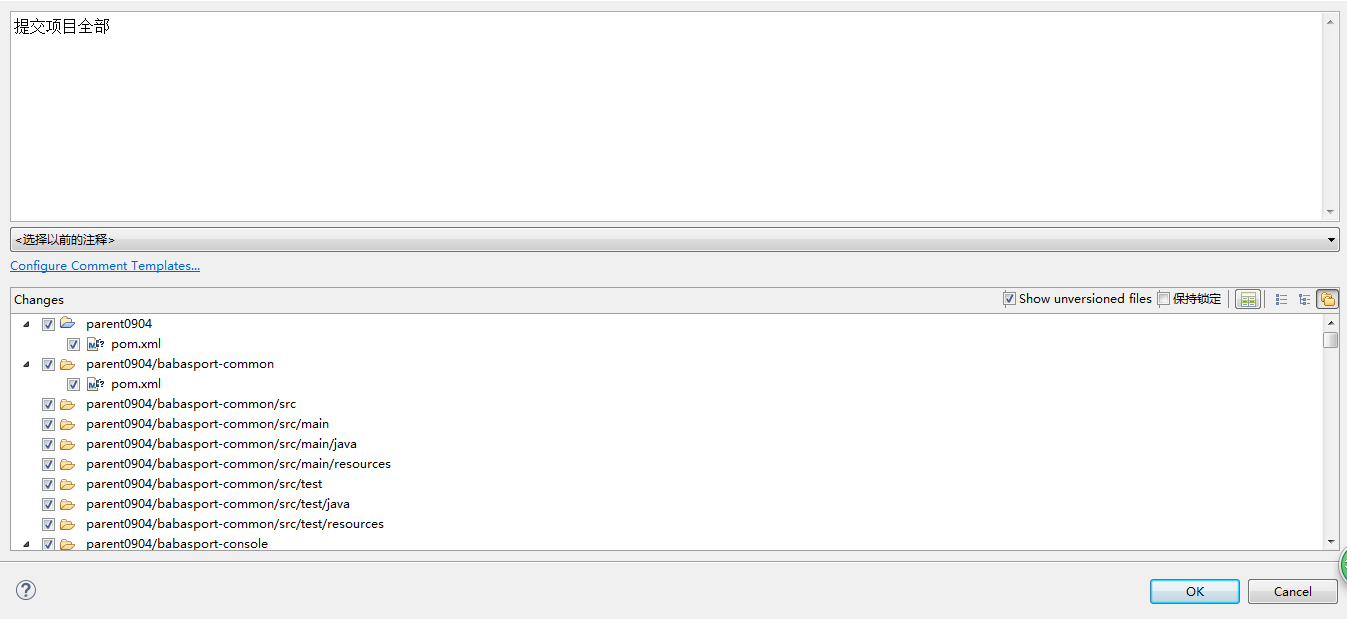

### 过滤提交

.setting .classpath .project target

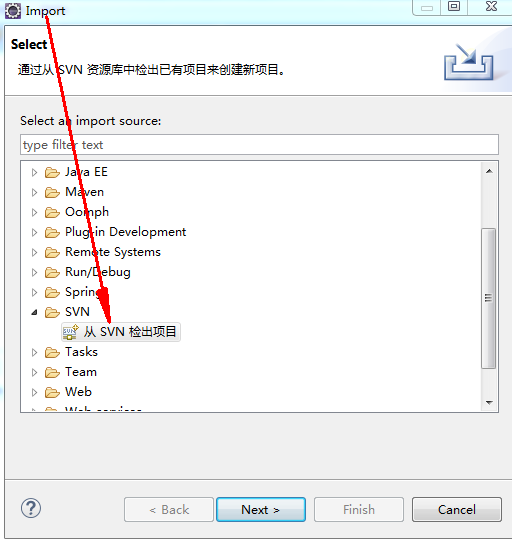


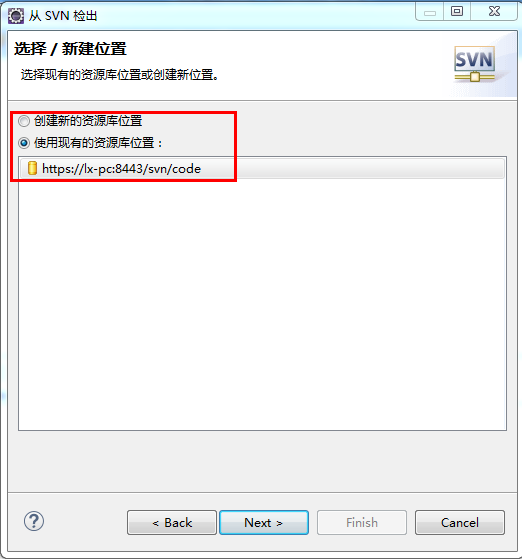
提交

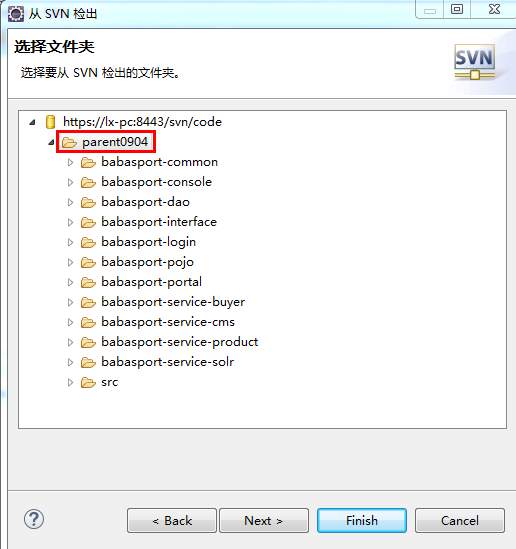




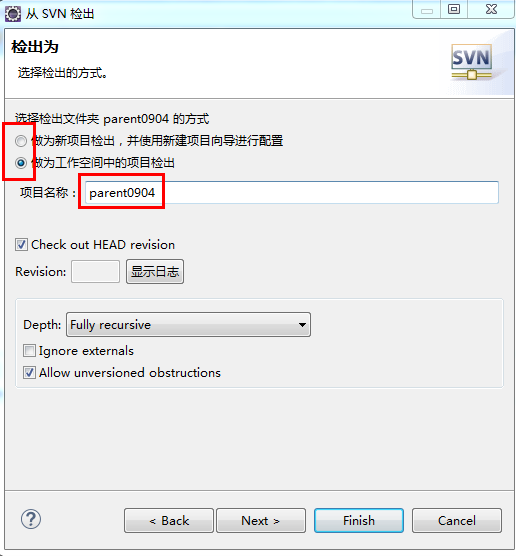
## 代码检出



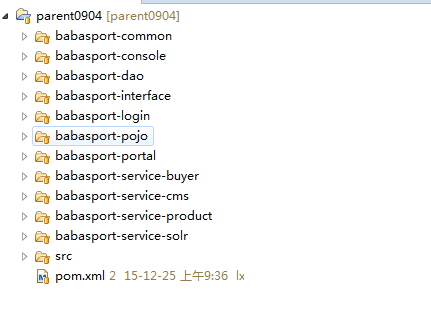




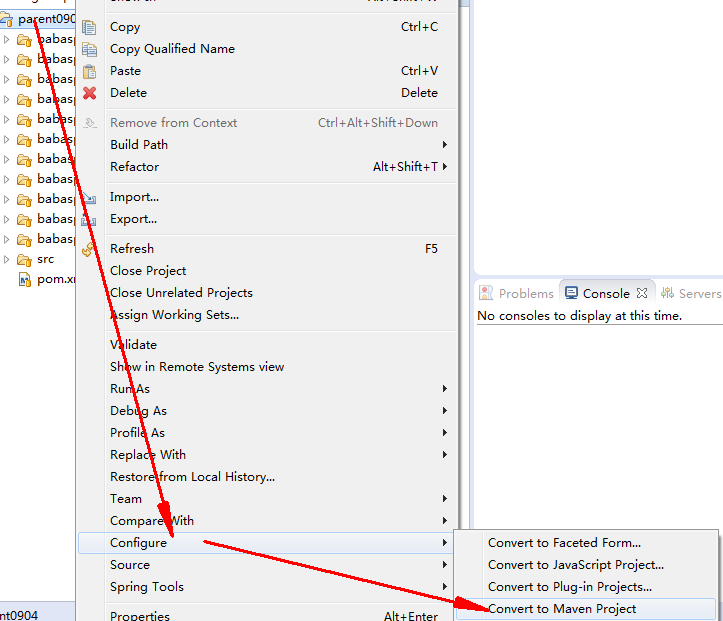
第三步



第四步

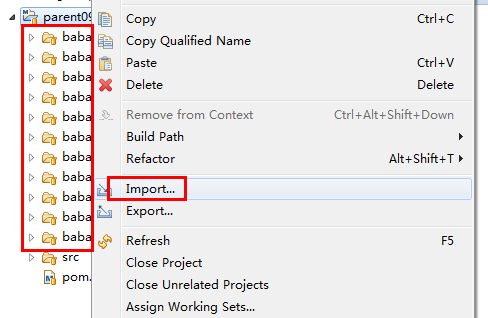


### 转成Maven工程

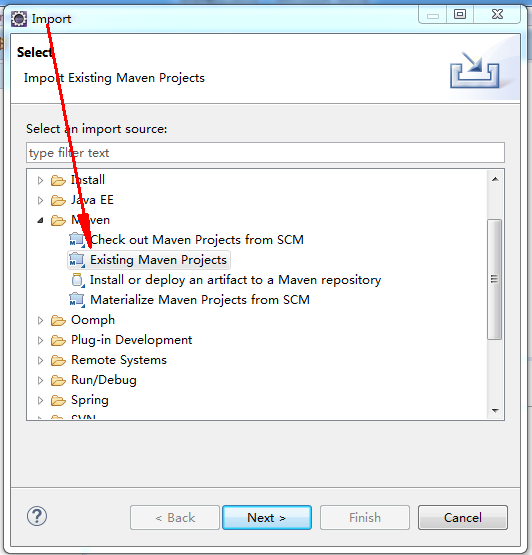


### 将父工程中的子项目作为存在的项目显示

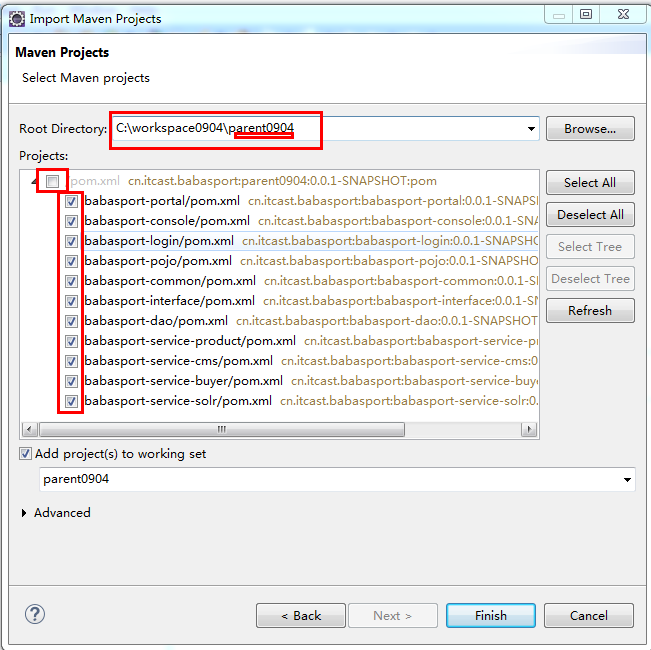
第一步：



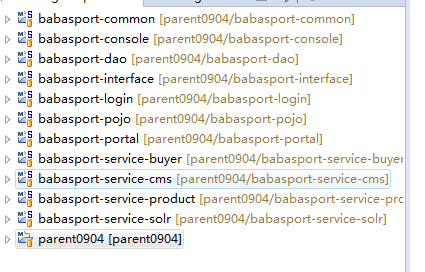
第二步；



第三步：



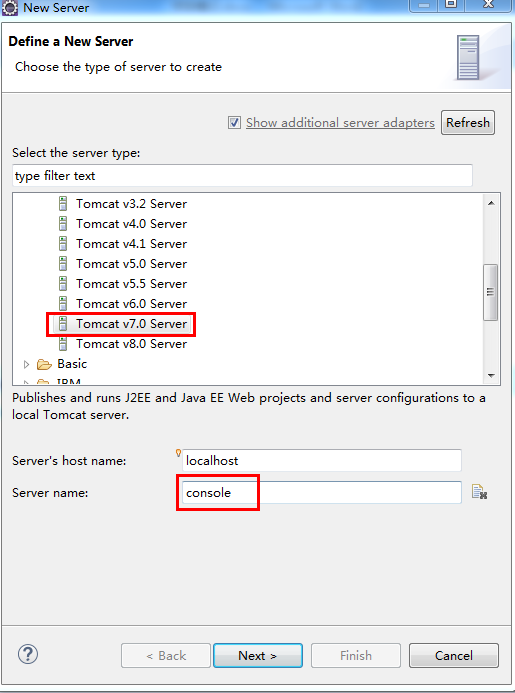
第四步

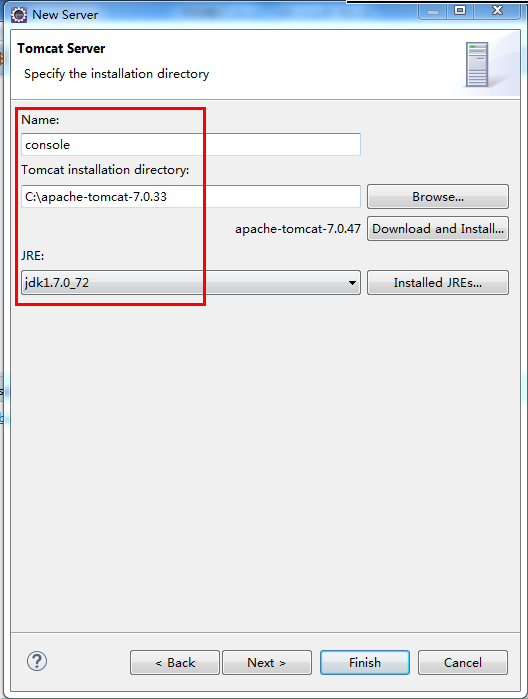


# 设置Tomcat 7 7个

干净的Tomcat7

## 创建Tomcat

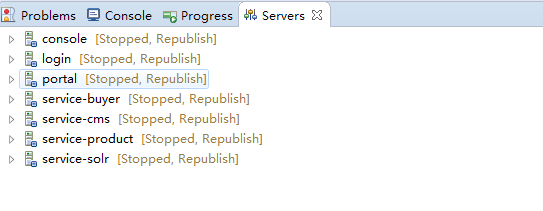


第二步；

第三步：

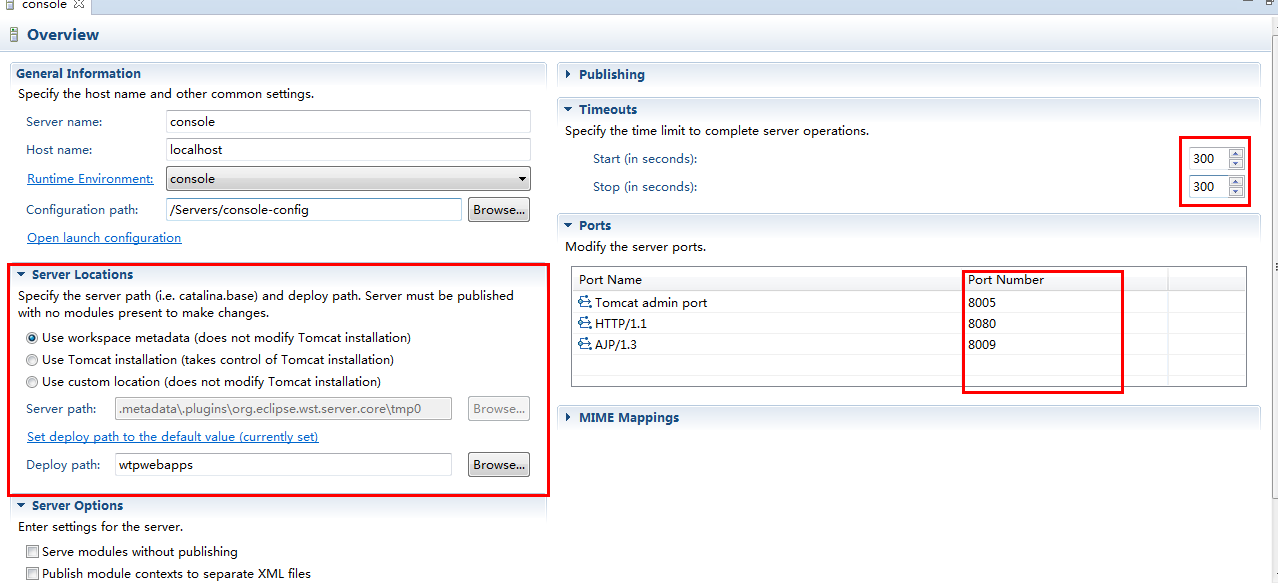


结果

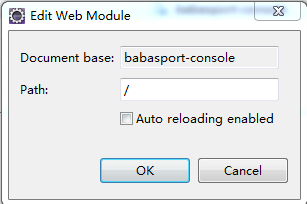


## 端口设置 8080---8086

console



第二步：



发布 路径

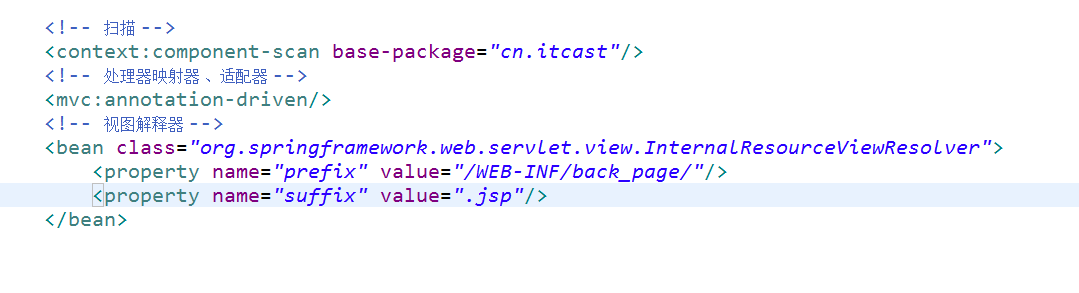
C:\workspace0904\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\babasport-console

# 配置Springmvc

## Web.xml 前端控制器



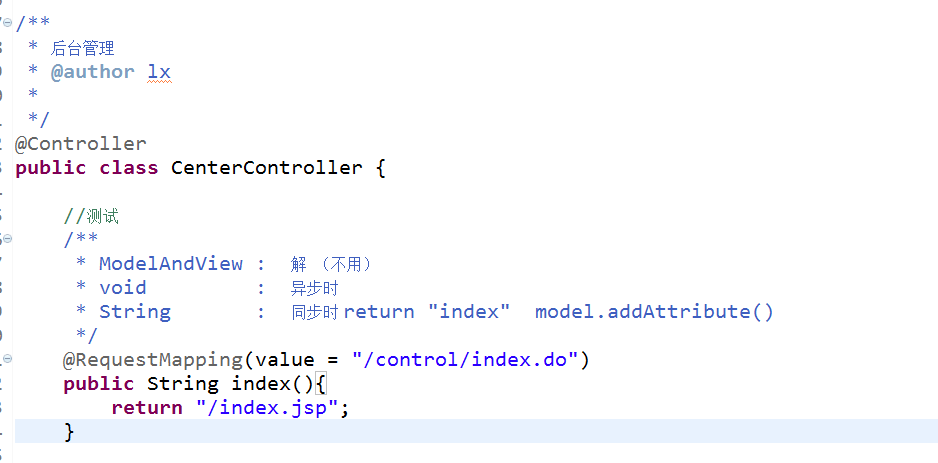
## Springmvc-console.xml 处理器映射器、适配器、视图解释器、扫描



# Springmvc测试

## Controller层

Index.jsp



# 整合Dubbo

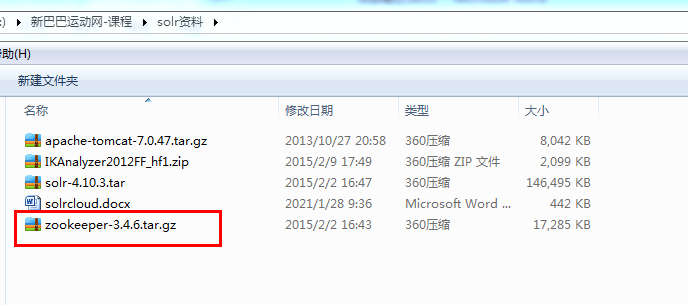
第一步：注册中心 中介 zookeeper （动物园管理员）

大数据（hadoop集群） 大象

Zookeeper solr dubbo

## 搭建zookeeper

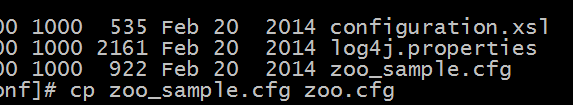
### 上传zookeeper包到centos上



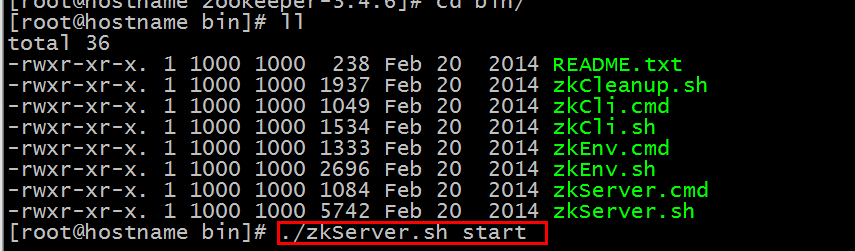
### 解压到指定目录



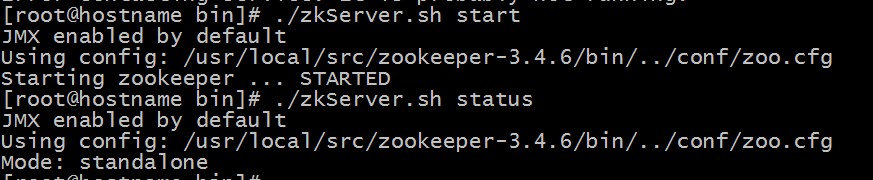
### 配置zookeeper的配置文件



### 启动

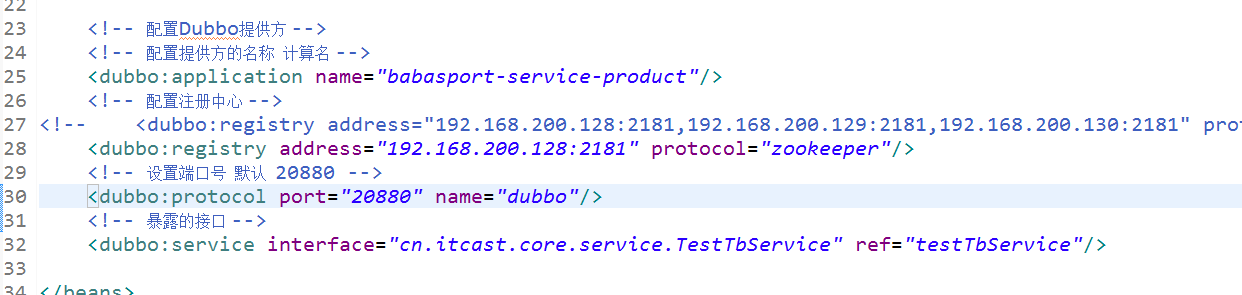


### 状态

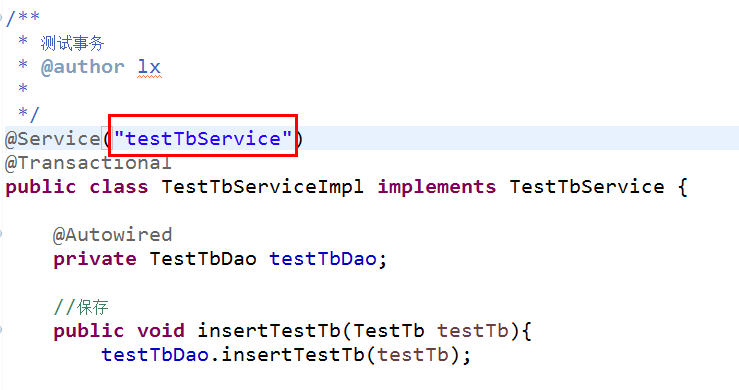


## 服务提供方

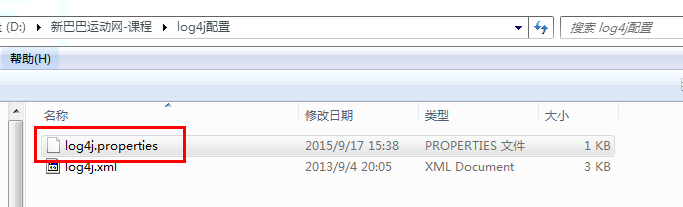
babasport-service-product



注意事项

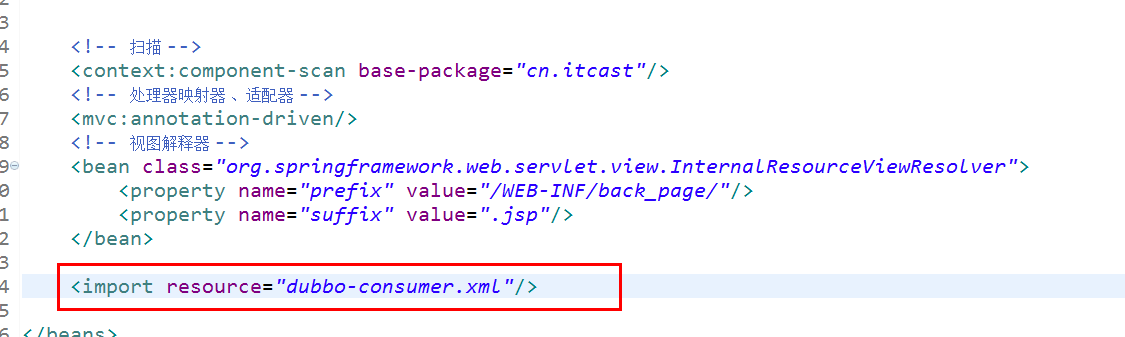


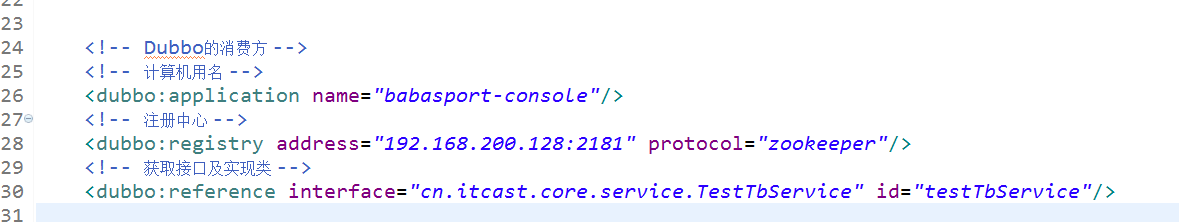
### 日志查看Dubbo注册信息



## 服务消费方

导入

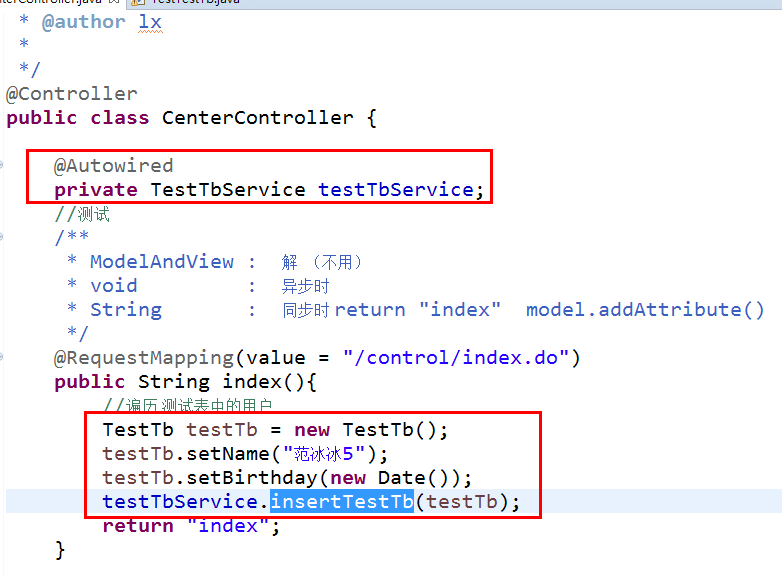




## 注意事项

使用Dubbo传递参数 实现序列化接口

## 测试Dubbo



# 后台搭建

## 内部转发开始

## 进入index.jsp

|  |
| --- |
| //入口页面 index.jsp  @RequestMapping(value = "/control/index.do")  public String index(){    return "index";  } |

## 进入top.jsp 、main.jsp

|  |
| --- |
| //top  @RequestMapping(value = "/top.do")  public String top(){    return "top";  }  //main  @RequestMapping(value = "/main.do")  public String main(){    return "main";  } |

## 进入身体的left right

|  |
| --- |
| //left  @RequestMapping(value = "/left.do")  public String left(){    return "left";  }  //right  @RequestMapping(value = "/right.do")  public String right(){    return "right";  } |

## 商品、订单模块跳转

|  |
| --- |
| //商品 身体  @RequestMapping(value = "/frame/product\_main.do")  public String product\_main(){  return "frame/product\_main";  }  //商品 身体 左  @RequestMapping(value = "/frame/product\_left.do")  public String product\_left(){  return "frame/product\_left";  }  //订单 身体  @RequestMapping(value = "/frame/order\_main.do")  public String order\_main(){  return "frame/order\_main";  }  //订单 身体 左  @RequestMapping(value = "/frame/order\_left.do")  public String order\_left(){  return "frame/order\_left";  } |

# 品牌管理

## 是否可用

在企业当中很多是不能删除的，商品，品牌，颜色，如果企业不想使用了，可以把是否可用字段设置成0：否 1：是

## 在新巴巴运动网的数据库中创建品牌表

品牌表：bbs\_brand (bbs babasport 简写) brand 是品牌的意思

Bigint : 10亿亿 （互联网项目）数据量非常大 int无法满足需要

Tingint : -128~127 只保存 0，1 足够用了 最小整数

|  |
| --- |
| DROP TABLE IF EXISTS `bbs\_brand`;  CREATE TABLE `bbs\_brand` (  `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT 'ID',  `name` varchar(40) NOT NULL COMMENT '名称',  `description` varchar(80) DEFAULT NULL COMMENT '描述',  `img\_url` varchar(80) DEFAULT NULL COMMENT '图片Url',  `web\_site` varchar(80) DEFAULT NULL COMMENT '品牌网址',  `sort` int(11) DEFAULT NULL COMMENT '排序:最大最排前',  `is\_display` tinyint(1) DEFAULT NULL COMMENT '是否可见 1:可见 0:不可见',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8 COMMENT='品牌'; |

## 创建与品牌表一一对应的JavaBean

由于品牌属于商品模块

实现序列化接口（把此JavaBean保存到远程的缓存服务器）建议实现

|  |
| --- |
| /\*\*  \* 品牌  \* @author lx  \*  \*/  public class Brand implements Serializable{  /\*\*  \* 默认的ID  \*/  private static final long serialVersionUID = 1L;    //品牌ID bigint  private Long id;  //品牌名称  private String name;  //描述  private String description;  //图片URL  private String imgUrl;  //排序 越大越靠前  private Integer sort;  //是否可用 0 不可用 1 可用  private Integer isDisplay;//is\_display  } |

## 条件参数字符串处理

前后去掉空格

本身就是空串，把空串视为null值

### 编写Springmvc转换器

传递的品牌名称在参数绑定之前就过滤上面的二种情况

### 配置Springmvc的配置文件

Springmvc-back.xml

|  |
| --- |
| <!-- 现成的标签 配置 处理器映射器 处理器适配器 -->  <mvc:annotation-driven conversion-service="conversionService"/>    <!-- 配置Springmvc的转换器 -->  <bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">  <!-- 配置多个转换器 -->  <property name="converters">  <list>  <bean class="去掉前后空格等处理类"/>  </list>  </property>  </bean> |

### 编写去掉前后空格等处理类

类名：CustomTrimConverter

指定二个泛型 分别是：传入的数据类型 别一个转换之后的数据类型

|  |
| --- |
| /\*\*  \* 处理传参 字符串  \* @author lx  \* String类型转String类型  \*  \*/  public class CustomTrimConverter implements Converter<String, String> {  //转换  public String convert(String source) {  // TODO Auto-generated method stub  try {  if(null != source){  //去掉前后空格  source = source.trim();  //本身就是空串  if(!"".equals(source)){  return source;  }  }    } catch (Exception e) {  // TODO: handle exception  }  return null;  }  } |

## 通过传递的参数查询数据库的数据

### 设置数据库查询数据

|  |
| --- |
| **INSERT INTO bbs\_brand VALUES ('1', '依琦莲', null, null, null, '99', '1');**  **INSERT INTO bbs\_brand VALUES ('2', '凯速（KANSOON）', null, null, null, 98, '1');**  **INSERT INTO bbs\_brand VALUES ('3', '梵歌纳（vangona）', null, null, null, 97, '1');**  **INSERT INTO bbs\_brand VALUES ('4', '伊朵莲', null, null, null, 96, '1');**  **INSERT INTO bbs\_brand VALUES ('5', '菩媞', null, null, null, 95, '1');**  **INSERT INTO bbs\_brand VALUES ('6', '丹璐斯', null, null, null, 94, '1');**  **INSERT INTO bbs\_brand VALUES ('7', '喜悦瑜伽', null, null, null, 93, '1');**  **INSERT INTO bbs\_brand VALUES ('8', '皮尔（pieryoga）', null, null, null, 92, '1');**  **INSERT INTO bbs\_brand VALUES ('9', '路伊梵（LEFAN）', null, null, null, 91, '1');**  **INSERT INTO bbs\_brand VALUES ('10', '金啦啦', null, null, null, 90, '0');**  **INSERT INTO bbs\_brand VALUES ('11', '来尔瑜伽（LaiErYoGA）', null, null, null, 89, '1');**  **INSERT INTO bbs\_brand VALUES ('12', '艾米达（aimida）', null, null, null, 88, '1');**  **INSERT INTO bbs\_brand VALUES ('13', '斯泊恩', null, null, null, 87, '1');**  **INSERT INTO bbs\_brand VALUES ('14', '康愫雅KSUA', null, null, null, 86, '1');**  **INSERT INTO bbs\_brand VALUES ('15', '格宁', null, null, null, 85, '1');**  **INSERT INTO bbs\_brand VALUES ('16', 'E奈尔（Enaier）', null, null, null, 84, '1');**  **INSERT INTO bbs\_brand VALUES ('17', '哈他', null, null, null, 83, '1');**  **INSERT INTO bbs\_brand VALUES ('18', '伽美斯（Jamars）', null, null, null, 82, '1');**  **INSERT INTO bbs\_brand VALUES ('19', '瑜伽树（yogatree）', null, null, null, 81, '1');**  **INSERT INTO bbs\_brand VALUES ('20', '兰博伊人（lanboyiren）', null, null, null, 80, '1');** |

### Mybatis框架（Dao接口及Mapper映射文件）

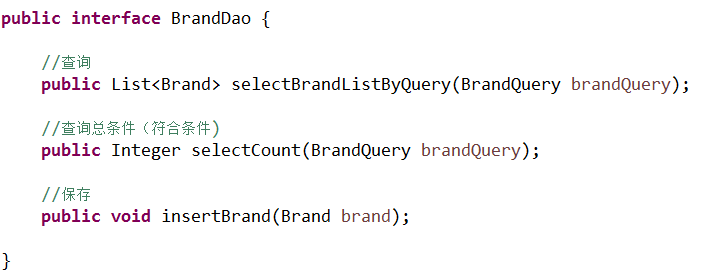
#### 创建BrandQuery

专用用于条件查询的对象

|  |
| --- |
| /\*\*  \* 品牌  \* @author lx  \*  \*/  public class BrandQuery implements Serializable{  /\*\*  \* 默认的ID  \*/  private static final long serialVersionUID = 1L;    //品牌ID bigint  private Long id;  //品牌名称  private String name;  //描述  private String description;  //图片URL  private String imgUrl;  //排序 越大越靠前  private Integer sort;  //是否可用 0 不可用 1 可用  private Integer isDisplay;//is\_display  此处省略 set get 方法……  //附加字段  //设置每页数的常量  private static final Integer DEFAULT\_PAGESIZE = 10;  //当前页 最小是1的整数  private Integer pageNo = 1;  //每页数  private Integer pageSize = DEFAULT\_PAGESIZE;  此处省略 set get 方法……  } |
|  |

#### Dao接口

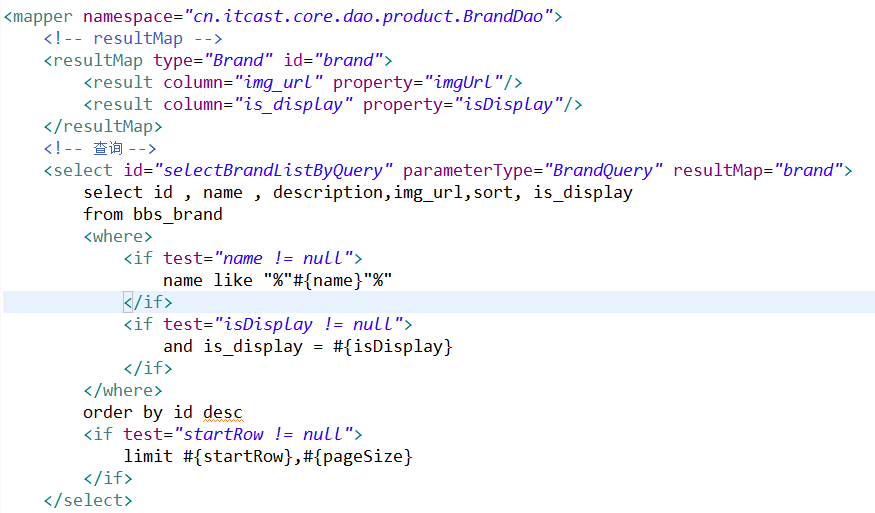
传递的参数尽量封装成对象 pageNo,Name,isDisplay 封装到BrandQuery对象



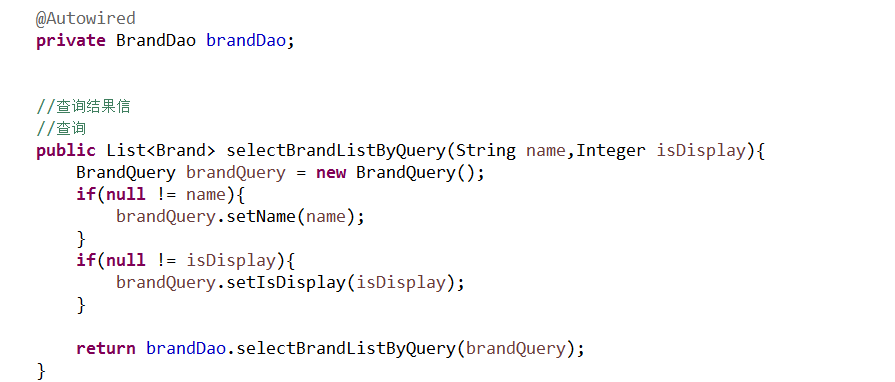
#### Mapper（Sql语句查询）

ResultType：要求数据库字段与JavaBean中的字段要一一对应（一样）

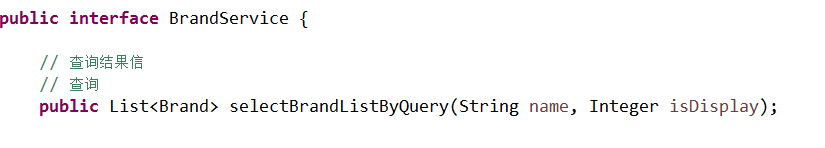
ResultMap：如果不一样，就可以使用ResultMap进行映射



### 编写BrandServiceImpl实现类



### 编写BrandService接口

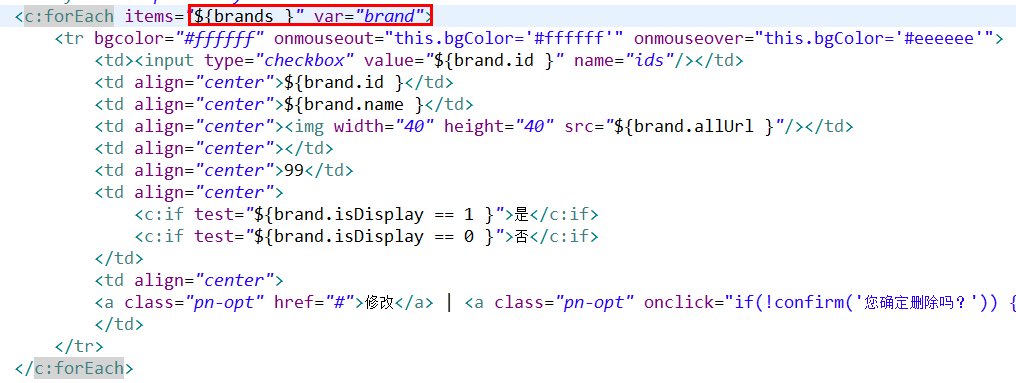


### 在BrandController层去调用BrandService接口

快速返回快捷键 Shift + Alt + L



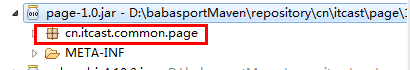
### 把拿到的数据遍历在品牌的列表页面上即可



### 品牌管理之分页查询

#### 返回分页对象

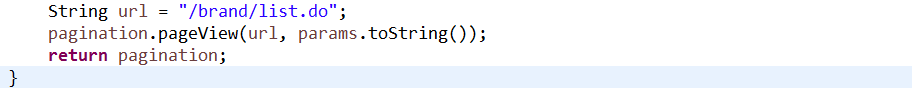
分页对象中有List<Brand>结果集 并包括分页（当前页、每页数、总条数）



百度、京东商城的分页

#### 在品牌Service层写返回分页对象的方法



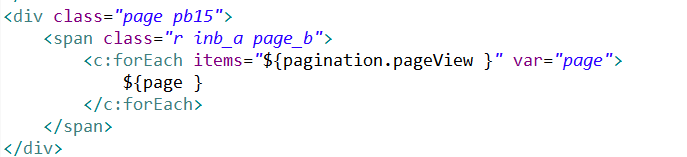


#### 在品牌Controller层



#### 页面显示

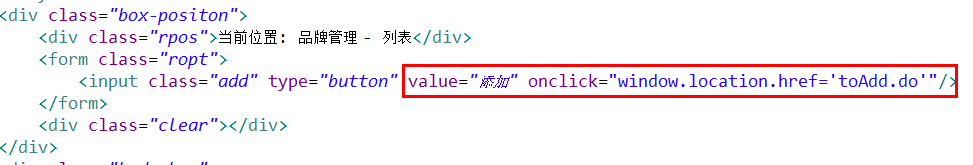




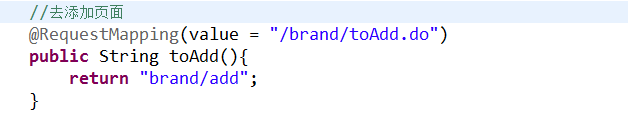
# 品牌添加

## 去添加页面

### 设置添加按钮的请求路径



### 编写BrandController 中的方法



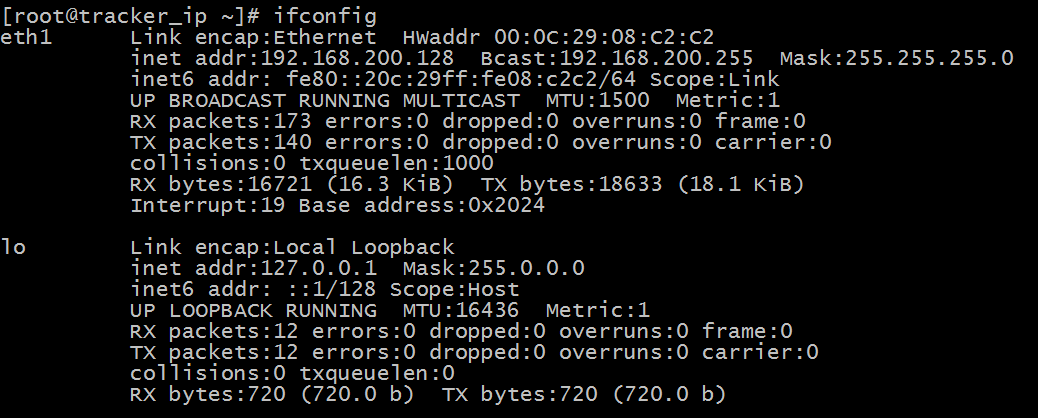
## 图片与服务器分离存储

### 专业的图片服务器 （企业）

分布式文件系统 FastDFS

好处：由多台机器组成，很多图片保存在不同机器来实现分布式，图片的访问速度快。

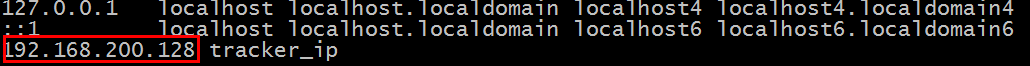
### 查看自己机器的IP



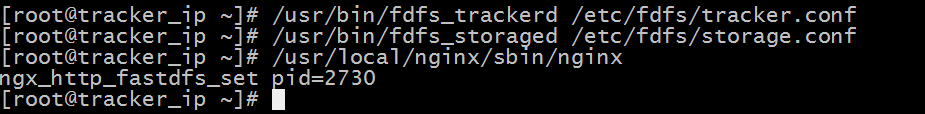
### 设置hosts文件



把红线内容改成自己的机器IP



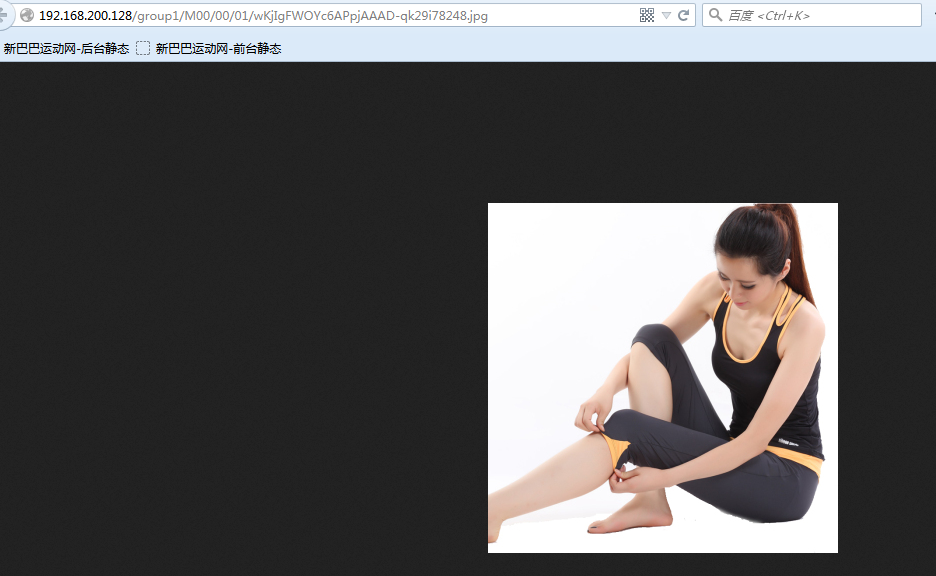
### 启动Tracker、Stoage、Nginx



### 测试是否启动成功

[http://192.168.200.128/group1/M00/00/01/wKjIgFWOYc6APpjAAAD-qk29i78248.jpg](http://自己机器的IP/group1/M00/00/01/wKjIgFWOYc6APpjAAAD-qk29i78248.jpg)

启动成功显示下面的图片



## FastDFS的Java接口连接图片服务器

### 配置FastDFS的文件fdfs\_client.conf

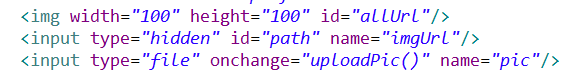
|  |
| --- |
| # connect timeout in seconds  # default value is 30s  connect\_timeout=30  # network timeout in seconds  # default value is 30s  network\_timeout=60  # the base path to store log files  base\_path=/home/fastdfs  # tracker\_server can ocur more than once, and tracker\_server format is  # "host:port", host can be hostname or ip address  tracker\_server=192.168.200.128:22122  #tracker\_server=192.168.101.4:22122  #standard log level as syslog, case insensitive, value list:  ### emerg for emergency  ### alert  ### crit for critical  ### error  ### warn for warning  ### notice  ### info  ### debug  log\_level=info  # if use connection pool  # default value is false  # since V4.05  use\_connection\_pool = false  # connections whose the idle time exceeds this time will be closed  # unit: second  # default value is 3600  # since V4.05  connection\_pool\_max\_idle\_time = 3600  # if load FastDFS parameters from tracker server  # since V4.05  # default value is false  load\_fdfs\_parameters\_from\_tracker=false  # if use storage ID instead of IP address  # same as tracker.conf  # valid only when load\_fdfs\_parameters\_from\_tracker is false  # default value is false  # since V4.05  use\_storage\_id = false  # specify storage ids filename, can use relative or absolute path  # same as tracker.conf  # valid only when load\_fdfs\_parameters\_from\_tracker is false  # since V4.05  storage\_ids\_filename = storage\_ids.conf  #HTTP settings  http.tracker\_server\_port=80  #use "#include" directive to include HTTP other settiongs  ##include http.conf |

### FastDFSUtils

|  |
| --- |
| /\*\*  \* 连接分布式文件系统的Java接口  \* @author lx  \*  \*/  public class FastDFSUtil {    //上传图片到分布式文件系统  public static String uploadPic(byte[] pic,String name,Long size) throws Exception{  //设置tracker的ip 读取fdfs\_client.conf  ClassPathResource resource = new ClassPathResource("fdfs\_client.conf");  ClientGlobal.init(resource.getClassLoader().getResource("fdfs\_client.conf").getPath());  //创建客户端  TrackerClient trackerClient = new TrackerClient();  TrackerServer trackerServer = trackerClient.getConnection();    StorageServer storageServer = null;    StorageClient1 storageClient1 = new StorageClient1(trackerServer,storageServer);    // jpg png  String ext = FilenameUtils.getExtension(name);    NameValuePair[] meta\_list = new NameValuePair[3];  meta\_list[0] = new NameValuePair("filename",name);  meta\_list[1] = new NameValuePair("fileext",ext);  meta\_list[2] = new NameValuePair("filesize",String.valueOf(size));  //路径  String path = storageClient1.upload\_file1(pic, ext, meta\_list);      return path;  }  } |

## 上传图片

### 浏览按钮的事件



### 上传图片的方法（jquery）异步上传

Jquery 框架 jquery.form.js 原理：js模拟出一个Form表单（提交图片到服务器）

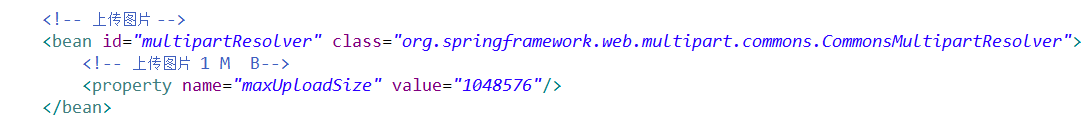
引入jquery

引入jquery.form.js



### 配置Springmvc 上传图片

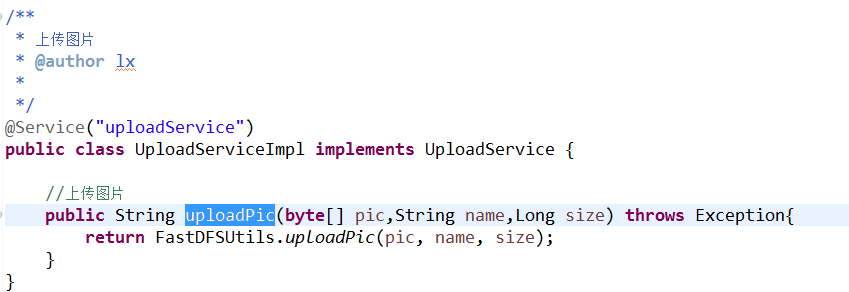
在springmvc-back.xml



### 接收上传的文件（图片）



### 创建UploadServiceImpl实现类



## 品牌提交保存

### 设置页面属性名

### BrandController

|  |
| --- |
| //添加保存  @RequestMapping(value = "/brand/add.do")  public String add(Brand brand){    //保存  brandService.insertBrandById(brand);    return "redirect:/brand/list.do";  } |

### BrandDao

|  |
| --- |
| //保存  public void insertBrandById(Brand brand); |

### BrandDao.xml

|  |
| --- |
| <!-- 添加保存 -->  <insert id="insertBrandById" parameterType="Brand">  insert into bbs\_brand  <trim prefix="(" suffix=")">  id,name,description,img\_url,sort,is\_display  </trim>  values  <trim prefix="(" suffix=")">  #{id},#{name},#{description},#{imgUrl},#{sort},#{isDisplay}  </trim>  </insert> |

### BrandServiceImpl

|  |
| --- |
| public void insertBrandById(Brand brand) {  // TODO Auto-generated method stub  brandDao.insertBrandById(brand);  } |

## 删除（批量）

### 全选

|  |
| --- |
| <script type="text/javascript">  //全选  function checkBox(name,checked){  //获取全部复选  $("input[name='" + name + "']").attr("checked",checked);  }  </script> |

### 删除（事件optDelete()）

### Form表单（复选框包起来）

### Jquery提交此Form

|  |
| --- |
| //删除 （批量）  function optDelete(name,isDisplay,pageNo){  //判断 请至少选择一个  var size = $("input[name='ids']:checked").size();  if(size == 0){  alert("请至少选择一个");  return;  }  //提示 你确定删除吗  if(!confirm("你确定删除吗?")){  return;  }  //提交Form  $("#jvForm").attr("action","/brand/deletes.do?name=" + name + "&isDisplay=" + isDisplay + "&pageNo=" + pageNo);  $("#jvForm").attr("method","post").submit();    } |

### BrandController 写删除方法

|  |
| --- |
| //删除  @RequestMapping(value = "/brand/deletes.do")  public String deletes(Long[] ids,Integer pageNo,String name,Integer isDisplay,Model model){  //删除  brandService.deleteBrandByIds(ids);  //判断  if(null != pageNo){  model.addAttribute("pageNo", pageNo);  }  //判断  if(null != name){  model.addAttribute("name", name);  }  //判断  if(null != isDisplay){  model.addAttribute("isDisplay", isDisplay);  }  return "redirect:/brand/list.do";  } |

### BrandDao 写删除方法

|  |
| --- |
| //删除  public void deleteBrandByIds(Long[] ids); |

### BrandDao.xml写删除的Sql

|  |
| --- |
| <!-- 删除 批量 (1,2,3,4,5)-->  <delete id="deleteBrandByIds" parameterType="Long[]">  delete from bbs\_brand  where id in  <foreach collection="array" item="id" open="(" close=")" separator=",">  #{id}  </foreach>  </delete> |

### BrandServiceImpl的删除方法

|  |
| --- |
| public void deleteBrandByIds(Long[] ids) {  // TODO Auto-generated method stub  brandDao.deleteBrandByIds(ids);  } |

## 删除（单删） 同学们完成

## 修改

### 去修改页面

通过ID查询Brand对象

#### 设置页面的路径

/brand/toEdit.do?id=${brand.id}

### 回显数据到修改页面上

丢掉ID 放在隐藏域中

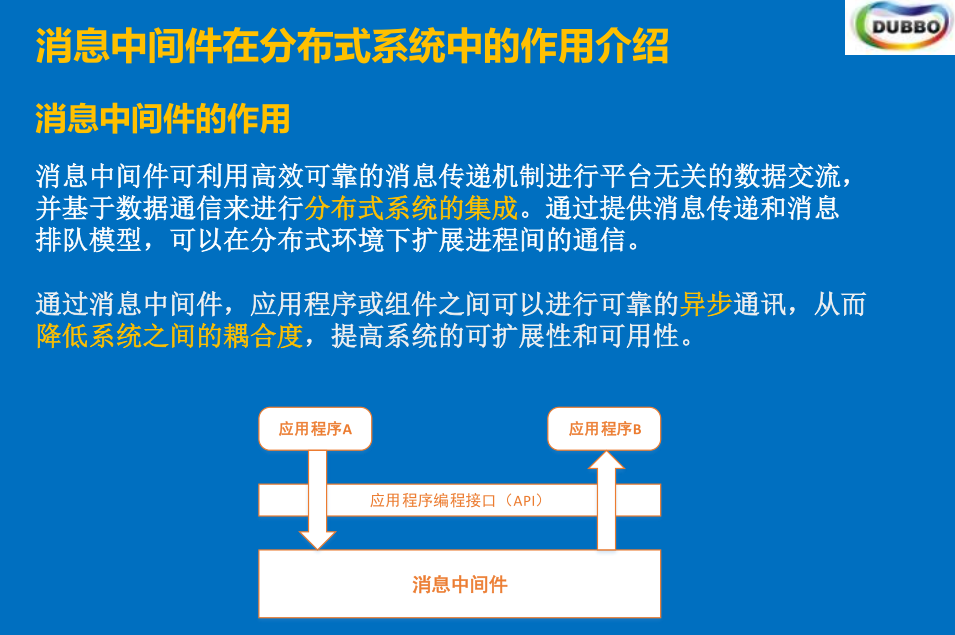
### 提交按钮 （修改数据）

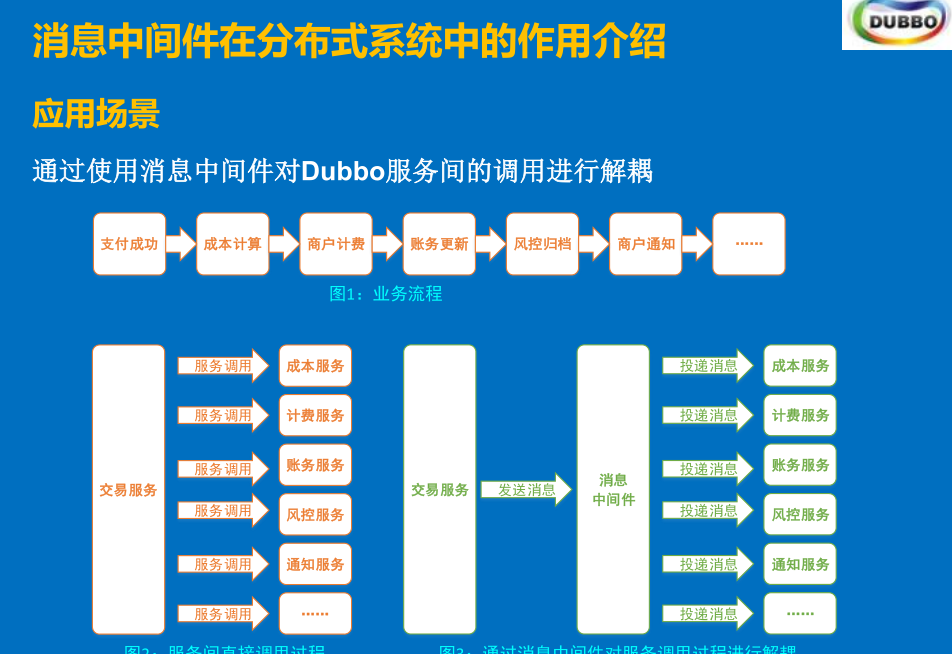
# 消息队列ActiveMQ

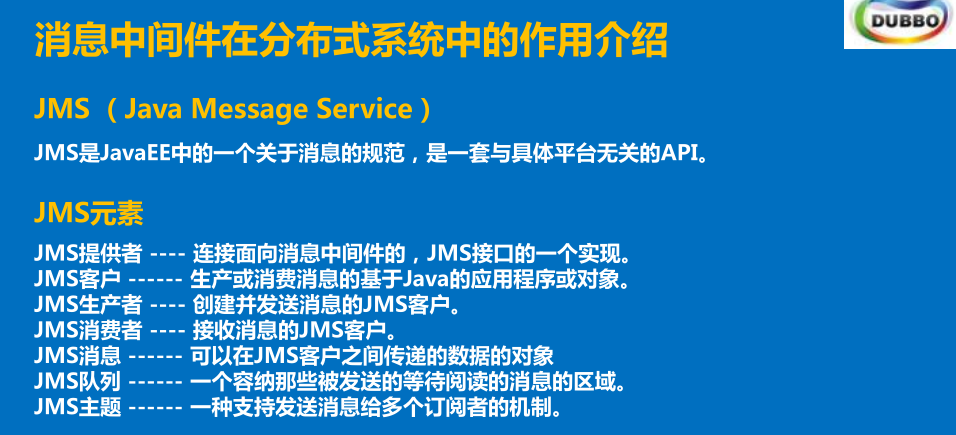
## 什么是消息队列

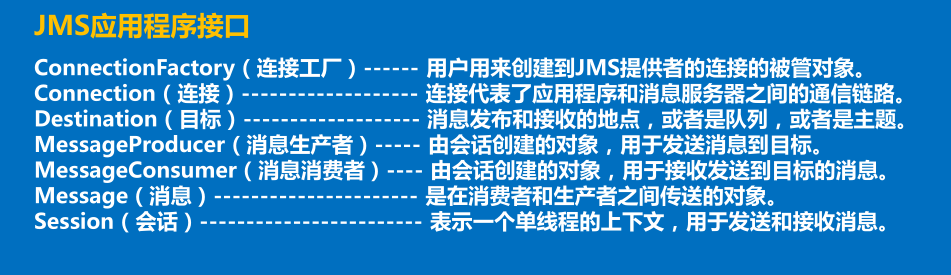


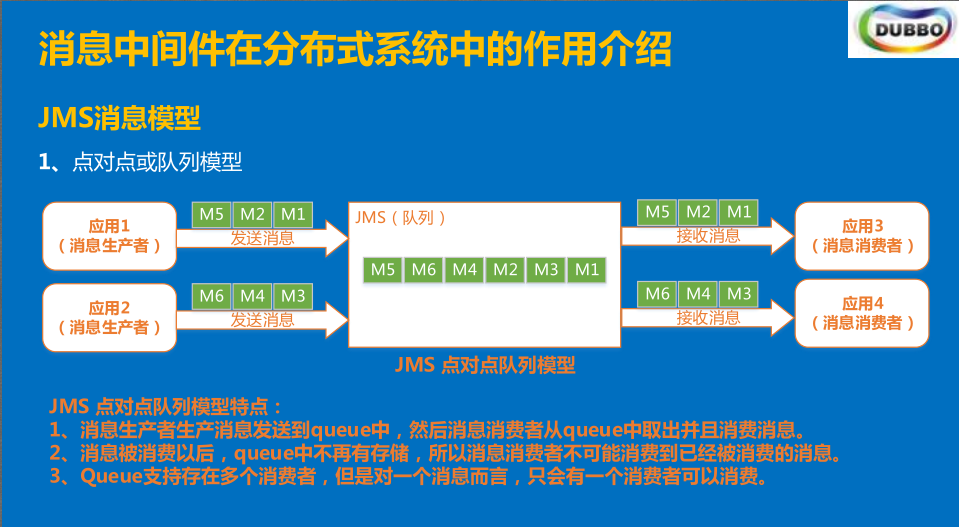
## 为什么使用MQ

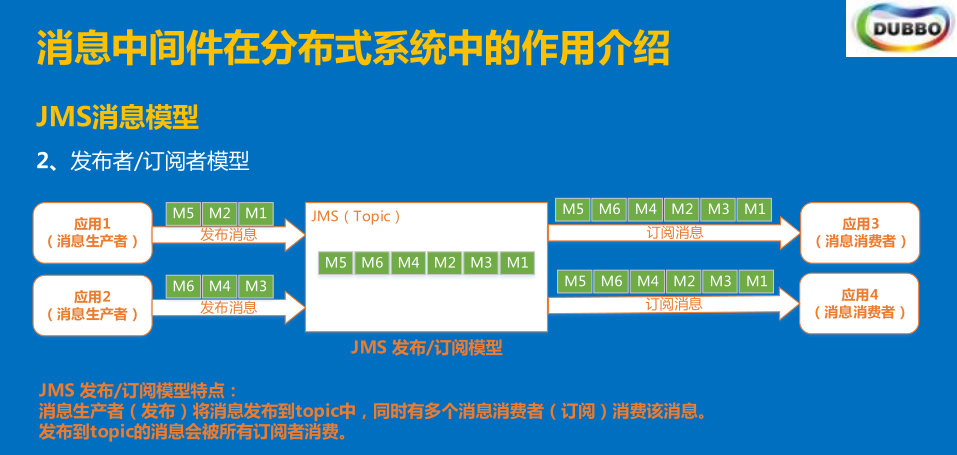














## ActiveMQ发送消息

### 配置ActiveMQ

|  |
| --- |
| <!-- 配置Mq -->  <!-- 配置Mq工厂 -->  <bean id="activeMQConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">  <property name="brokerURL" value="tcp://192.168.200.128:61616"/>  <property name="userName" value="admin"/>  <property name="password" value="admin"/>  </bean>  <!-- 配置Mq工厂的连接池 -->  <bean id="pooledConnectionFactoryBean" class="org.apache.activemq.pool.PooledConnectionFactoryBean">  <property name="connectionFactory" ref="activeMQConnectionFactory"/>  <property name="maxConnections" value="2"/>  </bean>    <!-- 配置由Spring管理工厂 -->  <bean id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory">  <property name="targetConnectionFactory" ref="pooledConnectionFactoryBean"/>  </bean>  <!-- 发送消息的模板 -->  <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">  <property name="connectionFactory" ref="connectionFactory"/>  <property name="defaultDestinationName" value="brandToRedis"/>  </bean> |

### 发送品牌对象

|  |
| --- |
| @Autowired  private JmsTemplate jmsTemplate;  // 修改  public void updateBrandById(final Brand brand) {  jmsTemplate.send(new MessageCreator() {  @Override  public Message createMessage(Session session) throws JMSException {  // TODO Auto-generated method stub  return session.createTextMessage(JSONObject.toJSONString(brand));  }  });  brandDao.updateBrandById(brand);  } |

### 接收ActiveMQ

第一步：创建babasport-queue项目

第二步：配置ActiveMQ

|  |
| --- |
| <!-- 配置Mq -->  <!-- 配置Mq工厂 -->  <bean id="activeMQConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">  <property name="brokerURL" value="tcp://192.168.200.128:61616"/>  <property name="userName" value="admin"/>  <property name="password" value="admin"/>  </bean>  <!-- 配置Mq工厂的连接池 -->  <bean id="pooledConnectionFactoryBean" class="org.apache.activemq.pool.PooledConnectionFactoryBean">  <property name="connectionFactory" ref="activeMQConnectionFactory"/>  <property name="maxConnections" value="2"/>  </bean>    <!-- 配置由Spring管理工厂 -->  <bean id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory">  <property name="targetConnectionFactory" ref="pooledConnectionFactoryBean"/>  </bean>  <!-- 发送消息的模板 -->  <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">  <property name="connectionFactory" ref="connectionFactory"/>  <property name="defaultDestinationName" value="brandToRedis"/>  </bean>    <!-- 消息队列的目的地 -->  <bean id="activeMQQueue" class="org.apache.activemq.command.ActiveMQQueue">  <constructor-arg>  <value>brandToRedis</value>  </constructor-arg>  </bean>    <!-- 自定义接收消息 通过Spring提供的监听器 -->  <bean id="messageListener" class="cn.itcast.core.message.CustomMessageListener"/>    <!-- Spring监听容器 -->  <bean id="SessionAwareMessageListenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer">  <!-- 监听消息队列的目的地 -->  <property name="destination" ref="activeMQQueue"/>  <!-- 连接MQ的工厂 -->  <property name="connectionFactory" ref="connectionFactory"/>  <!-- 监听到的消息 接收信息的类 -->  <property name="messageListener" ref="messageListener"/>  </bean> |

第三步：自定义消息监听器

|  |
| --- |
| import javax.jms.JMSException;  import javax.jms.Message;  import javax.jms.Session;  import org.apache.activemq.command.ActiveMQTextMessage;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jms.listener.SessionAwareMessageListener;  import com.alibaba.fastjson.JSONObject;  import cn.itcast.core.bean.product.Brand;  import cn.itcast.core.service.product.BrandService;  import redis.clients.jedis.Jedis;  /\*\*  \* 通过Spring提供的消息监听器 接收Mq中的消息  \* @author lx  \*  \*/  public class CustomMessageListener implements SessionAwareMessageListener<Message> {    @Autowired  private Jedis jedis;  @Override  public synchronized void onMessage(Message message, Session session) throws JMSException {  // TODO Auto-generated method stub  ActiveMQTextMessage m = (ActiveMQTextMessage)message;    String text = m.getText();    System.out.println(text);    Brand brand = JSONObject.parseObject(text, Brand.class);    //保存品牌到Redis中  jedis.hset("brand", String.valueOf(brand.getId()),brand.getName());    }  } |

# 商品管理

## 分析一下商品表关系（重要字段）

七张表：重点是二张表（商品表、库存表（最小销售单元）SKU） 、属性表、颜色表、图片表、品牌表、类型表

## Mybatis逆向工程

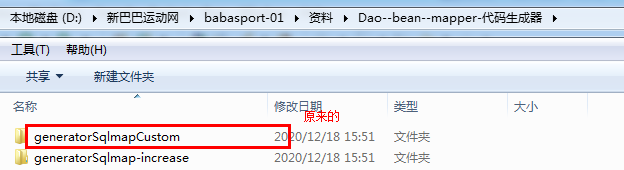
在接下来的开发中，不再写Sql语句

Dao接口

Mapper文件

JavaBean(与数据库一一对应的)

## 本次开发的逆向工程增强版（相对于之前的逆向工程）



### cn.itcast.core.bean.BbsBrand

JavaBean的名称更改成Brand

### cn.itcast.core.bean.BbsBrandExample

名称更改成BrandQuery

### JavaBean 中有中文注解

### JavaBean生成序列化接口

### JavaBean中有toString 方法

### JavaBean生成在子包（商品包）

### BrandMapper.java 生成BrandDao.java

### BrandMapper.xml生成BrandDao.xml

### 增强版支持物理分页

### 指定字段查询

Select id,name,description,…… from bbs\_brand

全表字段查询（表所有字段全查出来）

有时候只需要查询部分字段。

### 注意：数据库tingint类型最小整数

-128~127

Mybatis逆向工程把tingint 类型 默认生成JavaBean是Boolean类型

## 逆向工程Junit测试

|  |
| --- |
| package cn.itcast;  import java.util.List;  import org.junit.Test;  import org.springframework.beans.factory.annotation.Autowired;  import cn.itcast.common.junit.SpringJunitTest;  import cn.itcast.core.bean.product.Product;  import cn.itcast.core.bean.product.ProductQuery;  import cn.itcast.core.dao.product.ProductDao;  /\*\*  \* 测试增强版Mybatis逆向工程  \* @author lx  \*  \*/  public class TestProduct extends SpringJunitTest{  @Autowired  private ProductDao productDao;  //测试商品  @Test  public void testProduct() throws Exception {  //查询  //指定字段查询 . 物理分页 、 设置多个条件 、 id排序  // select \* from bbs\_brand <where> and is\_show = 1  ProductQuery example = new ProductQuery();  //设置多个条件  example.createCriteria().andIsShowEqualTo(true);  //指定字段查询 id name  example.setFields("id,name");  //按照id排序  example.setOrderByClause("id desc");  //物理分页 设置页号 、每页数  example.setPageNo(2);  example.setPageSize(3);      //int countByExample = productDao.countByExample(example);    //System.out.println(countByExample);    //查询一条数据  /\* Product product = productDao.selectByPrimaryKey(275L);  System.out.println(product);\*/  //查询多条数据 结果集  List<Product> products = productDao.selectByExample(example);  for (Product p : products) {  System.out.println(p);  }    //保存  //修改  //删除    }    } |

## 商品列表查询（带条件+分页）

### 入口（商品、商品管理、查询）

/product/list.do

### 创建ProductController

|  |
| --- |
| /\*\*  \* 商品管理  \* 列表  \* 添加  \* 上架  \* @author lx  \*  \*/  @Controller  public class ProductController {  @Autowired  private ProductService productService;  @Autowired  private BrandService brandService;  @Autowired  private TypeService typeService;  @Autowired  private ColorService colorService;  //去列表页面  @RequestMapping(value = "/product/list.do")  public String list(Integer pageNo,String name,Long brandId,Boolean isShow,Model model){    List<Brand> brands = brandService.selectBrandListByQuery(null, 1);  model.addAttribute("brands", brands);      if(null != name){  model.addAttribute("name", name);    }  if(null != brandId){    model.addAttribute("brandId", brandId);    }  //上下架  if(null != isShow){    model.addAttribute("isShow", isShow);  }else{  model.addAttribute("isShow", false);  }  Pagination pagination = productService.selectPaginationByQuery(pageNo, name, brandId, isShow);  model.addAttribute("pagination", pagination);  return "product/list";    } |

### 创建ProductServiceImpl

|  |
| --- |
| @Service("productService")  @Transactional  public class ProductServiceImpl implements ProductService{  @Autowired  private ProductDao productDao;  @Autowired  private ImgDao imgDao;  //返回分页对象  public Pagination selectPaginationByQuery(Integer pageNo  ,String name,Long brandId,Boolean isShow){  ProductQuery productQuery = new ProductQuery();    productQuery.setOrderByClause("id desc");    productQuery.setPageNo(Pagination.cpn(pageNo));  productQuery.setPageSize(3);    StringBuilder params = new StringBuilder();    Criteria createCriteria = productQuery.createCriteria();  if(null != name){  createCriteria.andNameLike("%" + name + "%");    params.append("name=").append(name);  }  if(null != brandId){    createCriteria.andBrandIdEqualTo(brandId);    params.append("&brandId=").append(brandId);  }  //上下架  if(null != isShow){  createCriteria.andIsShowEqualTo(isShow);  params.append("&isShow=").append(isShow);  }else{  createCriteria.andIsShowEqualTo(false);  params.append("&isShow=").append(false);  }    Pagination pagination = new Pagination(  productQuery.getPageNo(),  productQuery.getPageSize(),  productDao.countByExample(productQuery)  );    List<Product> products = productDao.selectByExample(productQuery);  for (Product product : products) {  ImgQuery imgQuery = new ImgQuery();  imgQuery.createCriteria().andProductIdEqualTo(product.getId()).andIsDefEqualTo(true);  List<Img> imgs = imgDao.selectByExample(imgQuery);  product.setImg(imgs.get(0));  }    pagination.setList(products);    String url = "/product/list.do";  pagination.pageView(url, params.toString());    return pagination;    } |

### 页面回显示

|  |
| --- |
| <form action="/product/list.do" method="post" style="padding-top:5px;">  名称: <input type="text" name="name" value="${name }"/>  <select name="brandId">  <option value="">请选择品牌</option>  <c:forEach items="${brands }" var="brand">  <option value="${brand.id }" <c:if test="${brand.id == brandId}">selected="selected"</c:if>>${brand.name }</option>  </c:forEach>  </select>  <select name="isShow">  <option value="true" <c:if test="${isShow}">selected="selected"</c:if>>上架</option>  <option value="false" <c:if test="${!isShow}">selected="selected"</c:if>>下架</option>  </select>  <input type="submit" class="query" value="查询"/>  </form> |

# 商品添加

## 去添加页面并加载品牌、类型、材质、颜色

设置添加请求路径 /product/toAdd.do

|  |
| --- |
| //去商品添加页面  @RequestMapping(value = "/product/toAdd.do")  public String toAdd(Model model){  //类型  List<Type> types = typeService.selectTypeList();  model.addAttribute("types", types);  //品牌  List<Brand> brands = brandService.selectBrandListByQuery(null, 1);  model.addAttribute("brands", brands);  //颜色  List<Color> colors = colorService.selectColorList();  model.addAttribute("colors", colors);    return "product/add";      } |

## Fck上传图片

### Fck上传图片路径设置

|  |
| --- |
| var fck = new FCKeditor("productdesc");  fck.BasePath = "/res/fckeditor/";  fck.Height = 400 ;  //上传路径 设置  fck.Config["ImageUploadURL"] = "/upload/uploadFck.do";  fck.ReplaceTextarea(); |

### 保存图片到FastDFS

|  |
| --- |
| //上传Fck  @RequestMapping(value = "/upload/uploadFck.do")  public void uploadFck(HttpServletRequest request,HttpServletResponse response) throws Exception{    //强转原生态的request  MultipartRequest mr = (MultipartRequest)request;  //多张图片 本次就一张图片  Map<String, MultipartFile> fileMap = mr.getFileMap();    Set<Entry<String, MultipartFile>> entrySet = fileMap.entrySet();  for (Entry<String, MultipartFile> entry : entrySet) {  MultipartFile pic = entry.getValue();    String path = uploadService.uploadPic(pic.getBytes(), pic.getOriginalFilename(), pic.getSize());    UploadResponse ok = UploadResponse.getOK(Constants.IMG\_URL + path);  //返回对象  response.getWriter().print(ok);    }  } |

### 返回路径给Fck

采用Fck的jar包中response.UploadResponse类来完成路径的返回

|  |
| --- |
| //返回Fck的上传路径  UploadResponse ok = UploadResponse.getOK(url);    response.getWriter().print(ok); |

## 提交商品信息

### 设置页面字段

### 商品提交ProductController接收商品对象

|  |
| --- |
| //商品添加  @RequestMapping(value = "/product/add.do")  public String add(Product product,Model model){    //保存  productService.insertProduct(product);  return "redirect:/product/list.do";  } |

### 商品保存

保存商品、保存图片、保存Sku（最小销售单元）

|  |
| --- |
| //商品添加  public void insertProduct(Product product){  //设置上下架 默认要求是下架  product.setIsShow(false);  //设置是否删除 不删除 1  product.setIsDel(true);  //保存商品  productDao.insertSelective(product);  //图片保存  Img img = product.getImg();  //设置商品ID  img.setProductId(product.getId());  //设置是否是默认  img.setIsDef(true);  //保存图片表  imgDao.insertSelective(img);  //保存Sku  // 商品ID 、颜色、尺码 （最小销售）  for(String color : product.getColors().split(",")){  //9 转Long  //创建Sku对象  Sku sku = new Sku();  //设置颜色  sku.setColorId(Long.parseLong(color));  //设置商品ID  sku.setProductId(product.getId());  for(String size : product.getSizes().split(",")){  //设置尺码  sku.setSize(size);  //设置运费  sku.setDeliveFee(10f);  //市场价格  sku.setMarketPrice(0f);  //售价  sku.setPrice(0f);  //库存  sku.setStock(0);  //默认200件 可更改  sku.setUpperLimit(200);  //遍历尺码 处 保存Sku  skuDao.insertSelective(sku);  }    }  } |

### 设置返回商品ID

|  |
| --- |
| <insert id="insertSelective" parameterType="cn.itcast.core.bean.product.Product" useGeneratedKeys="true" keyProperty="id"> |

### 修复商品列表问题

#### 商品列表页面（图片的请求路径设置）

##### 页面上

|  |
| --- |
| <img width="50" height="50" src="${product.img.allUrl }"/> |

##### 加载图片

|  |
| --- |
| //返回分页对象  public Pagination selectPaginationByQuery(ProductQuery productQuery){  //创建分页对象  //当前页  //每页数  //总条数(符合条件的）  Pagination pagination = new Pagination(  productQuery.getPageNo(),  productQuery.getPageSize(),  productDao.countByExample(productQuery)  );  //超过最大当前页时，使用最后一页  productQuery.setPageNo(pagination.getPageNo());    //把商品结果中设置图片数据（默认）  List<Product> products = productDao.selectByExample(productQuery);  //遍历商品  for (Product product : products) {    //查询商品ID对应的默认图片  ImgQuery imgQuery = new ImgQuery();  //设置商品ID  imgQuery.createCriteria()  .andProductIdEqualTo(product.getId())  .andIsDefEqualTo(true);  //集合中只有一张图片  List<Img> imgs = imgDao.selectByExample(imgQuery);  product.setImg(imgs.get(0));  }  //设置结果集  pagination.setList(products);    return pagination;  } |

##### Img对象中设置全路径

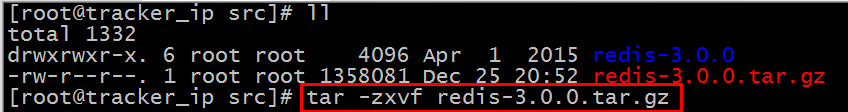
|  |
| --- |
| //全路径  public String getAllUrl(){  return Constants.IMG\_URL + url;  } |

## Redis服务器的搭建（单版机）

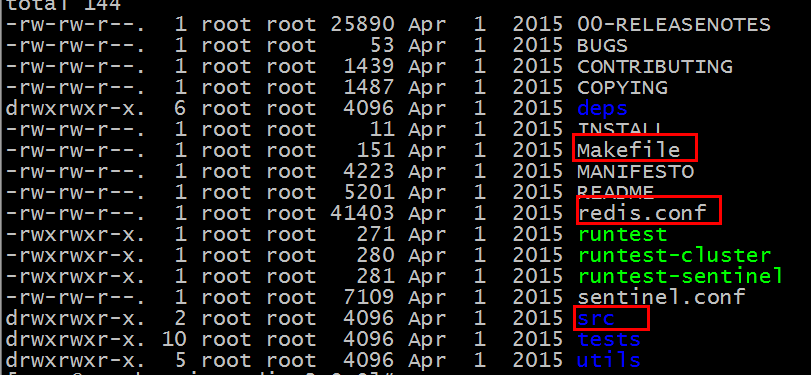
### 复制到指定目录/usr/local/src/



### 解压Redis的包



### 需要知道的目录



### 解译Redis（源码在src中）



### 安装Redis



### 启动Redis

前台启动（不般不建议使用）

后台启动（建议使用）

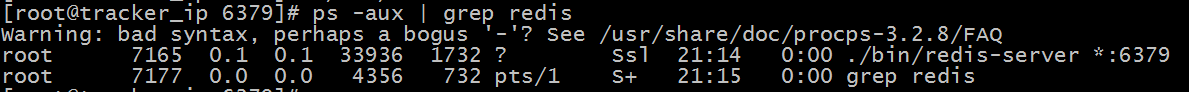
Redis.conf

daemonize:yes

后台启动Redis



### 查看Redis是否启动成功



### 客户端连接Redis服务器



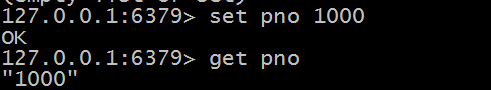
### 演示Redis的命令

#### 查看Redis中有多少个Key

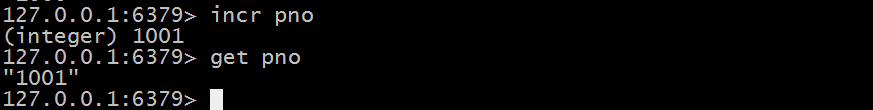


#### 设置商品编号到Redis中

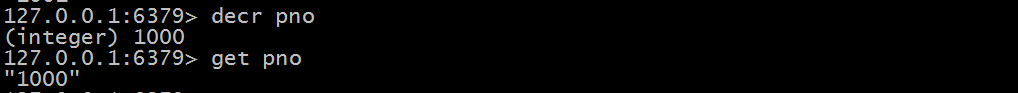
Product\_no简写pno



#### Redis让商品编号自增长1



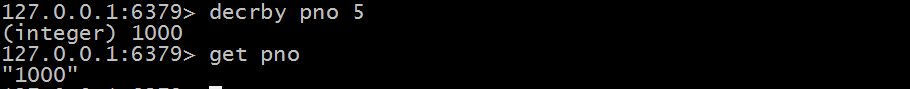
#### Redis让商品编号自减1



#### Redis让商品编号自增指定长度



#### Redis让商品编号自减指定长度



### 复制redis.conf

### redis的Java接口连接

|  |
| --- |
| @Test  public void testJedis() throws Exception {  //创建Jedis的客户端  Jedis jedis = new Jedis("192.168.200.128",6379);  //设置商品name 春夏秋冬衣服  //jedis.set("pname", "春夏秋冬衣服");  String name = jedis.get("pname");  System.out.println(name);    } |

### Spring管理Redis的Java接口

|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:task="http://www.springframework.org/schema/task"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-4.0.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/task  http://www.springframework.org/schema/task/spring-task-4.0.xsd  http://code.alibabatech.com/schema/dubbo  http://code.alibabatech.com/schema/dubbo/dubbo.xsd">    <!-- Jedis交由Spring管理 -->  <bean id="jedis" class="redis.clients.jedis.Jedis">  <constructor-arg index="0" value="192.168.200.128" type="java.lang.String"/>  <constructor-arg index="1" value="6379"/>  </bean>    </beans> |

### 商品编号由Redis统一生成

|  |
| --- |
| @Autowired  private Jedis jedis;  //商品保存  public void insertProduct(Product product){  //jedis 生成id  Long id = jedis.incr("pno");  product.setId(id);  //设置上下架 默认要求是下架  product.setIsShow(false);  //设置是否删除 不删除 1  product.setIsDel(true);  //保存商品  productDao.insertSelective(product);  //图片保存  Img img = product.getImg();  //设置商品ID  img.setProductId(product.getId());  //设置是否是默认  img.setIsDef(true);  //保存图片表  imgDao.insertSelective(img);  //保存Sku  // 商品ID 、颜色、尺码 （最小销售）  for(String color : product.getColors().split(",")){  //9 转Long  //创建Sku对象  Sku sku = new Sku();  //设置颜色  sku.setColorId(Long.parseLong(color));  //设置商品ID  sku.setProductId(product.getId());  for(String size : product.getSizes().split(",")){  //设置尺码  sku.setSize(size);  //设置运费  sku.setDeliveFee(10f);  //市场价格  sku.setMarketPrice(0f);  //售价  sku.setPrice(0f);  //库存  sku.setStock(0);  //默认200件 可更改  sku.setUpperLimit(200);  //遍历尺码 处 保存Sku  skuDao.insertSelective(sku);  }    }        } |
|  |

# 库存管理

## 设置库存按钮的请求路径

/sku/list.do?productId=${product.id}

## 创建SKuServiceImpl

|  |
| --- |
| package cn.itcast.core.service.product;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import cn.itcast.core.bean.product.Sku;  import cn.itcast.core.bean.product.SkuQuery;  import cn.itcast.core.dao.product.ColorDao;  import cn.itcast.core.dao.product.SkuDao;  /\*\*  \* 库存管理  \* @author lx  \*  \*/  @Service("skuService")  @Transactional  public class SkuServiceImpl implements SkuService {  @Autowired  private SkuDao skuDao;  @Autowired  private ColorDao colorDao;    //查询Sku结果集 通过商品ID  public List<Sku> selectSkuListByProductId(Long productId){  SkuQuery skuQuery = new SkuQuery();  //设置商品ID  skuQuery.createCriteria().andProductIdEqualTo(productId);  //返回的Sku结果集进行遍历  List<Sku> skus = skuDao.selectByExample(skuQuery);  for (Sku sku : skus) {  sku.setColor(colorDao.selectByPrimaryKey(sku.getColorId()));  }  return skus;  }  } |

## SkuController层调用SkuService层中的方法、查询Sku结果集通过商品ＩＤ

|  |
| --- |
| package cn.itcast.core.controller;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import cn.itcast.core.bean.product.Sku;  import cn.itcast.core.service.product.SkuService;  /\*\*  \* 库存管理  \* 加载库存页面数据  \* 修改库存数据  \* @author lx  \*  \*/  @Controller  public class SkuController {  @Autowired  private SkuService skuService;    //去库存列表页面  @RequestMapping(value = "/sku/list.do")  public String list(Long productId,Model model){    //加载Sku结果集 通过商品ID  List<Sku> skus = skuService.selectSkuListByProductId(productId);  //回显到页面  model.addAttribute("skus", skus);    return "sku/list";    }  } |

## 到sku/list页面上进行遍历结果集

|  |
| --- |
| <c:forEach items="${skus }" var="sku">  <tr bgcolor="#ffffff" onmouseover="this.bgColor='#eeeeee'" onmouseout="this.bgColor='#ffffff'">  <td><input type="checkbox" name="ids" value="${sku.id }"/></td>  <td>${sku.productId }</td>  <td align="center">${sku.color.name }</td>  <td align="center">${sku.size }</td>  <td align="center"><input type="text" id="m52" value="${sku.marketPrice }" disabled="disabled" size="10"/></td>  <td align="center"><input type="text" id="p52" value="${sku.price }" disabled="disabled" size="10"/></td>  <td align="center"><input type="text" id="i52" value="${sku.stock }" disabled="disabled" size="10"/></td>  <td align="center"><input type="text" id="l52" value="${sku.upperLimit }" disabled="disabled" size="10"/></td>  <td align="center"><input type="text" id="f52" value="${sku.deliveFee }" disabled="disabled" size="10"/></td>  <td align="center">不是</td>  <td align="center"><a href="javascript:updateSku(52)" class="pn-opt">修改</a> | <a href="javascript:addSku(52)" class="pn-opt">保存</a></td>  </tr>  </c:forEach> |

## 修改按钮

需要：改变当前行input框的从不可写改成可写

|  |
| --- |
| <script type="text/javascript">  //点击修改  function updateSku(skuId){  //市场价格  $("#m" + skuId).attr("disabled",false);  //售价  $("#p" + skuId).attr("disabled",false);  //库存  $("#s" + skuId).attr("disabled",false);  //运费  $("#f" + skuId).attr("disabled",false);  //购物限制  $("#l" + skuId).attr("disabled",false);    }  </script> |

## 保存

|  |
| --- |
| //点击保存  function addSku(skuId){  //市场价格  var m = $("#m" + skuId).attr("disabled",true).val();  //售价  var p = $("#p" + skuId).attr("disabled",true).val();  //库存  var s = $("#s" + skuId).attr("disabled",true).val();  //运费  var f = $("#f" + skuId).attr("disabled",true).val();  //购物限制  var l = $("#l" + skuId).attr("disabled",true).val();    //请求后台 所数据进行变更 后台打算用Sku对象接收  //设置Url  var url = "/sku/update.do";  //参数  var params = {"marketPrice" : m,"price" : p , "stock" : s, "deliveFee" : f , "upperLimit" : l,"id" : skuId};  $.post(url,params,function(data){  //回调信息 //JSON 格式 {message:'保存成功!'}  alert(data.message);  },"json");  }  </script> |

## 后台SkuController接收 Sku对象接收参数

|  |
| --- |
| //修改库存 信息 异步提交 JSON  @RequestMapping(value = "/sku/update.do")  public void update(Sku sku ,HttpServletResponse response) throws IOException{  //修改库存 信息  skuService.updateSkuById(sku);  //设置页面上的提示信息 {message:'保存成功!'}  JSONObject jo = new JSONObject();  jo.put("message", "保存成功!");  //数据回调  response.setContentType("application/json;charset=UTF-8");  response.getWriter().write(jo.toString());    } |

## SkuSericeImpl修改Sku信息

|  |
| --- |
| //修改库存信息 通过SkuID  public void updateSkuById(Sku sku){  skuDao.updateByPrimaryKeySelective(sku);  } |

# 上架 (页面上)

|  |
| --- |
| <script type="text/javascript">  function getTableForm() {  return document.getElementById('tableForm');  }  //上架  function isShow() {  if(Pn.checkedCount('ids')<=0) {  alert("请至少选择一个!");  return;  }  if(!confirm("确定上架吗?")) {  return;  }  var f = getTableForm();  f.action="/product/isShow.do";  f.submit();  }  </script> |

## 上架ProductController接收ids数组

|  |
| --- |
| //上架  @RequestMapping(value = "/product/isShow.do")  public String isShow(Long[] ids){  //上架 商品信息  //保存商品信息到Solr服务器      return "redirect:/product/list.do";  } |

## 上架ProductServiceImpl

|  |
| --- |
| //上架  public void isShow(Long[] ids){  //遍历商品ID  for (Long id : ids) {  //创建商品对象  Product product = new Product();  //设置ID  product.setId(id);  //设置商品状态 上架true 下架是 false  product.setIsShow(true);  //更改商品信息  productDao.updateByPrimaryKeySelective(product);  }  } |

## 搭建Solr服务器（单机版）

准备工作 把Solr的安装包 /usr/local/src/solr此目录

### 第一步：把Solr所需要的Jar复制到Tomcat/lib下



### 第二步：把Solr的war包复制到Tomcat/webapp下



### 第三步：设置Solr的家给Solr.war中的web.xml

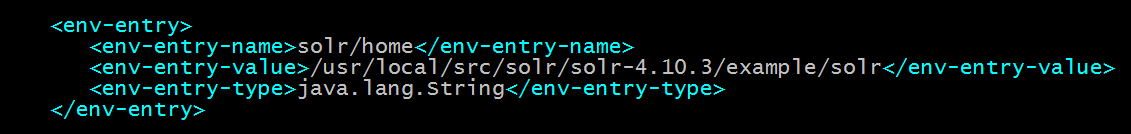
#### 解压Solr.war



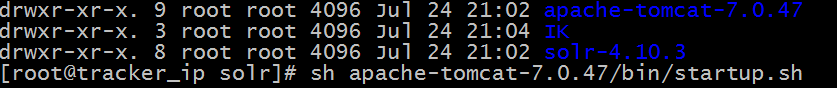
#### 注意事项



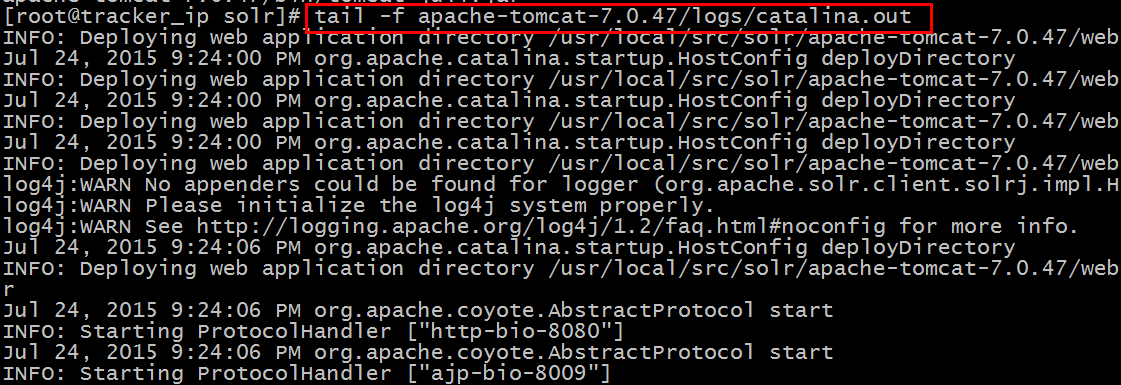
#### 设置Solr的家到web.xml



### 第四步：启动Tomcat

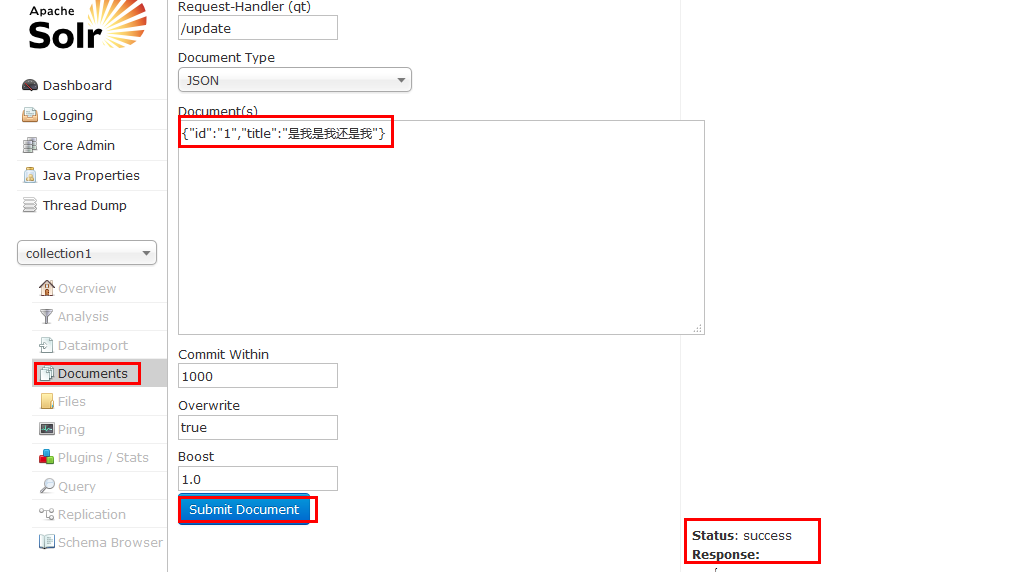


### 第五步：查看Tocmat的日志

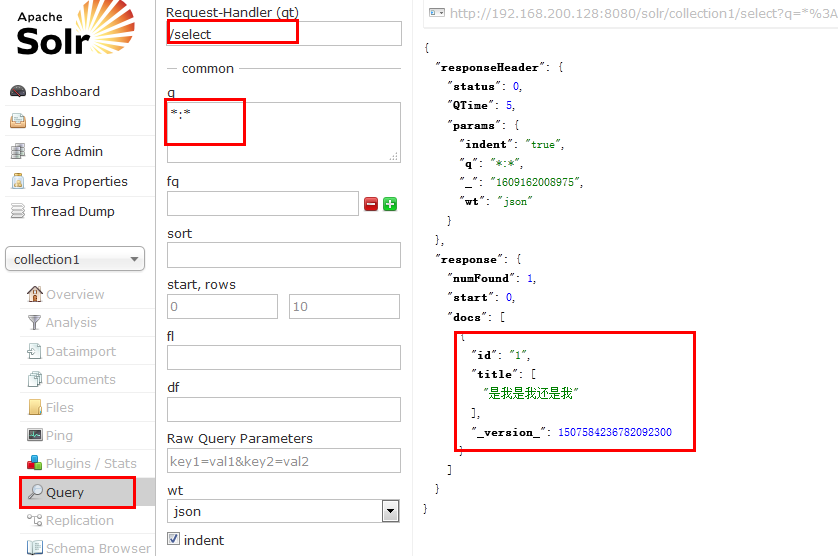


## 测试Solr服务器

### 通过浏览器保存数据到Solr服务器（测试）



### 通过浏览器查询出保存到Solr服务器上的数据（测试）



## Solr的Java接口的使用

### 保存 （简单保存二个字段、ID、Name）

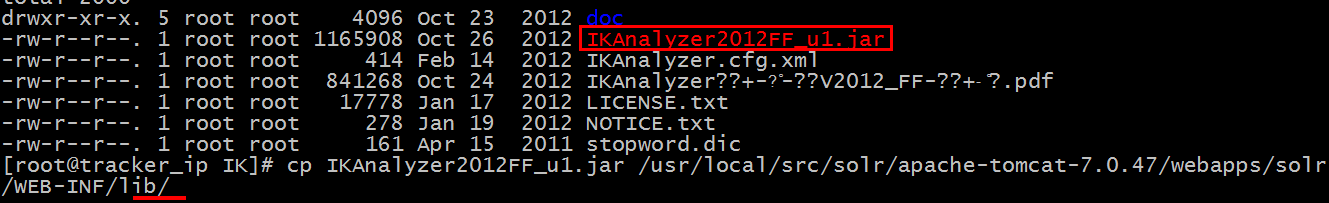
|  |
| --- |
| @Test  public void testAdd() throws Exception {  //连接服务器的请求路径  String baseURL = "http://192.168.200.128:8080/solr";  //创建Solr的客户端 连接Solr服务器  SolrServer solrServer = new HttpSolrServer(baseURL);  //创建一个存储数据的Solr对象  SolrInputDocument doc = new SolrInputDocument();  //保存ID  doc.setField("id", 2);  //保存Name  doc.setField("name", "我是中国人");  //保存数据  solrServer.add(doc);  //手动提交  solrServer.commit();  } |

### 查询（连接服务器，关键词“所有”）

|  |
| --- |
| //Solr服务器的Java接口查询  @Test  public void testQuery() throws Exception {  //连接服务器的请求路径  String baseURL = "http://192.168.200.128:8080/solr";  //创建Solr的客户端 连接Solr服务器  SolrServer solrServer = new HttpSolrServer(baseURL);      //查询  SolrQuery params = new SolrQuery();  //关键词  params.set("q", "\*:\*");  //执行查询 返回查询结果集  QueryResponse response = solrServer.query(params);  //获取结果集  SolrDocumentList docs = response.getResults();  //取出结果的总条数  long numFound = docs.getNumFound();    System.out.println("总条数：" + numFound);    //结果集遍历  for (SolrDocument doc : docs) {  //获取ID  String id = (String) doc.get("id");    //获取标题  String name = (String) doc.get("name");    System.out.println(id);  System.out.println(name);  }  } |

## 给Solr服务器配置IK分词器

### 第一步：把IK的Jar包复制给solr/WEB-INf/lib下



### 第二步：创建solr/WEB-INF/classes



### 第三步：把IK的配置文件复制到classes中



### 第四步：设置自定义的数据类型（设置成分词方式为IK分词器）

#### 配置自定义数据类型的位置



#### 配置自定义数据类型在schema.xml

|  |
| --- |
| <!-- 自定义的数据类型 支持IKAnalyzer-->  <fieldType name="text\_ik" class="solr.TextField">  <analyzer type="index" isMaxWordLength="false" class="org.wltea.analyzer.lucene.IKAnalyzer"/>  <analyzer type="query" isMaxWordLength="true" class="org.wltea.analyzer.lucene.IKAnalyzer"/>  </fieldType> |

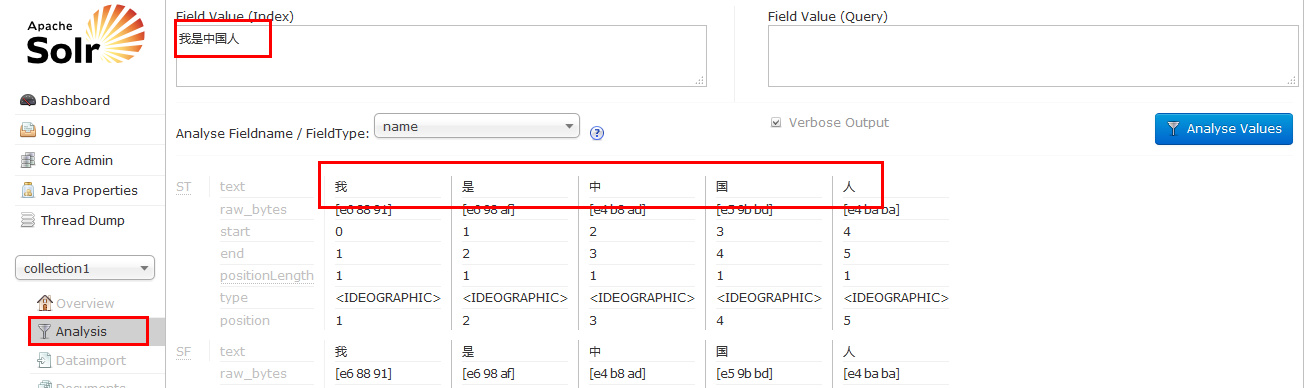
#### 配置自定义字段（数据类型使用上面IK的）

|  |
| --- |
| <!-- 自定义字段 name\_ik 网站查询的关键词字段 -->  <field name="name\_ik" type="text\_ik" indexed="true" stored="true" /> |

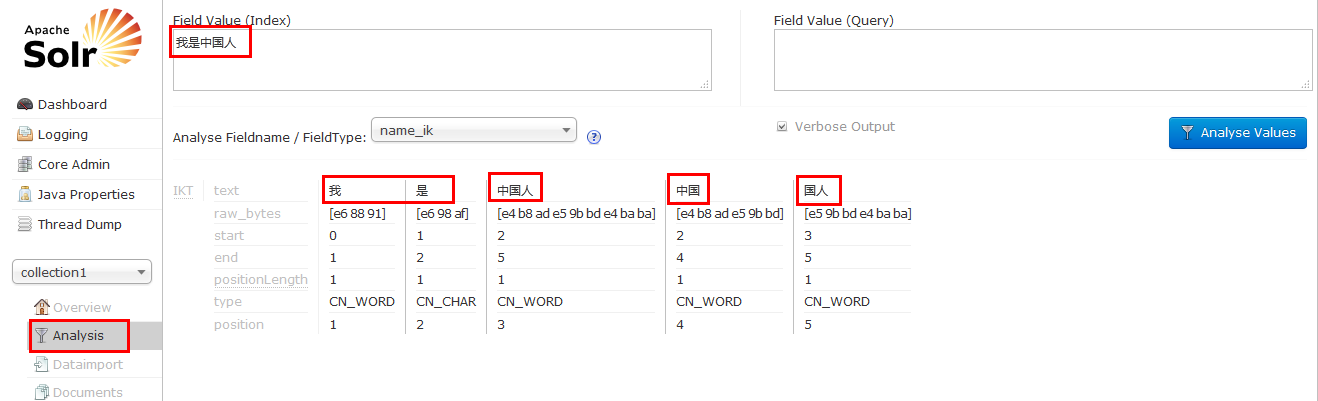
#### 重启Tomcat

|  |
| --- |
|  |

#### 查看分词前



#### 查看分词后（IK分词器）

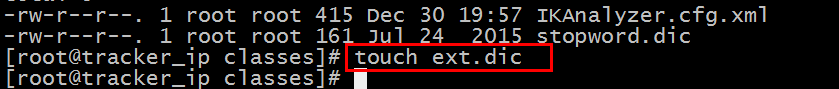


#### 扩展词器

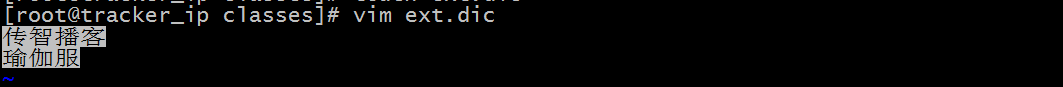
##### 配置IK配置文件



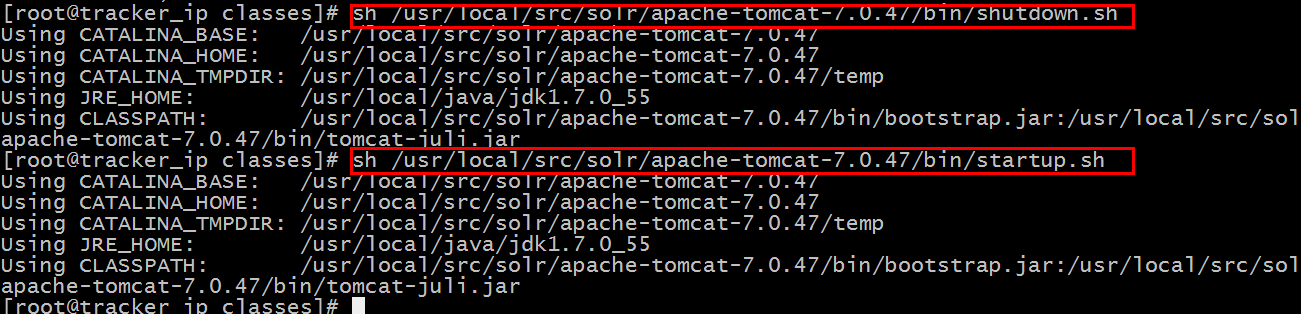
##### 创建扩展字典



##### 配置扩展字典



##### 重启Tomcat

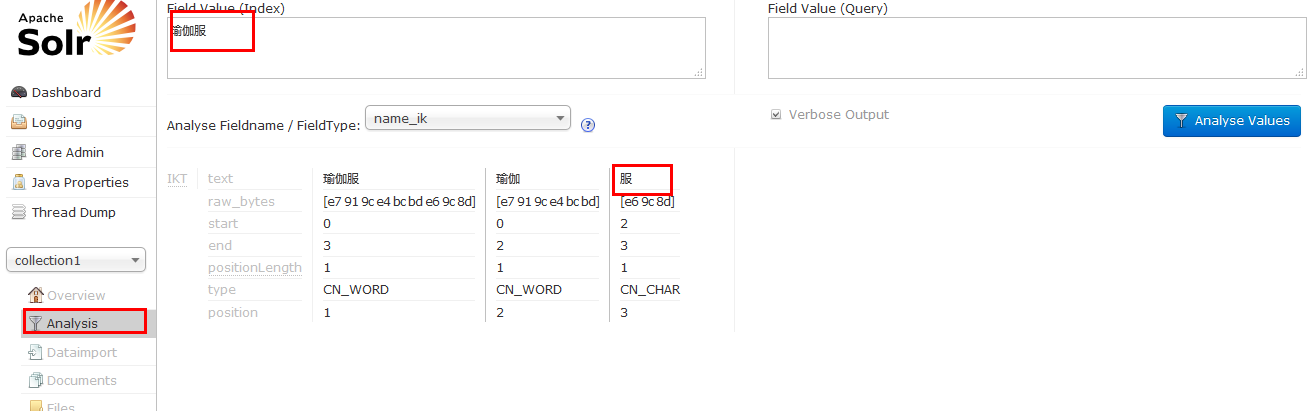


##### 效果

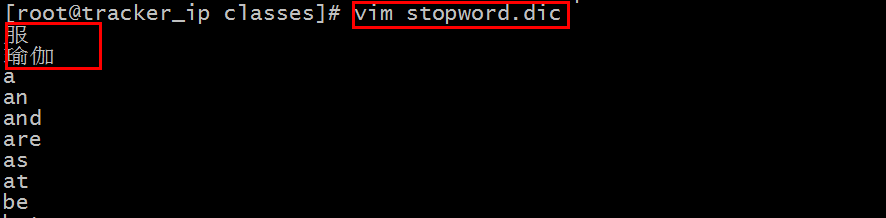


#### 停词器

##### 应用场景、不想要分词



##### 配置停词文件



##### 重启Tocmat

##### 效果



## 把商品信息保存到Solr服务器中

上架时，更改商品上下架状态、保存商品信息到Solr服务器中

商品ID

商品名称

图片的Url（默认图片）

价格

品牌ID

|  |
| --- |
| //上架  public void isShow(Long[] ids){  //遍历商品ID  for (Long id : ids) {  //创建商品对象  Product product = new Product();  //设置ID  product.setId(id);  //设置商品状态 上架true 下架是 false  product.setIsShow(true);  //更改商品信息  productDao.updateByPrimaryKeySelective(product);  //保存商品信息到Solr服务器  SolrInputDocument doc = new SolrInputDocument();  //商品ID  doc.setField("id", id);  //商品名称  Product p = productDao.selectByPrimaryKey(id);  doc.setField("name\_ik", p.getName());  //图片的Url（默认图片）  ImgQuery imgQuery = new ImgQuery();  //商品ID  //默认图片 1  imgQuery.createCriteria().andProductIdEqualTo(id).andIsDefEqualTo(true);  //只有一条数据  List<Img> imgs = imgDao.selectByExample(imgQuery);  doc.setField("url", imgs.get(0).getUrl());  //价格 select price from bbs\_sku where product\_id = 276 order by price asc limit 0, 1  SkuQuery skuQuery = new SkuQuery();  //指定价格字段进行查询  skuQuery.setFields("price");  //设置 商品ID  skuQuery.createCriteria().andProductIdEqualTo(id);  //按照价格降序排序  skuQuery.setOrderByClause("price asc");  //设置当前页为1  skuQuery.setPageNo(1);  //每页数设置为1  skuQuery.setPageSize(1);  List<Sku> skus = skuDao.selectByExample(skuQuery);  doc.setField("price", skus.get(0).getPrice());  //品牌ID  doc.setField("brandId", p.getBrandId());  //保存到Solr服务器  //提交  try {  solrServer.add(doc);  solrServer.commit();  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  } |

## 设置Solr服务器中自定义的字段brandId

<field name="brandId" type="int" indexed="true" stored="true" />

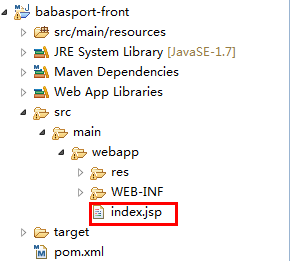
# 前台域名

http://localhost:8081

更改成

<http://www.babasport.com:8081>

要求进入首页



## 重定向的路径

<http://localhost:8081/product/display/list.shtml?keyword>=瑜伽服

## index.jsp内容如下

|  |
| --- |
| <%@page import="java.net.URLEncoder"%>  <%@ page language="java" import="java.util.\*" pageEncoding="utf-8"%>  <!-- 重定向到检索页面 -->  <%response.sendRedirect("/product/display/list.shtml?keyword=" + URLEncoder.encode("瑜伽服", "UTF-8")); %> |

## 在babasport-front创建 ProductController

|  |
| --- |
| package cn.itcast.core.controller;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  /\*\*  \* 前台管理  \* 商品的检索  \* 商品的详情页面  \* @author lx  \*  \*/  @Controller  public class ProductController {  //去检索页面  @RequestMapping(value = "/product/display/list.shtml")  public String list(String keyword){      return "product/product";  }  } |

## 用EL表达式获取请求路径后的参数值

|  |
| --- |
| <input type="text" value="${param.keyword }"/> |

## 前台开发要求不要使用A标签的href

原因：前台页面样式非常多，情况非常容易出现二次提交(a标签的href)

希望：a标签的onclick事件

## 搜索按钮

|  |
| --- |
| <div class="h-search">  <input type="text" value="${param.keyword }" id="keyword"/>  <div class="h-se-btn"><a href="javascript:void(0)" onclick="searchKeyword()">搜索</a></div>  </div>  //搜索  function searchKeyword(){  //获取关键词  var k = $("#keyword").val();  //去后台查询  window.location.href = "/product/display/list.shtml?keyword=" + k;  } |

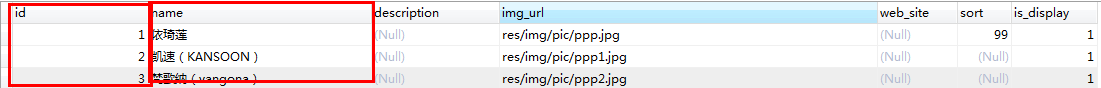
## 在ProductController中

|  |
| --- |
| /\*\*  \* 前台 商品列表  \* 商品详情页面（单品页）  \* @author lx  \*  \*/  @Controller  public class ProductController {  @Autowired  private ProductService productService;  @Autowired  private BrandService brandService;  @Autowired  private SkuService skuService;  //去商品列表页面  @RequestMapping(value = "/product/display/list.shtml")  public String list(Integer pageNo,String keyword,Long brandId,String price,Model model){  //标识  Boolean flag = true;  //条件  Map<String, String> query = new HashMap<String, String>();  //判断 品牌ID  if(null != brandId){    flag = false;    query.put("品牌", brandService.selectNameFormRedisById(brandId));    model.addAttribute("brandId", brandId);  }  if(null != price){  flag = false; //0-79 600  if(price.split("-").length == 2){  query.put("价格", price);  }else {  query.put("价格", price + "以上");  }  model.addAttribute("price", price);  }    model.addAttribute("flag", flag);  model.addAttribute("query", query);  //查询所有品牌  List<Brand> brands = brandService.selectBrandListFromRedis();    model.addAttribute("brands", brands);    Pagination pagination = productService.selectPaginationBySolr(pageNo, keyword,brandId,price);      model.addAttribute("pagination", pagination);      return "product/product";  } |

## 在ProductServiceImpl中

|  |
| --- |
| //查询Solr服务器 通过关键词  public Pagination selectPaginationBySolr(Integer pageNo,String keyword,  Long brandId,String price){  //查询  SolrQuery solrQuery = new SolrQuery();    //拼接条件  StringBuilder params = new StringBuilder();    //关键词  solrQuery.set("q", "name\_ik:" + keyword);    params.append("keyword=").append(keyword);    //创建商品条件对象  ProductQuery productQuery = new ProductQuery();  //当前页  productQuery.setPageNo(Pagination.cpn(pageNo));  productQuery.setPageSize(4);  //分页 select \* from bbs\_sfd where limit startRow pageSize (pageNo-1)  //开始行  solrQuery.setStart(productQuery.getStartRow());  //每行数  solrQuery.setRows(productQuery.getPageSize());    //过滤  //品牌ID  if(null != brandId){  solrQuery.addFilterQuery("brandId:" + brandId);  }  //价格 0-79 600  if(null != price){  String[] p = price.split("-");  if(p.length == 2){  Float startP = new Float(p[0]);  Float endP = new Float(p[1]);  solrQuery.addFilterQuery("price:[" + startP + " TO " + endP + "]");  }else if(p.length == 1){  Float startP = new Float(p[0]);  Float endP = new Float("99999999");  solrQuery.addFilterQuery("price:[" + startP + " TO " + endP + "]");  }  }  //排序  solrQuery.addSort("price", ORDER.asc);  //高亮  //1：开启高亮开关  solrQuery.setHighlight(true);  //2：设置要高亮的字段  solrQuery.addHighlightField("name\_ik");  //3:设置高亮字段的前缀 <span style='color:red'>瑜伽服</span>  solrQuery.setHighlightSimplePre("<span style='color:red'>");  //4:设置高亮字段的后缀  solrQuery.setHighlightSimplePost("</span>");        QueryResponse response = null;  try {  response = solrServer.query(solrQuery);  } catch (SolrServerException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    SolrDocumentList docs = response.getResults();    //结果集  List<Product> products = new ArrayList<Product>();  //遍历  for (SolrDocument doc : docs) {  //创建商品对象  Product product = new Product();  //商品ID  String id = (String) doc.get("id");  product.setId(Long.parseLong(id));  //取高亮  Map<String, Map<String, List<String>>> highlighting = response.getHighlighting();  Map<String, List<String>> map = highlighting.get(id);  List<String> list = map.get("name\_ik");  String name = list.get(0);  //商品名称  //String name = (String) doc.get("name\_ik");  product.setName(name);  //图片URL  String url = (String) doc.get("url");  Img img = new Img();  img.setUrl(url);  product.setImg(img);  //价格  product.setPrice((Float) doc.get("price"));    //结果集  products.add(product);    }  //总条数  long numFound = docs.getNumFound();    Pagination pagination = new Pagination(  productQuery.getPageNo(),  productQuery.getPageSize(),  (int)numFound  );  //结果集  pagination.setList(products);  //分页展示  String url = "/product/display/list.shtml";  pagination.pageView(url, params.toString());    return pagination;  } |

# 把品牌添加或修改到Redis服务器中



## Hash类型

Map map = new HashMap();

map.put(“id”,1);

map.put(“name”,依琦莲)

### 存储一个品牌

|  |
| --- |
| [root@tracker\_ip 6379]# ./bin/redis-server redis.conf  [root@tracker\_ip 6379]# ./bin/redis-cli  127.0.0.1:6379> keys \*  (empty list or set)  127.0.0.1:6379> hset brand id 1  (integer) 1  127.0.0.1:6379> hget brand id  "1"  127.0.0.1:6379> hset brand name 依琦莲  (integer) 1  127.0.0.1:6379> hget brand name  "\xe4\xbe\x9d\xe7\x90\xa6\xe8\x8e\xb2"  127.0.0.1:6379> hmset brand id 2 name 凯速  OK  127.0.0.1:6379> hget brand id  "2"  127.0.0.1:6379> hget brand name  "\xe5\x87\xaf\xe9\x80\x9f"  127.0.0.1:6379> hgetall brand  1) "id"  2) "2"  3) "name"  4) "\xe5\x87\xaf\xe9\x80\x9f"  127.0.0.1:6379> |

### 存储多个品牌

|  |
| --- |
| [root@tracker\_ip 6379]# ./bin/redis-cli  127.0.0.1:6379> keys \*  1) "brand"  127.0.0.1:6379> del brand  (integer) 1  127.0.0.1:6379> keys \*  (empty list or set)  127.0.0.1:6379> hmset brand:1 id 1 name 依琦莲  OK  127.0.0.1:6379> keys \*  1) "brand:1"  127.0.0.1:6379> hmset brand:2 id 2 name 凯速  OK  127.0.0.1:6379> keys \*  1) "brand:1"  2) "brand:2"  127.0.0.1:6379> |

### Java接口存储多个品牌

#### 在BrandServiceImpl的修改方法中

注入JedisPool

|  |
| --- |
| @Autowired  private Jedis jedis;  public void updateBrandById(Brand brand) {  // TODO Auto-generated method stub  //保存品牌数据到Redis中  //创建Map  Map<String,String> map = new HashMap<String,String>();  //id  map.put("id", String.valueOf(brand.getId()));  //name  map.put("name", brand.getName());    jedis.hmset("brand:" + brand.getId(),map);    //修改品牌在Mysql中的数据  brandDao.updateBrandById(brand);  } |

### 取一个品牌（命令）

|  |
| --- |
| 127.0.0.1:6379> keys \*  1) "brand:1"  2) "brand:6"  3) "brand:2"  4) "brand:7"  127.0.0.1:6379> hgetall brand:1  1) "id"  2) "1"  3) "name"  4) "\xe4\xbe\x9d\xe7\x90\xa6\xe8\x8e\xb2"  127.0.0.1:6379> |

### 取多个品牌（命令）

|  |
| --- |
| 127.0.0.1:6379> keys \*  1) "brand:1"  2) "brand:2"  3) "brand:6"  4) "pno"  5) "brand:7"  127.0.0.1:6379> keys brand:\*  1) "brand:1"  2) "brand:2"  3) "brand:6"  4) "brand:7"  127.0.0.1:6379> hgetall brand:6  1) "name"  2) "\xe4\xb8\xb9\xe7\x92\x90\xe6\x96\xaf"  3) "id"  4) "6"  127.0.0.1:6379> |

### Java接口取多个品牌（在BrandServiceImpl中）

|  |
| --- |
| //查询品牌从Redis中  public List<Brand> selectBrandListFromRedis(){  // brand:1,2,3,4...  Set<String> keys = jedis.keys("brand:\*");    List<Brand> brands = new ArrayList<Brand>();    for (String key : keys) {  Brand brand = new Brand();  //ID  brand.setId(Long.parseLong(jedis.hget(key, "id")));  //名称  brand.setName(jedis.hget(key, "name"));    brands.add(brand);  }    return brands;  } |

# 前台商品检索过滤条件查询

## 品牌过滤

|  |
| --- |
| //全局变量  var keyword = '${param.keyword}';  //回显品牌ID  var brandId = '${brandId}';  //价格回显  var price = '${price}';  //点击品牌 id==品牌ID  function fqBrand(id){  //判断价格是否已经选择  if(price != ''){  window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&brandId=" + id + "&price=" + price;  }else{  window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&brandId=" + id;  }  } |

|  |
| --- |
| //判断 品牌ID  if(null != brandId){  //添加过滤条件  params.addFilterQuery("brandId:" + brandId);  //回显品牌ID  model.addAttribute("brandId", brandId);  } |

## 价格过滤

|  |
| --- |
| //点击价格 p就是价格  function fqPrice(p){  //判断是否已经选择品牌ID  if(brandId != ''){  //拼接上品牌ID  window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&price=" + p+ "&brandId=" + brandId;  }else{  window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&price=" + p ;    }  } |

|  |
| --- |
| //判断价格  if(null != price){  //添加价格过滤条件  String[] p = price.split("-");  if(p.length == 2){  //第一种： 0-79  //开始价  Float pStart = new Float(p[0]);  //结束价格  Float pEnd = new Float(p[1]);  //价格过滤  params.addFilterQuery("price:[" + pStart +" TO " + pEnd + "]");  }else if(p.length == 1){  //第二种：600~无限大  //开始价  Float pStart = new Float(p[0]);  //结束价格  Float pEnd = new Float("99999999");  //价格过滤  params.addFilterQuery("price:[" + pStart +" TO " + pEnd + "]");  }  //回显价格  model.addAttribute("price", price);  } |

## 已选条件

|  |
| --- |
| //已选条件标识  Boolean flag = false;  //已选条件存储  Map<String,String> query = new HashMap<String,String>();    //判断 品牌ID  if(null != brandId){  //添加过滤条件  params.addFilterQuery("brandId:" + brandId);  //回显品牌ID  model.addAttribute("brandId", brandId);  //已选条件标识  flag = true;  //添加过滤条件到query  query.put("品牌", jedis.hget("brand:" + brandId, "name"));  } |

|  |
| --- |
| <c:if test="${flag }">  <li><label>已选条件：</label>  <p class="sel">  <c:forEach items="${query }" var="q">  <a href="javascript:void(0);">  <em>${q.key }：</em>${q.value }  <cite title="关闭此筛选条件">X</cite></a>  </c:forEach>  </p>  </li>  </c:if> |

## 取消过滤条件

|  |
| --- |
| //点击取消过滤条件  function cleanFq(key){  //品牌  if("品牌" == key){  //判断价格是否已经选择  if(price != ''){  window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&price=" + price;  }else{  window.location.href = "/product/display/list.shtml?keyword=" + keyword;  }  //价格  }else if("价格" == key){  //判断是否已经选择品牌ID  if(brandId != ''){  //拼接上品牌ID  window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&brandId=" + brandId;  }else{  window.location.href = "/product/display/list.shtml?keyword=" + keyword;    }  }  } |

# 前台商品详情页面

## 去商品详情页面

### 设置请求路径

/product/detail.shtml?id=${product.id}

* 330行

|  |
| --- |
| <a href="javascript:void(0)" onclick="window.open('/product/detail.shtml?id=${product.id}')" |

### 通过商品ID加载数据

* 商品表 商品名称、商品介绍、包装清单
* 图片表 商品ID加默认查询一张图片 （图片Url）
* 库存表（SKu表）、最小销售单元表 （通过商品ID查询Sku结果集）（必须有库存）
* 颜色表 通过Sku当中的颜色ID查询颜色表
* 去掉重复的颜色

### 编写ProductController

|  |
| --- |
| //去商品详情页面  @RequestMapping(value = "/product/detail.shtml")  public String detail(Long id,Model model){    //加载商品及图片  Product product = productService.selectProductAndImgById(id);  model.addAttribute("product", product);  //Sku结果集 及 颜色  List<Sku> skus = skuService.selectSkuListWithStockByProductId(id);    //Set集合 去掉重复的  Set<Color> colors = new HashSet<Color>();  //遍历Sku  for (Sku sku : skus) {  //Sku中的Color对象放到Set<Color> 中  colors.add(sku.getColor());  }    model.addAttribute("skus", skus);  model.addAttribute("colors", colors);      return "product/productDetail";  }  @Autowired  private ProductService productService;  @Autowired  private SkuService skuService; |

### Sku联动

#### 点击颜色：让颜色变成红边框

|  |
| --- |
| //点击颜色 id == 颜色ID  function colorToRed(target,id){  //颜色变成红边框  $(target).attr("class","changToRed");  } |

#### 清理之前点击过的红色

|  |
| --- |
| //点击颜色 id == 颜色ID  function colorToRed(target,id){  //清理所有A标签  $("#colors a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","changToWhite");  });    //颜色变成红边框  $(target).attr("class","changToRed");  } |

#### 通过遍历Skus结果集 判断与当前选择的颜色ID相等,来获取尺码

|  |
| --- |
| //点击颜色 id == 颜色ID  function colorToRed(target,id){  //清理所有A标签  $("#colors a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","changToWhite");  });  //颜色变成红边框  $(target).attr("class","changToRed");    //skus  <c:forEach items="${skus }" var="sku">  if(id == '${sku.colorId}'){  //alert('${sku.size}');//M L  $("#" + '${sku.size}').attr("class","changToWhite");  }  </c:forEach>  } |

#### 清理之前尺码不可点击

|  |
| --- |
| //点击颜色 id == 颜色ID  function colorToRed(target,id){  //清理尺码  $("#sizes a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","not-allow");  });  //清理所有颜色  $("#colors a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","changToWhite");  });  //颜色变成红边框  $(target).attr("class","changToRed");    //skus  <c:forEach items="${skus }" var="sku">  if(id == '${sku.colorId}'){  //alert('${sku.size}');//M L  $("#" + '${sku.size}').attr("class","changToWhite");  }  </c:forEach>  } |

#### 要求第一个尺码变成红边框、其它有库存尺码变成白边框

|  |
| --- |
| //点击颜色 id == 颜色ID  function colorToRed(target,id){  //清理尺码  $("#sizes a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","not-allow");  });  //清理所有颜色  $("#colors a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","changToWhite");  });  //颜色变成红边框  $(target).attr("class","changToRed");    //标记  var flag = 0;  //skus  <c:forEach items="${skus }" var="sku">  if(id == '${sku.colorId}'){  if(flag == 0){  //第一次进来  $("#" + '${sku.size}').attr("class","changToRed");  //设置标记  flag = 1;  }else{  //alert('${sku.size}');//M L  $("#" + '${sku.size}').attr("class","changToWhite");  }  }  </c:forEach>  } |

#### 变更巴巴价、市场价、运费、库存

|  |
| --- |
| //点击颜色 id == 颜色ID  function colorToRed(target,id){  //清理尺码  $("#sizes a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","not-allow");  });  //清理所有颜色  $("#colors a").each(function(){  //对每一个A标签的一个操作  $(this).attr("class","changToWhite");  });  //颜色变成红边框  $(target).attr("class","changToRed");    //标记  var flag = 0;  //skus  <c:forEach items="${skus }" var="sku">  if(id == '${sku.colorId}'){  if(flag == 0){  //第一次进来  $("#" + '${sku.size}').attr("class","changToRed");  //设置标记  flag = 1;  //设置巴巴价 ￥  $("#bbprice").html("￥" + '${sku.price}');  //市场价  //库存  //运费    }else{  //alert('${sku.size}');//M L  $("#" + '${sku.size}').attr("class","changToWhite");  }  }  </c:forEach>  } |

#### 点击尺码

|  |
| --- |
| //点击尺码 id == 尺码 （M 、S 、L。。。  function sizeToRed(target,id){  //判断是不是不可点击的  var cc = $(target).attr("class");  if(cc == "not-allow"){  return;  }  //清理尺码  $("#sizes a").each(function(){  //对每一个A标签的一个操作  var c = $(this).attr("class");  //判断样式是否是红边框  if(c == "changToRed"){  //设置成红边框  $(this).attr("class","changToWhite");  }  });  //尺码变成红边框  $(target).attr("class","changToRed");  //控制巴巴价  //skus  <c:forEach items="${skus }" var="sku">  //颜色与尺码全相同  if(id == '${sku.size}' && colorId == '${sku.colorId}'){  //设置巴巴价 ￥  $("#bbprice").html("￥" + '${sku.price}');  //市场价  //库存  //运费  }  </c:forEach>      } |

#### 页面初始化加载时、触发第一个颜色的onclick事件

|  |
| --- |
| //jquery初始化程序  $(function(){    //页面加载时、触发第一个颜色的onclick事件  $("#colors a:first").trigger("click");    }); |

## 为购买按钮预留参数

skuId

购买的数量

|  |
| --- |
| //全局变量  var skuId;  //立即购买  function buy(){  window.location.href='/shopping/buyerCart.shtml?skuId=' + skuId + "&amount=" + $("#num").val();  } |

## + -(购买数量)

### +

|  |
| --- |
| //jquery初始化程序  $(function(){    //页面加载时、触发第一个颜色的onclick事件  $("#colors a:first").trigger("click");    //+ jquery给+绑定oncick事件  $("#add").click(function(){  //点击+时、执行的程序  //获取当前多少件  var num = $("#num").val();  //追加1件  num++;  //如果数量大于购买限制时  if(num > upperLimit){  alert("不能再买了");  return;  }  //把追加后的数量设置给num  $("#num").val(num);  });    });  //购买限制  var upperLimit; |

### –

|  |
| --- |
| //-  $("#sub").click(function(){  //点击+时、执行的程序  //获取当前多少件  var num = $("#num").val();  //追加1件  num--;  //如果数量大于购买限制时  if(num == 0){  alert("你还买不了买了");  return;  }  //把追减后的数量设置给num  $("#num").val(num);  }); |

## 将对商品详情页面（单品页）进行优化（访问速度上的优化）

### 将此页面做成静态化

将动态的页面做成静态化（html）

好处：用户访问静态页面时，不走后台、不走数据库，直接访问页面

### 静态化技术Freemarker的基本语法

FreeMarker是一个用Java语言编写的模板引擎，它基于模板来生成文本输出。FreeMarker与Web容器无关，即在Web运行时，它并不知道Servlet或[HTTP](http://baike.baidu.com/view/9472.htm)。它不仅可以用作表现层的实现技术，而且还可以用于生成XML，JSP或Java 等。

目前企业中:主要用Freemarker做静态页面或是页面展示

#### 使用Freemarker所用到的jar

把下载到的jar包（freemarker-2.3.16.jar）放到\WebRoot\WEB-INF\lib目录下。下载地址：http://freemarker.sourceforge.net/freemarkerdownload.html （注：官方网站.org的经常打不开）

如果使用的是Maven结构,可在pom.xml中引入以下坐标

<dependency>

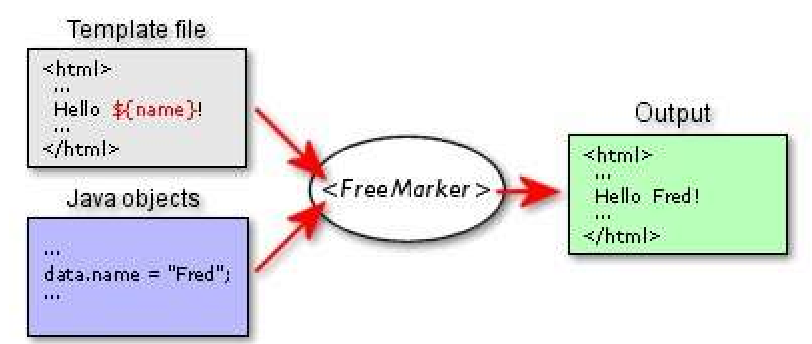
<groupId>freemarker</groupId>

<artifactId>freemarker</artifactId>

<version>2.3.18</version>

</dependency>

#### Freemarker原理图



模板 + 数据模型 = 输出

#### 搭建Freemarker的运行环境

|  |
| --- |
| package cn.itcast;  import java.io.File;  import java.io.FileWriter;  import java.io.Writer;  import java.util.HashMap;  import java.util.Map;  import freemarker.template.Configuration;  import freemarker.template.Template;  /\*\*  \* 搭建Freemarker环境、并测试  \* @author lx  \*  \*/  public class FreemarkerDemo {  //Freemarker从这里开始  public static void main(String[] args) throws Exception {  //创建Freemarker实例  Configuration conf = new Configuration();  //设置模板 freemarker.html  String dir = "C:\\workspace\\freemarker\\ftl\\";  //设置模板的目录  conf.setDirectoryForTemplateLoading(new File(dir));  //指定模板 返回模板对象  Template template = conf.getTemplate("freemarker.html");  //设置数据  Map root = new HashMap();  //世界你好  root.put("world", "世界你好");  //输出  Writer out = new FileWriter(new File(dir + "hello.html"));  //Freemarker处理数据及模板 、并输出  template.process(root, out);    out.close();    System.out.println("生成完毕");      }  } |

#### 对象输出

|  |
| --- |
| //对象  Person person = new Person();  person.setId(8);  person.setName("哈哈");  root.put("person", person);  ${person.id}/${person.name} |

#### List集合

|  |
| --- |
| //List集合    List<String> persons = new ArrayList<String>();  persons.add("范冰冰");  persons.add("李冰冰");  persons.add("何灵");  root.put("persons", persons);  <#list persons as person>  ${person}  </#list> |

#### Map遍历

|  |
| --- |
| Map mx = new HashMap();  mx.put("fbb","范冰冰");  mx.put("lbb","李冰冰");  root.put("mx",mx);  <#list mx?keys as key>  ${mx[key]}  </#list>  <!-- 第二种： -->  ${mx.fbb}/${mx.lbb} |

#### List<Map> 遍历

|  |
| --- |
| List<Map> maps = new ArrayList<Map>();  Map pms1 = new HashMap();  pms1.put("id1", "范冰冰");  pms1.put("id2", "李冰冰");  Map pms2 = new HashMap();  pms2.put("id1", "曾志伟");  pms2.put("id2", "何炅");  maps.add(pms1);  maps.add(pms2);  root.put("maps", maps);  <!-- 第一种 -->  <#list maps as map>  <#list map?keys as key>  ${map[key]}  </#list>  </#list>  <!-- 第二种 -->  <#list maps as map>  ${map.id1}/${map.id2}  </#list> |

#### 遍历集合的索引

|  |
| --- |
| List<String> persons = new ArrayList<String>();  persons.add("范冰冰");  persons.add("李冰冰");  persons.add("何灵");  root.put("persons", persons);  <#list persons as person>  ${person\_index}  </#list> |

### 将静态化技术应用于项目（新巴巴运动网的商品详情页面）

#### Freemarker作为一种服务来使用

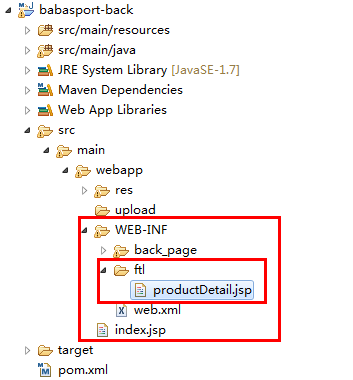
babasport-service cn.itcast.core.service.staticpage.StaticPageServiceImpl 交由Spring来管理

#### 获取Configuration

|  |
| --- |
| package cn.itcast.core.service.staticpage;  import org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;  import freemarker.template.Configuration;  /\*\*  \* 静态化服务类  \* @author lx  \*  \*/  public class StaticPageServiceImpl implements StaticPageService{  private Configuration conf;  public void setFreeMarkerConfigurer(FreeMarkerConfigurer freeMarkerConfigurer) {  this.conf = freeMarkerConfigurer.getConfiguration();  }  //静态化方法  public void index(){  //从 Configuration    }  } |

|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-3.0.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-3.0.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx-3.0.xsd ">  <!-- Freemarker配置 -->  <bean id="staticPageService" class="cn.itcast.core.service.staticpage.StaticPageServiceImpl">  <!-- 注入FreeMarkerConfigurer -->  <property name="freeMarkerConfigurer">  <bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"></bean>  </property>  </bean>      </beans> |

#### 模板位置



#### 设置模板路径

|  |
| --- |
| <!-- Freemarker配置 -->  <bean id="staticPageService" class="cn.itcast.core.service.staticpage.StaticPageServiceImpl">  <!-- 注入FreeMarkerConfigurer -->  <property name="freeMarkerConfigurer">  <bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">  <!-- 采用相对路径来设置模板目录 -->  <property name="templateLoaderPath" value="/WEB-INF/ftl/"/>  </bean>  </property>  </bean> |

#### 指定模板 返回模板对象

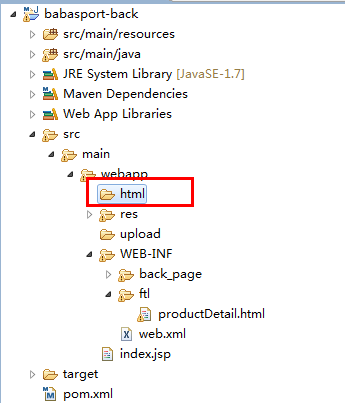
|  |
| --- |
| //静态化方法  public void index() throws Exception{  //指定模板 返回模板对象  Template template = conf.getTemplate("productDetail.html");  } |

#### 模板中jstl标签更改成Freemarker标签

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <#list skus as sku>  //颜色与尺码全相同  if(id == '${sku.size}' && colorId == '${sku.colorId}'){  //设置巴巴价 ￥  $("#bbprice").html("￥" + '${sku.price}');  //市场价  //库存  //运费  //设置SkuID  skuId = '${sku.id}';  }  </#list>  <#list colors as color>  <a onclick="colorToRed(this,'${color.id}')" href="javascript:void(0)" title="西瓜红" class="changToWhite"><img width="25" height="25" data-img="1" src="/res/img/pic/ppp00.jpg" alt="西瓜红 "><i>${color.name }</i></a>  </#list> |

#### 让Freemarker处理模板及数据、并输出

#### 输出位置



#### 静态化方法

|  |
| --- |
| package cn.itcast.core.service.staticpage;  import java.io.File;  import java.io.FileOutputStream;  import java.io.FileWriter;  import java.io.IOException;  import java.io.OutputStreamWriter;  import java.io.Writer;  import java.util.Map;  import javax.servlet.ServletContext;  import org.springframework.web.context.ServletContextAware;  import org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;  import freemarker.template.Configuration;  import freemarker.template.Template;  import freemarker.template.TemplateException;  /\*\*  \* 静态化服务类  \* @author lx  \*  \*/  public class StaticPageServiceImpl implements StaticPageService,ServletContextAware{  private Configuration conf;  public void setFreeMarkerConfigurer(FreeMarkerConfigurer freeMarkerConfigurer) {  this.conf = freeMarkerConfigurer.getConfiguration();  }  //静态化方法  public void index(Map<String,Object> root,Long id) {  //获取到全路径  String path = getPath("/html/product/" + id + ".html");  //创建文件  File f = new File(path);  //父文件夹  File parentFile = f.getParentFile();  //判断父文件夹存在不存在  if(!parentFile.exists()){  parentFile.mkdirs();  }  //输出流  Writer out = null;  //处理template  try {  //流写的过程 指定编码UTF-8  out = new OutputStreamWriter(new FileOutputStream(f), "UTF-8");  //指定模板 返回模板对象 读UTF-8  Template template = conf.getTemplate("productDetail.html");    template.process(root, out);  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }finally{  //关流  if(out != null){  try {  out.close();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }    }    }    //获取真实全路径  public String getPath(String name){  return servletContext.getRealPath(name);  }  //声明一个ServletContext  private ServletContext servletContext;  public void setServletContext(ServletContext servletContext) {  // TODO Auto-generated method stub  this.servletContext = servletContext;    }  } |

#### 上架（后台方法中调用静态化方法）

在ProductServiceImpl的上架方法中

|  |
| --- |
| //静态化（商品详情页面）  Map<String,Object> root = new HashMap<String,Object>();    //加载商品及图片  Product pp = selectProductAndImgById(id);  root.put("product", pp);  //Sku结果集 及 颜色  List<Sku> ss = skuService.selectSkuListWithStockByProductId(id);    //Set集合 去掉重复的  Set<Color> colors = new HashSet<Color>();  //遍历Sku  for (Sku sku : ss) {  //Sku中的Color对象放到Set<Color> 中  colors.add(sku.getColor());  }    root.put("skus", ss);  root.put("colors", colors);    staticPageService.index(root, id);  @Autowired  private StaticPageService staticPageService;  @Autowired  private SkuService skuService; |

#### 设置前台项目的路径

在商品检索页面、商品名称

|  |
| --- |
| <a href="javascript:void(0)" onclick="window.open('http://localhost:8080/html/product/${product.id}.html')" |

# 登陆（在babasport-service项目中）

## Session提供类

以下为SessionProvider接口

|  |
| --- |
| public interface SessionProvider {  //把用户名入到 Redis中 30分钟  public void setAttributeForUserName(String name,String value);    //把验证码放到Redis中 10分钟  public void setAttributeForCode(String name,String value);    //取用户名  public String getAttibuteForUserName(String name);    //取验证码  public String getAttibuteForCode(String name);    } |

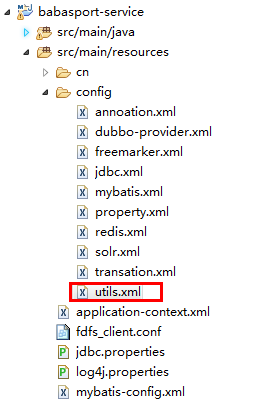
### 远程Session（保存用户或验证码到Redis中）

|  |
| --- |
| /\*\*  \* Session提供类  \* @author lx  \*  \*/  public class CacheSessionProvider implements SessionProvider {  @Autowired  private Jedis jedis;    //用户超时分钟  private Integer sexp = 30;  //验证码超时时间  private Integer cexp = 10;  public void setCexp(Integer cexp) {  this.cexp = cexp;  }  public void setSexp(Integer sexp) {  this.sexp = sexp;  }  //把用户名入到 Redis中 30分钟  public void setAttributeForUserName(String name, String value) {  // TODO Auto-generated method stub  // K CSESSIONID : V fbb2014  jedis.set(name + ":" + Constants.USER\_NAME, value);  //30分钟  jedis.expire(name + ":" + Constants.USER\_NAME, sexp\*60);  }  //把验证码放到Redis中 10分钟  public void setAttributeForCode(String name, String value) {  // TODO Auto-generated method stub  jedis.set(name + ":" +Constants.CODE\_NAME, value);  //10分钟  jedis.expire(name + ":" + Constants.CODE\_NAME, cexp\*60);  }    //取用户名  public String getAttibuteForUserName(String name) {  // TODO Auto-generated method stub CSESSIONID  String value = jedis.get(name + ":" + Constants.USER\_NAME);  //重新设置时间  if(null != value){  //30分钟  jedis.expire(name + ":" + Constants.USER\_NAME, sexp\*60);  }  return value;  }  //取验证码  public String getAttibuteForCode(String name) {  // TODO Auto-generated method stub  String value = jedis.get(name + ":" + Constants.CODE\_NAME);  //重新设置时间  if(null != value){  //10分钟  jedis.expire(name + ":" + Constants.CODE\_NAME, cexp\*60);  }  return value;  }  } |

### 系统常量Constants

|  |
| --- |
| package cn.itcast.common.web;  /\*\*  \* 系统常量  \* @author lx  \*  \*/  public interface Constants {  //自定义JESSIONID名称  public static final String CJSESSIONID = "CJSESSIONID";  } |

### 把Session提供类交给Spring管理



|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:task="http://www.springframework.org/schema/task"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-4.0.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/task  http://www.springframework.org/schema/task/spring-task-4.0.xsd  http://code.alibabatech.com/schema/dubbo  http://code.alibabatech.com/schema/dubbo/dubbo.xsd">    <!-- Session 提供类 -->  <bean id="sessionProvider" class="cn.itcast.core.service.user.CacheSessionProvider">  <!-- 用户超时时间 -->  <property name="sexp" value="60"/>  <!-- 验证码超时时间 -->  <!-- <property name="cexp" value="1"/> -->  </bean>      </beans> |

## 商品检索页面登陆按钮跳转到登陆页面（在babasport-login项目中）

是否拦截的请求路径（约定如下）

* 去登陆页面不拦截

/shopping/

* 结算按钮 提交订单 必须登陆

/buyer

### 在商品检索页面上写去登陆页面方法

|  |
| --- |
| //登陆  function login(){  window.location.href = "/shopping/login.shtml";  } |

### 创建LoginController

|  |
| --- |
| /\*\*  \* 去登陆页面  \* 提交登陆表单  \* 验证码生成  \* 加密密码  \* @author lx  \*  \*/  @Controller  public class LoginController {      //去登陆页面  @RequestMapping(value = "/shopping/login.shtml",method=RequestMethod.GET)  public String login(){  return "buyer/login";  }} |

## 验证码生成

### 在业务常量中定义验证码名称

|  |
| --- |
| package cn.itcast.core.web;  /\*\*  \* 业务常量  \* @author lx  \*  \*/  public interface Constants {  //图片服务器  public static final String IMG\_URL = "http://192.168.200.128/";    //验证码常量  public static final String CODE\_SESSION = "CODE\_SESSION";  //用户名  public static final String USER\_SESSION = "USER\_SESSION";  } |

### 在Login页面上54行

|  |
| --- |
| <img src="/shopping/getCodeImage.shtml" onclick="this.src='/shopping/getCodeImage.shtml?d='+new Date()" class="code" alt="换一张" /><a href="javascript:void(0);" onclick="$('.code').attr('src','/shopping/getCodeImage.shtml?d='+new Date())" title="换一张">换一张</a></li> |

### 在LoginController类中生成的验证码

|  |
| --- |
| @Autowired  private SessionProvider sessionProvider;  //验证码生成  @RequestMapping(value = "/shopping/getCodeImage.shtml")  public void getCodeImage(HttpServletRequest request,HttpServletResponse response){  System.out.println("#######################生成数字和字母的验证码#######################");  BufferedImage img = new BufferedImage(68, 22,  BufferedImage.TYPE\_INT\_RGB);  // 得到该图片的绘图对象  Graphics g = img.getGraphics();    Random r = new Random();    Color c = new Color(200, 150, 255);    g.setColor(c);    // 填充整个图片的颜色    g.fillRect(0, 0, 68, 22);    // 向图片中输出数字和字母    StringBuffer sb = new StringBuffer();    char[] ch = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".toCharArray();    int index, len = ch.length;    for (int i = 0; i < 4; i++) {    index = r.nextInt(len);    g.setColor(new Color(r.nextInt(88), r.nextInt(188), r.nextInt    (255)));    g.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 22));  // 输出的 字体和大小    g.drawString("" + ch[index], (i \* 15) + 3, 18);  //写什么数字，在图片 的什么位置画    sb.append(ch[index]);    }  //把上面生成的验证码放到Session域中  sessionProvider.setAttributeForCode(request,response, Constants.CODE\_SESSION, sb.toString());  try {  ImageIO.write(img, "JPG", response.getOutputStream());  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    } |

## 登陆表单提交

### 设置Login页面上的字段及Form的Action的请求路径

|  |
| --- |
| <form id="jvForm" action="/shopping/login.shtml" method="post">  <input type="hidden" name="returnUrl" value="${param.directUrl}"/>  <ul class="uls form">  <li id="errorName" class="errorTip" style="display:none">${error}</li>  <li><label for="username">用户名：</label>  <span class="bg\_text">  <input type="text" id="username" name="username" maxLength="100" />  </span>  </li>  <li><label for="password">密　码：</label>  <span class="bg\_text">  <input type="password" id="password" name="password" maxLength="32" />  </span>  </li>  <li><label for="captcha">验证码：</label>  <span class="bg\_text small">  <input type="text" id="captcha" name="code" maxLength="7"/>  </span>  <img src="/shopping/getCodeImage.shtml" onclick="this.src='/shopping/getCodeImage.shtml?d='+new Date()" class="code" alt="换一张" /><a href="javascript:void(0);" onclick="$('.code').attr('src','/shopping/getCodeImage.shtml?d='+new Date())" title="换一张">换一张</a></li>  <li><label for="">&nbsp;</label><input type="submit" value="登 录" class="hand btn66x23"/><a href="#" title="忘记密码？">忘记密码？</a></li>  </ul>  </form> |

### LoginController接收登陆表单

|  |
| --- |
| /\*\*  \* 去登陆页面  \* 提交登陆表单  \* 验证码生成  \* 加密密码  \* @author lx  \*  \*/  @Controller  public class LoginController {  @Autowired  private SessionProvider sessionProvider;    @Autowired  private BuyerService buyerService;    //去登陆页面  @RequestMapping(value = "/shopping/login.shtml",method=RequestMethod.GET)  public String login(){  return "buyer/login";  }  //登陆表单提交  @RequestMapping(value = "/shopping/login.shtml",method=RequestMethod.POST)  public String login(String username,String password,String code,String returnUrl  ,HttpServletRequest request,HttpServletResponse response,Model model){  //考虑到用户的体验  //1:验证码不能为空  if(null != code){  //2:验证码码必须正确  if(code.equalsIgnoreCase(sessionProvider.  getAttribute(request, response, Constants.CODE\_SESSION))){  //3:用户名不能为空  if(null != username){  //4:密码不能为空  if(null != password){  //5:用户名必须正确  Buyer buyer = buyerService.selectBuyerByUserName(username);  if(null != buyer){  //6:密码必须正确  if(buyer.getPassword().equals(encodePassword(password))){  //7:把用户名放到Session中  sessionProvider.setAttributeForUserName(request, response,  Constants.USER\_SESSION, buyer.getUsername());    //8:登陆成功、返回用户之前访问的页面（难点）  return "redirect:" + returnUrl;  }else{  //提示信息  model.addAttribute("error", "密码必须正确");  return "buyer/login";  }    }else{  //提示信息  model.addAttribute("error", "用户名必须正确");  return "buyer/login";  }    }else{  //提示信息  model.addAttribute("error", "密码不能为空");  return "buyer/login";  }    }else{  //提示信息  model.addAttribute("error", "用户名不能为空");  return "buyer/login";  }    }else{  //提示信息  model.addAttribute("error", "验证码必须正确");  return "buyer/login";  }    }else{  //提示信息  model.addAttribute("error", "验证码不能为空");  return "buyer/login";  }    }    //加密密码（ MD5 + 十六进制 ） 加盐  public String encodePassword(String password){  //JDK MD5  String algorithm = "MD5";    //加盐  //password = "qazwswewrqweqeqequuhjhgjhhgj";  //  char[] encodeHex = null;  try {  MessageDigest instance = MessageDigest.getInstance(algorithm);  //开始进行MD5加密  byte[] digest = instance.digest(password.getBytes());  //开始进行十六进制加密  encodeHex = Hex.encodeHex(digest);  } catch (NoSuchAlgorithmException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return new String(encodeHex);    }  } |

### 对返回用户之前访问的页面请求路径转义

#### 在检索页面上去登陆页面的方法中

|  |
| --- |
| //登陆  function login(){  window.location.href = "/shopping/login.shtml?returnUrl=" + encodeURIComponent(window.location.href);  } |

# Springmvc拦截器

## Springmvc的配置文件进行配置

|  |
| --- |
| <!-- Springmvc的拦截器 目前：判断用户是否登陆 -->  <mvc:interceptors>  <!-- 配置多个拦截器 -->  <mvc:interceptor>  <!-- 拦截器规则 拦截所有（web.xml 配置 \*.shtml)-->  <mvc:mapping path="/\*\*/\*"/>  <!-- 拦截器类 -->  <bean class="cn.itcast.core.web.springmvc.SpringmvcInterceptor"></bean>  </mvc:interceptor>  </mvc:interceptors> |

## 拦截器类（判断用户是否登陆、同时传递到页面一个标记（用户是否登陆）（在babasport-front中）

|  |
| --- |
| /\*\*  \* 拦截器  \* 上下文  \* 处理请求路径（有些页面是不拦截 、而有些是拦截）  \* 用户是否登陆  \* @author lx  \*  \*/  public class SpringmvcInterceptor implements HandlerInterceptor{  //注入Session提供 类  @Autowired  private SessionProvider sessionProvider;  //进入Handler之前  public boolean preHandle(HttpServletRequest request,  HttpServletResponse response, Object handler) throws Exception {  // TODO Auto-generated method stub  //判断用户是否登陆  String username = sessionProvider.getAttribute(request, response, Constants.USER\_SESSION);  if(null != username){  //没有Model 但有Request  //传递页面一个标记 （是否登陆）  request.setAttribute("isLogin", true);  }else{  //传递页面一个标记 （是否登陆）  request.setAttribute("isLogin", false);  }  //正常进入Handler true  //不让进入Handler false  return true;  }  //进入Handler之后  public void postHandle(HttpServletRequest request,  HttpServletResponse response, Object handler,  ModelAndView modelAndView) throws Exception {  // TODO Auto-generated method stub    }  //页面渲染后  public void afterCompletion(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex)  throws Exception {  // TODO Auto-generated method stub    }  } |

## 页面上判断传递是否登陆标记

|  |
| --- |
| <c:if test="${!isLogin }">  <li class="dev"><a href="javascript:void(0)" onclick="login()" title="登陆">[登陆]</a></li>  <li class="dev"><a href="javascript:void(0)" onclick="register()" title="免费注册">[免费注册]</a></li>  </c:if>  <c:if test="${isLogin }">  <li class="dev"><a href="javascript:void(0)" onclick="logout()" title="退出">[退出]</a></li>  <li class="dev"><a href="javascript:void(0)" onclick="myOrder()" title="我的订单">我的订单</a></li>  </c:if> |

# 结算

## 结算按钮的请求路径设置

/buyer/trueBuy.shtml

## 判断用户是否登陆

如果没有登陆，跳转到登陆页面去登陆，登陆完成后，跳转回购物车页面，再让用户去结算

### 结算按钮的请求路径

|  |
| --- |
| //结算  function trueBuy(){  window.location.href = "/buyer/trueBuy.shtml?returnUrl=" + window.location.pathname;  } |

### Springmvc拦截器中

|  |
| --- |
| //判断用户是否登陆  String username = sessionProvider.getAttribute(request, response, Constants.USER\_SESSION);  //判断当前的请求路径是否必须登陆  //URL：http://localhost:8080/buyer/trueBuy.shtml  //URI:/buyer/trueBuy.shtml  String requestURI = request.getRequestURI();  if(requestURI.startsWith(URI\_INTERCEPTOR)){  //必须登陆  if(null != username){  //没有Model 但有Request  //传递页面一个标记 （是否登陆）  request.setAttribute("isLogin", true);  }else{  //传递页面一个标记 （是否登陆）  request.setAttribute("isLogin", false);  //重定向到登陆页面  response.sendRedirect("/shopping/login.shtml?returnUrl=" +  request.getParameter("returnUrl"));  //不能进入Handler方法  return false;  }  }else{  //可登可不登陆  if(null != username){  //没有Model 但有Request  //传递页面一个标记 （是否登陆）  request.setAttribute("isLogin", true);  }else{  //传递页面一个标记 （是否登陆）  request.setAttribute("isLogin", false);  }  }    }    //正常进入Handler true  //不让进入Handler false  return true; |

## 判断购物车中是否有商品

防止用户提交订单之后或删除购物车之后，点击浏览器上的返回按钮

|  |
| --- |
| //结算 去提交订单页面  @RequestMapping(value = "/buyer/trueBuy.shtml")  public String trueBuy(HttpServletRequest request,HttpServletResponse response){  //判断 购物车中是否有商品  //判断用户是否登陆  String username = sessionProvider.getAttribute(request, response, Constants.USER\_SESSION);  //从Reds中取此用户的购物车  Jedis jedis = jedisPool.getResource();  List<String> lrange = jedis.lrange("buyerCart:" + username, 0, -1);  if(null != lrange && lrange.size() > 0){    }else{  //此用户购物车中无商品  //刷新购物车  return "redirect:/shopping/buyerCart.shtml";  }  //正常情况下  return "product/productOrder";  } |

## 判断购物车中的商品是否有库存

库存定义：库存为0或库存小于购物数量都视为无货，刷新购物车页面，显示商品的状态为无货

|  |
| --- |
| //结算 去提交订单页面  @RequestMapping(value = "/buyer/trueBuy.shtml")  public String trueBuy(HttpServletRequest request,HttpServletResponse response,Model model){  //判断 购物车中是否有商品  //判断用户是否登陆  String username = sessionProvider.getAttribute(request, response, Constants.USER\_SESSION);  //从Reds中取此用户的购物车  Jedis jedis = jedisPool.getResource();  List<String> lrange = jedis.lrange("buyerCart:" + username, 0, -1);  if(null != lrange && lrange.size() > 0){  //创建购物车  BuyerCart buyerCart = new BuyerCart();    //创建判断购物车中是否有无货的  Boolean flag = true;    //有商品、、判断商品是否有库存  for (String l : lrange) {  //创建 Sku  Sku sku = new Sku();  sku.setId(Long.parseLong(l));  //创建购物项  BuyerItem item = new BuyerItem();  item.setSku(sku);  //设置购买数量  item.setAmount(Integer.parseInt(jedis.hget("buyerItem:" + username, l)));  //设置是否有货标识  Sku s = skuService.selectSkuBySkuId(Long.parseLong(l));  if(s.getStock() < item.getAmount()){  //设置无货标识  item.setIsHave(false);  //有无货的  flag = false;  }  //把购物项添加到购物车中  buyerCart.addItem(item);  }  //购物车数据装满  //把购物车数据装满  //购物车中有购物项(有商品 （之前Cookie里的。也可以当前新追加的）  //购物车中无购物项（购物项是空的）  List<BuyerItem> items = buyerCart.getItems();  if(items.size() > 0){  for (BuyerItem buyerItem : items) {  //通过SkuID查询（SKu表、颜色表、商品表、图片表）  Sku sku = skuService.selectSkuById(buyerItem.getSku().getId());  //设置全数据的SKu到购物项中  buyerItem.setSku(sku);  }  }  //跳转页面  if(flag){  //全有货  //正常情况下  return "product/productOrder";  }else{  //有无货的商品存在  //将购物车回显到页面  model.addAttribute("buyerCart", buyerCart);  return "product/cart";  }    }else{  //此用户购物车中无商品  //刷新购物车  return "redirect:/shopping/buyerCart.shtml";  }  } |

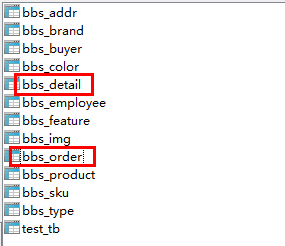
## 加载提交订单页面的数据

收货地址、购物车回显

跳转提交订单页面

# 提交订单

## 订单表及订单详情表



## 订单表重要字段

订单ID 由Redis服务器生成

购物车小计

运费

订单金额（商品金额）

总金额

支付方式 （页面选择后传递到后台）

支付要求 （页面选择后传递到后台）

支付状态 到付、待付款

订单状态 提交订单

用户ID 登陆的用户

生成时间 当前时间

电话确认（页面选择后传递到后台）

送货时间（页面选择后传递到后台）

## 订单详情表重要字段

ID

订单表ID

商品ID

商品名称

颜色名

尺码

价格

购买的数量

## 提交订单按钮

### OrderServiceImpl

|  |
| --- |
| /\*\*  \* 保存订单  \* 保存订单详情  \* @author lx  \*  \*/  @Service("orderService")  @Transactional  public class OrderServiceImpl implements OrderService {  @Autowired  private Jedis jedis;  @Autowired  private OrderDao orderDao;  @Autowired  private DetailDao detailDao;    //保存订单  public void insertOrder(Order order,BuyerCart buyerCart){  // 保存订单表  //生成订单号  Long id = jedis.incr("oid");  order.setId(id);  //运费  order.setDeliverFee(buyerCart.getFee());  //订单金额  order.setOrderPrice(buyerCart.getProductPrice());  //总金额  order.setTotalFee(buyerCart.getTotalPrice());  //判断是否为到付  if(order.getPaymentWay() == 0){  //到付 0  order.setIsPaiy(0);    }else{  //待付款 1  order.setIsPaiy(1);  }  //订单状态  order.setOrderState(0);  //订单生成时间  order.setCreateDate(new Date());  //保存订单  orderDao.insertSelective(order);    // 保存订单详情表  List<BuyerItem> items = buyerCart.getItems();  //遍历购物项  for (BuyerItem buyerItem : items) {  //订单详情对象  Detail detail = new Detail();  //设置订单ID  detail.setOrderId(id);  //商品ID  detail.setProductId(buyerItem.getSku().getProductId());  //商品名称  detail.setProductName(buyerItem.getSku().getProduct().getName());  //颜色名  detail.setColor(buyerItem.getSku().getColor().getName());  //尺码  detail.setSize(buyerItem.getSku().getSize());  //价格  detail.setPrice(buyerItem.getSku().getPrice());  //购买的数量  detail.setAmount(buyerItem.getAmount());  //保存订单详情表  detailDao.insertSelective(detail);  }    }  } |

### OrderController

|  |
| --- |
| package cn.itcast.core.controller;  import java.util.List;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import redis.clients.jedis.Jedis;  import redis.clients.jedis.JedisPool;  import cn.itcast.common.web.session.SessionProvider;  import cn.itcast.core.bean.BuyerCart;  import cn.itcast.core.bean.BuyerItem;  import cn.itcast.core.bean.order.Order;  import cn.itcast.core.bean.product.Sku;  import cn.itcast.core.service.order.OrderService;  import cn.itcast.core.service.product.SkuService;  import cn.itcast.core.web.Constants;  /\*\*  \* 提交订单  \*  \* @author lx  \*  \*/  @Controller  public class OrderController {  @Autowired  private SessionProvider sessionProvider;  @Autowired  private JedisPool jedisPool;  @Autowired  private SkuService skuService;  @Autowired  private OrderService orderService;  // 提交订单按钮  @RequestMapping(value = "/buyer/confirmOrder.shtml")  public String confirmOrder(Order order, HttpServletRequest request,  HttpServletResponse response) {  // 用户名取购物车  // 判断用户是否登陆  String username = sessionProvider.getAttribute(request, response,  Constants.USER\_SESSION);    //订单所属用户  order.setBuyerId(username);    // 从Reds中取此用户的购物车  Jedis jedis = jedisPool.getResource();  List<String> lrange = jedis.lrange("buyerCart:" + username, 0, -1);  if (null != lrange && lrange.size() > 0) {  // 创建购物车  BuyerCart buyerCart = new BuyerCart();  // 有商品、、判断商品是否有库存  for (String l : lrange) {  // 创建 Sku  Sku sku = new Sku();  sku.setId(Long.parseLong(l));  // 创建购物项  BuyerItem item = new BuyerItem();  item.setSku(sku);  // 设置购买数量  item.setAmount(Integer.parseInt(jedis.hget("buyerItem:"  + username, l)));  // 把购物项添加到购物车中  buyerCart.addItem(item);  }  // 购物车数据装满  // 把购物车数据装满  // 购物车中有购物项(有商品 （之前Cookie里的。也可以当前新追加的）  // 购物车中无购物项（购物项是空的）  List<BuyerItem> items = buyerCart.getItems();  if (items.size() > 0) {  for (BuyerItem buyerItem : items) {  // 通过SkuID查询（SKu表、颜色表、商品表、图片表）  Sku sku = skuService.selectSkuById(buyerItem.getSku()  .getId());  // 设置全数据的SKu到购物项中  buyerItem.setSku(sku);  }  }  //保存  orderService.insertOrder(order, buyerCart);  //清空购物车  jedis.del("buyerCart:" + username,"buyerItem:" + username);  }  return "product/confirmOrder";  }  } |

# 企业整体架构分析

企业整个架构图.vsd



# 企业高并发成熟解决方案

## 高并发

同一时间内，有多个进程或线程，访问同一资源，叫高并发

## 高可用

（HA）必须有备机（高度的可以使用、安全）

## 什么叫负载均衡？

由于目前现有网络的各个核心部分随着业务量的提高，访问量和数据流量的快速增长，其处理能力和计算强度也相应地增大，使得单一的服务器设备根本无法承担。在此情况下，如果扔掉现有设备去做大量的硬件升级，这样将造成现有资源的浪费，而且如果再面临下一次业务量的提升时，这又将导致再一次硬件升级的高额成本投入，甚至性能再卓越的设备也不能满足当前业务量增长的需求。   
　　针对此情况而衍生出来的一种廉价有效透明的方法以扩展现有网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性的技术就是负载均衡（Load Balance）。

## 负载均衡的种类

1）一种是通过硬件来进行解决，常见的硬件有NetScaler、F5、Radware和Array等商用的负载均衡器，但是它们是比较昂贵的

2）一种是通过软件来进行解决的，常见的软件有LVS、Nginx、apache等,它们是基于Linux系统并且开源的负载均衡策略.

## 什么是Apache + JK?

Apache是世界使用排名第一的Web[服务器](http://baike.baidu.com/view/899.htm)软件。它可以运行在几乎所有广泛使用的[计算机平台](http://baike.baidu.com/view/2269685.htm)上，由于其[跨平台](http://baike.baidu.com/view/469855.htm)和安全性被广泛使用，是最流行的Web服务器端软件

JK是apache提供的一款为解决大量请求而分流处理的开源插件

## 什么是nginx?

**Nginx**（发音同 engine x）是一款轻量级的[Web](http://baike.baidu.com/view/3912.htm) 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个BSD-like 协议下发行。由俄罗斯的程序设计师Igor Sysoev(伊戈尔·西索夫)所开发，供俄国大型的入口网站及搜索引擎Rambler(漫步者)（俄文：Рамблер）使用。其特点是占有[内存](http://baike.baidu.com/view/1082.htm)少，[并发](http://baike.baidu.com/view/684757.htm)能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：[新浪](http://baike.baidu.com/view/2410.htm)、[网易](http://baike.baidu.com/view/6043.htm)、 [腾讯](http://baike.baidu.com/view/1591.htm)等。

优点：

1:可运行linux,并有 [Windows](http://baike.baidu.com/view/4821.htm) 移植版。

2:在高连接并发的情况下，Nginx是Apache服务器不错的替代品Nginx在[美国](http://baike.baidu.com/view/2398.htm)是做虚拟主机生意的老板们经常选择的软件平台之一。能够支持高达 50,000 个并发连接数的响应

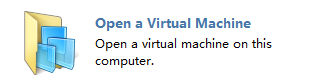
## 什么是LVS?

LVS的英文全称是Linux Virtual Server，即Linux虚拟服务器。它是我们国家的章文嵩博士的一个开源项目。在linux内核2.6中，它已经成为内核的一部分，在此之前的内核版本则需要重新编译内核。

## 安装Nginx之前说明

* 四台Centos6.0以上机器
* 一台tomcat 1
* 一台Tomcat 2
* 一台负载均衡（Nginx）（主机）
* 一台负载均衡（Nginx）（备机）

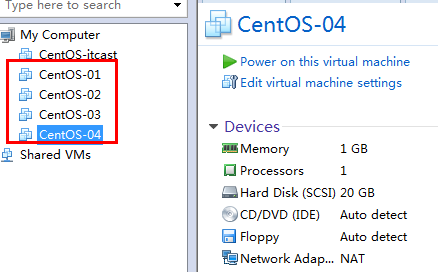
## 打开四台机器



## 重命名Centos机器



## 准备工作完成



## 四个IP

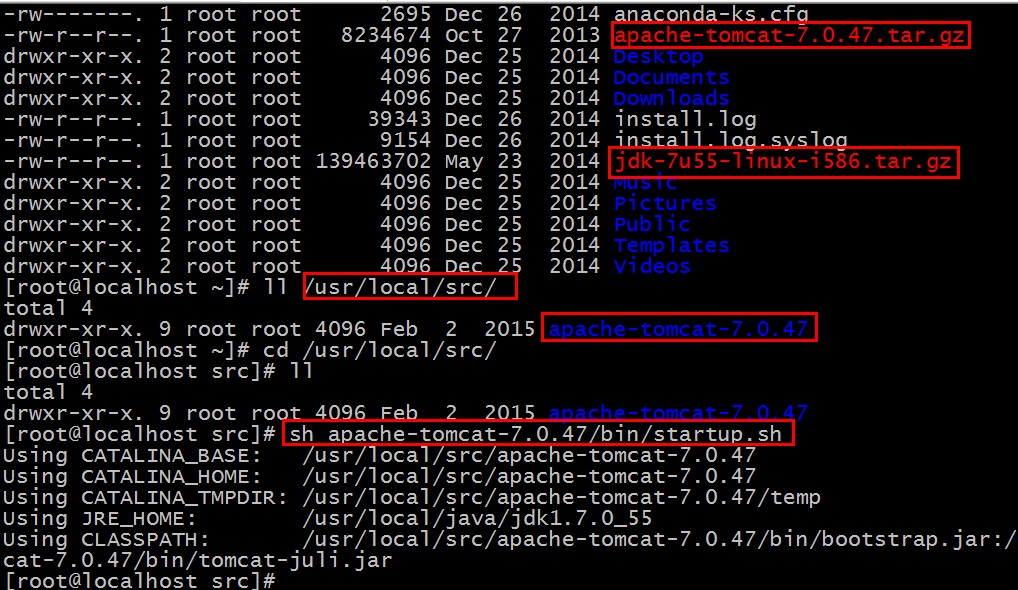
192.168.200.133 ： 负载均衡备机

192.168.200.134 ： 负载均衡主机

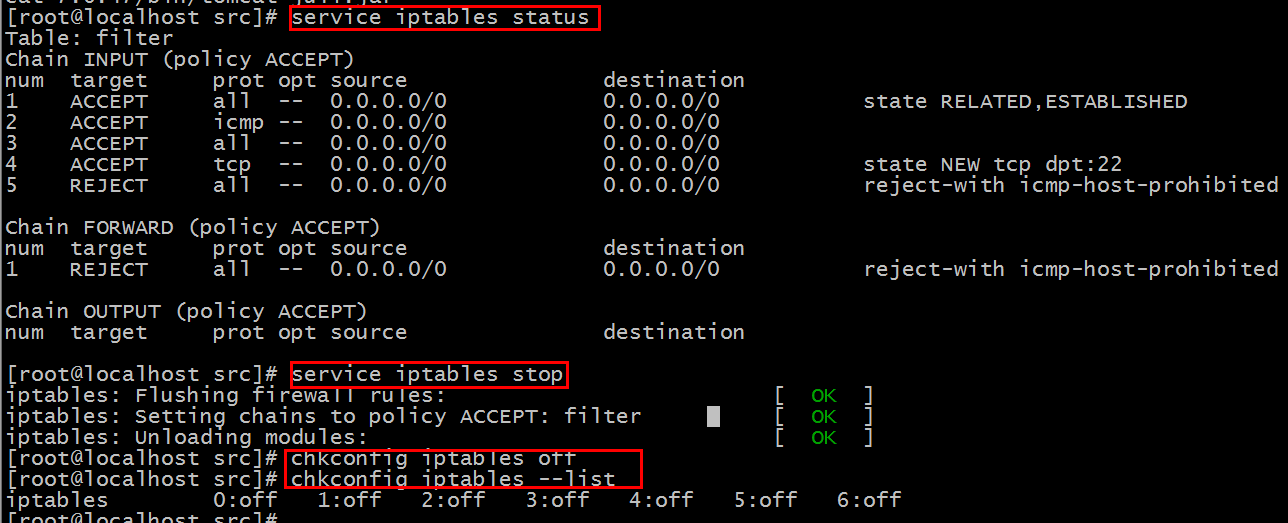
192.168.200.135 ： Tomcat 1

192.168.200.136 ： Tomcat 2

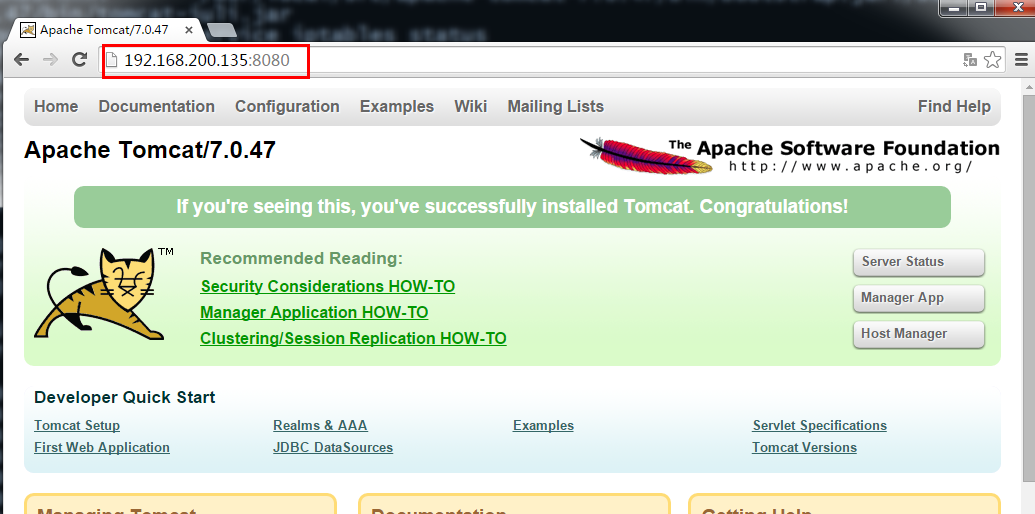
## 安装二台Tomcat



## 关闭防火墙



## 访问成功



## 负载均衡主机搭建（Nginx）

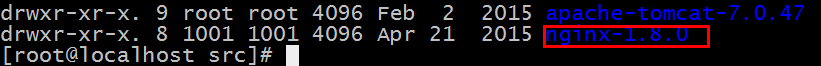
请查看 资料\nginx高并发解决\教案\nginx-1.0.docx

将nginx-1.8.0.tar.gz拷贝至linux服务器。

解压：

tar -zxvf nginx-1.8.0.tar.gz –C /usr/local/src/

cd cd /usr/local/src/



进入nginx-1.8.0

1. configure

./configure --help查询详细参数（参考本教程附录部分：nginx编译参数）

参数设置如下：

./configure \

--prefix=/usr/local/nginx \

--pid-path=/var/run/nginx/nginx.pid \

--lock-path=/var/lock/nginx.lock \

--error-log-path=/var/log/nginx/error.log \

--http-log-path=/var/log/nginx/access.log \

--with-http\_gzip\_static\_module \

--http-client-body-temp-path=/var/temp/nginx/client \

--http-proxy-temp-path=/var/temp/nginx/proxy \

--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \

--http-uwsgi-temp-path=/var/temp/nginx/uwsgi \

--http-scgi-temp-path=/var/temp/nginx/scgi

编译

make

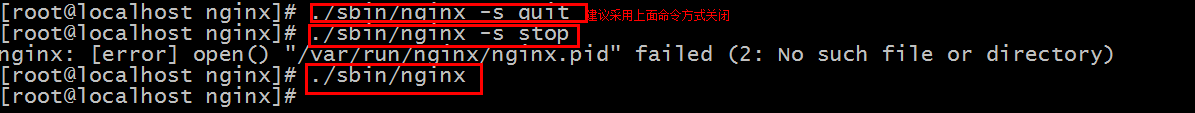
**注意：上边将临时文件目录指定为/var/temp/nginx，需要在/var下创建temp及nginx目录**

mkdir -p /var/temp/nginx

安装

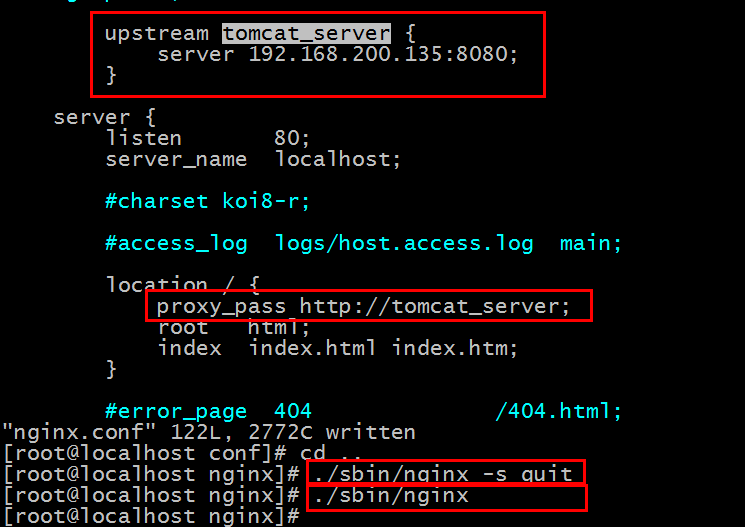
make install

## Nginx的启动及关闭

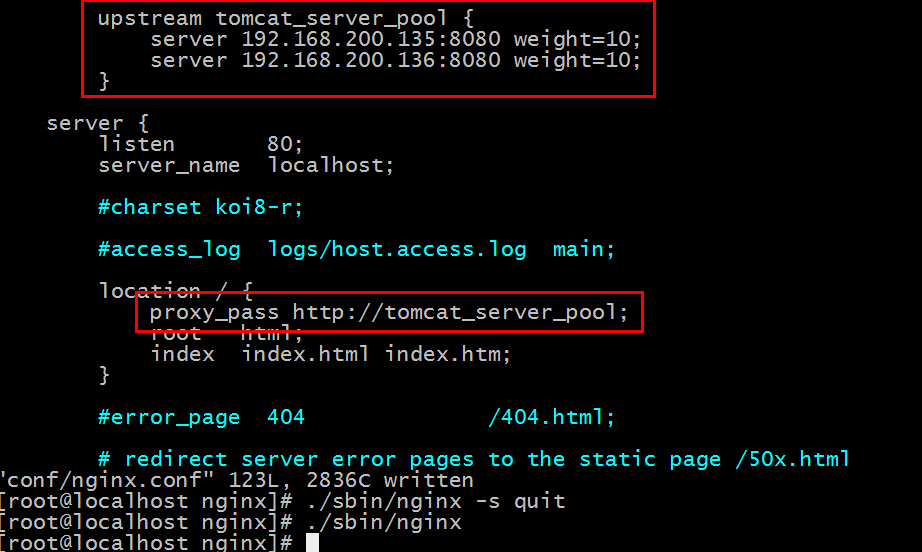


## 配置nginx反向代理（反向代理服务器）

在nginx.conf文件中配置



## 配置Nginx的负载均衡



## 测试Nginx的三种能力

转发、故障移除、恢复添加

## 搭建Nignx的备机

### 安装Nginx （略）

### 在二台Nginx上安装 Keepalived软件

资料\nginx高并发解决\高可用\keepalived\安装包



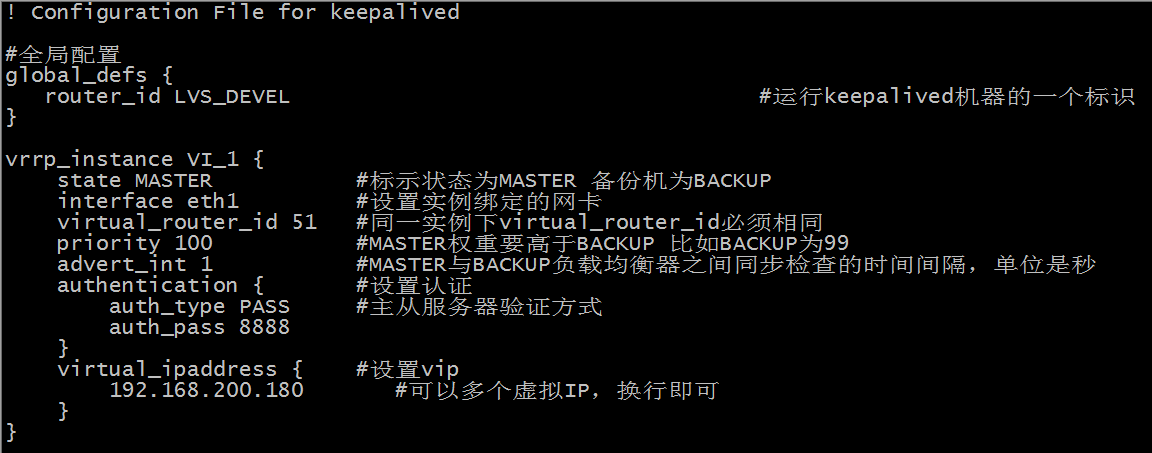
rpm -ivh keepalived-1.2.13-5.el6\_6.i686.rpm

查看安装目录

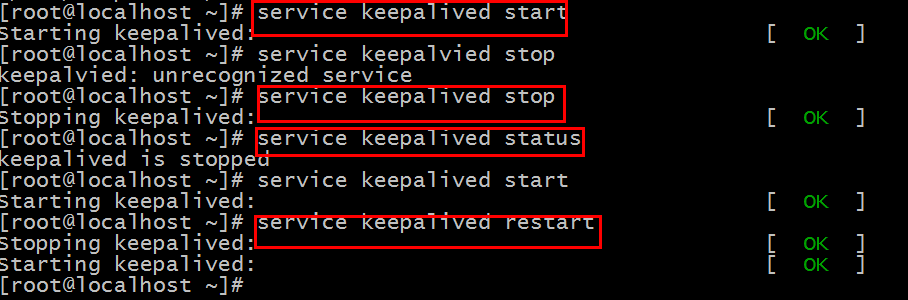
rpm -ql keepalived

### 配置keepalived 的主机

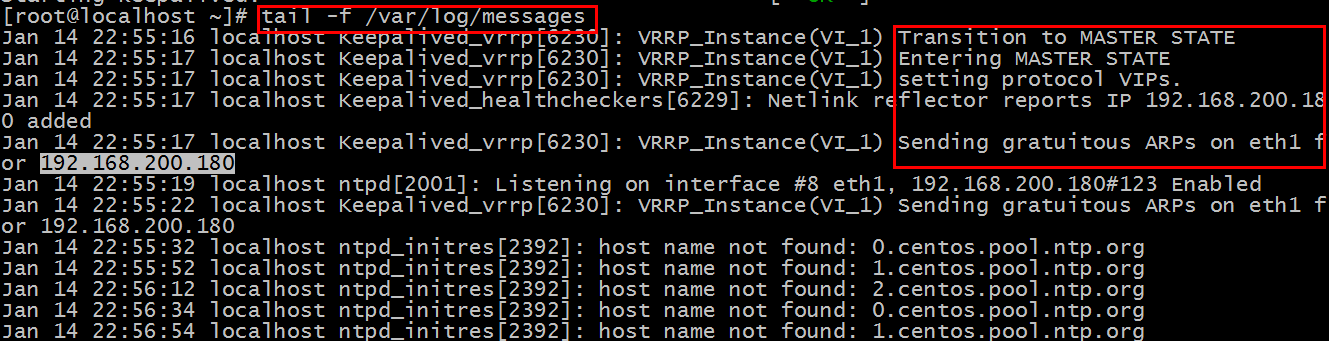




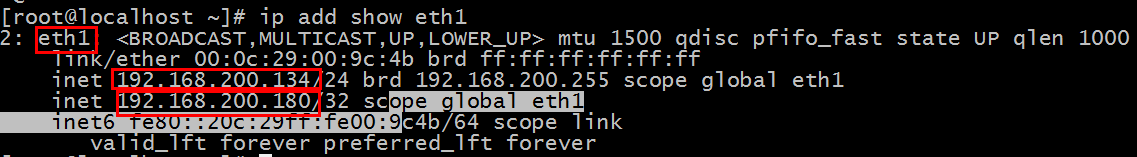
启动keepalived



查看keepalived日志



查看eth1网卡接口上是否有VIP

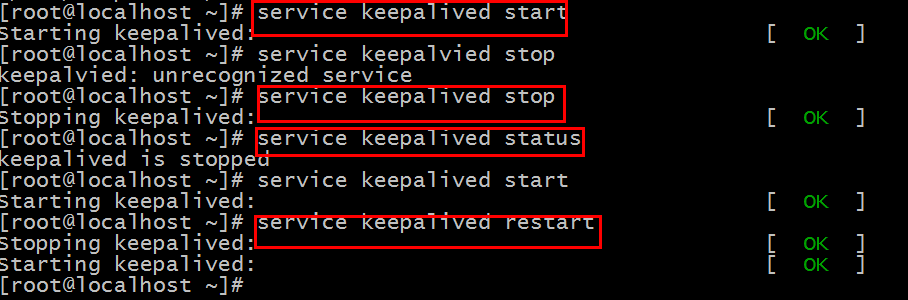


### 配置keepalived 备机





启动keepalived



查看日志



## 问题 （Nginx与keepalived不同时存在）

keepalived是通过检测keepalived进程是否存在判断服务器是否宕机，如果keepalived进程在但是nginx进程不在了那么keepalived是不会做主备切换，所以我们需要写个脚本来监控nginx进程是否存在，如果nginx不存在就将keepalived进程杀掉。

## 解决（检查Nginx进程、并关闭Keepalived进程

#!/bin/bash

# 如果进程中没有nginx则将keepalived进程kill掉

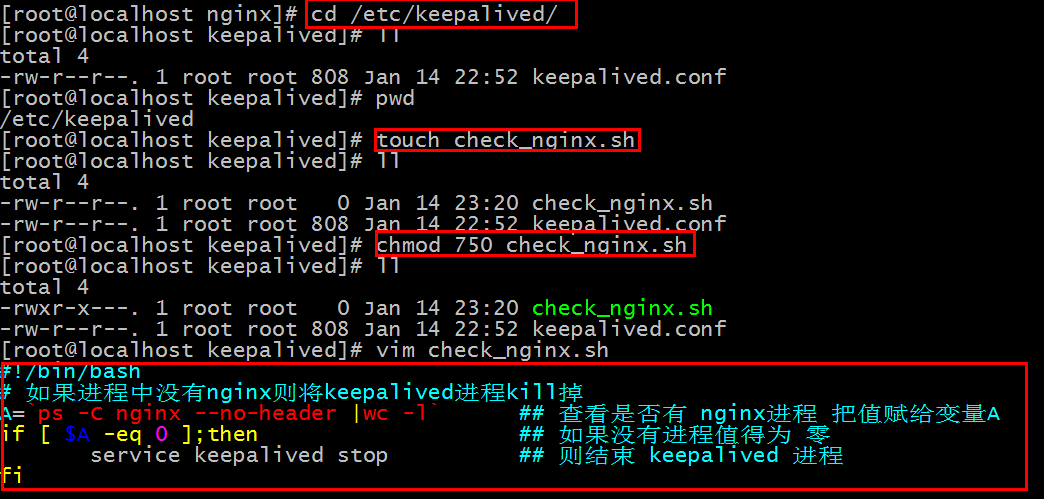
A=`ps -C nginx --no-header |wc -l` ## 查看是否有 nginx进程 把值赋给变量A

if [ $A -eq 0 ];then ## 如果没有进程值得为 零

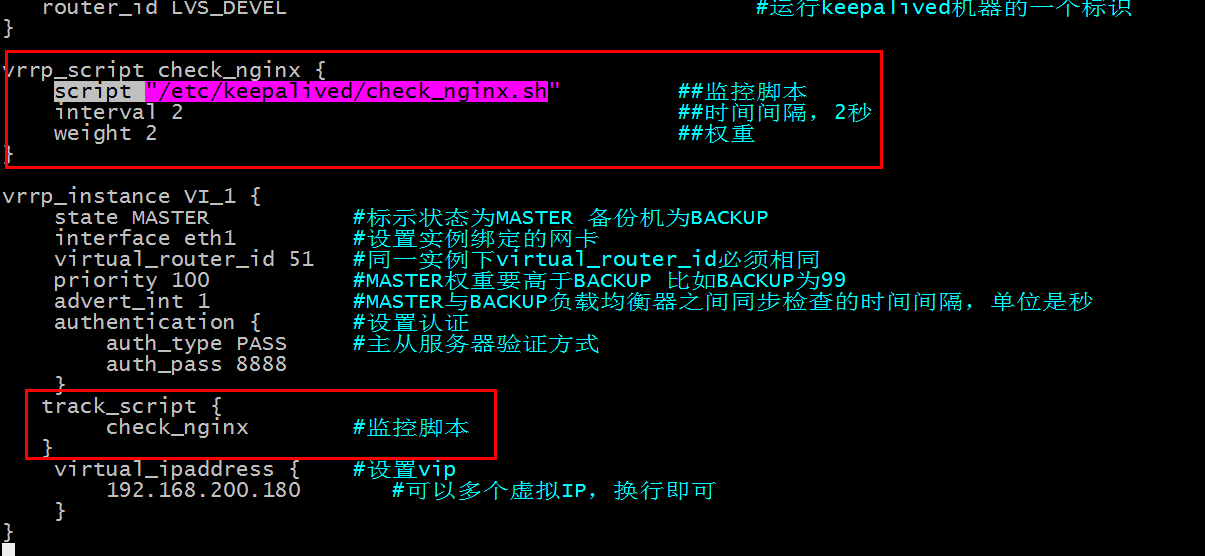
service keepalived stop ## 则结束 keepalived 进程

fi

## 配置check\_nginx.sh



## 由keepalived 主机来执行上面的脚本



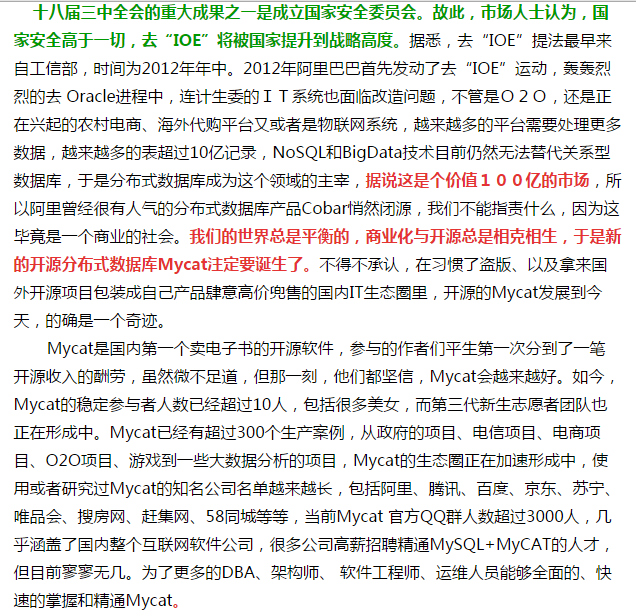
# 数据库集群方案

Mycat官方网站

<http://mycat.io/>



## Mycat的诞生



## Mycat安装

## Mycat的使用

# 项目总结

## 项目之外介绍

### 公司的性质是什么？

外包（注意：最好说是外包公司，否则下面很多问题不好回答）

### 你们开发的项目上线了吗？

上线了

### 你们的项目能访问吗？域名是多少？

由于是外包项目、项目卖给谁都不知道、域名也不知道

### 你们的项目卖了多少钱？

本人只是开发人员、并没有看到公司的合同、不知道卖了多少钱

### 公司开发人员多少个？

我刚去公司时，只有5个人，我走时有23人

### 分几个项目组？

分二个项目组

### 每个项目组有多少人？

一个组有10人、一个组有13人

### 你在哪个组？

我在10人组

### 你主要负责什么模块？

商品模块

### 你在你们组中处于什么地位？

主要负责开发商品模块的同时，帮助同事完成订单模块及购物车模块

### 你们项目的开发周期？

预计开发6个月

实际开发9个月

## 项目本身介绍

### 系统后台（商品编辑、审核、上架）

商品编辑：商品的添加、删除、修改、查询（带条件、分页）

审核：人工审核（审核前台商品详情页面）

上架：更改商品状态、添加商品信息到Solr服务器、静态化商品页面

注意：商品编号由Reids统一生成

### 系统前台（商品检索、商品详情、购物车、结算、提交订单）

商品检索：采用Solr服务器进行全文检索、支持扩展词、停用词、自定义字段、IK分词器、关键词高亮、过滤条件存储在Redis集群中

商品详情：采用静态化技术

购物车：非登陆时采用Cookie进行存储、并保存七天、登陆时采用Redis进行存储、永久保存

结算：1）要求用户必须登陆2）购物车必须有商品3）购物车中商品必须有库存

是否有货约定：购物数量大于库存数量时，视为无货。反之有货

提交订单：保存订单表及订单详情表（注意：订单表有商品表或库存表毫无关联，因为订单是快照）

### 单点登陆系统