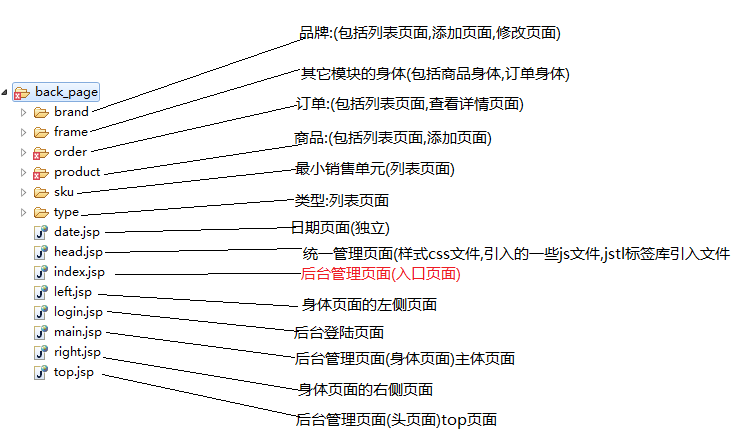
# 搭建后台管理框架

## 后台页面----目录结构



## 第一步:将前台,后台静态页面从webapp中移动到WEB-INF下

## 第二步:由于目录结构发生变化,每一个页面中<include>标签都需要修改

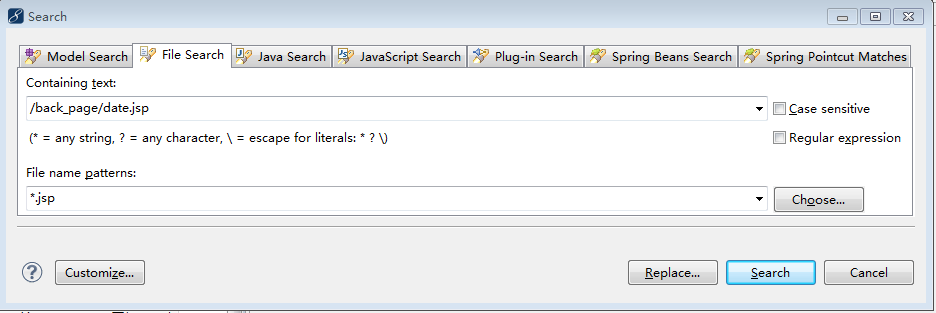
**修改前**

**<%@ include file="/back\_page/head.jsp" %>**

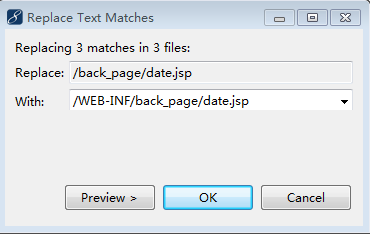
**修改后**

**<%@ include file="/WEB-INF/back\_page/head.jsp" %>**

## 第三步:批量替换快捷键 Ctrl+H



## 第四步:点击Replace键



## 第五步:点击OK完成

## 第六步:创建cn.itcast.core.controller包

## 第七步:创建CenterController类

* index():返回index 对应/WEB-INF/back\_page/index.jsp页面
* top(): 返回top对应/WEB-INF/back\_page/top.jsp页面
* main(): 返回main对应/WEB-INF/back\_page/main.jsp页面
* left(): 返回left对应/WEB-INF/back\_page/left.jsp页面
* right(): 返回right对应/WEB-INF/back\_page/rihgt.jsp页面

## 第八步:创建FrameController类

* productMain():返回frame/product\_main对应/WEB-INF/back\_page/frame/product\_main.jsp页面
* productLeft (): 返回product\_left对应/WEB-INF/back\_page/frame/product\_left.jsp页面
* orderMain (): 返回order\_main对应/WEB-INF/back\_page/frame/order\_main.jsp页面
* orderLeft (): 返回order\_left对应/WEB-INF/back\_page/frame/order\_left.jsp页面

## 第九步:创建ProductController类

* list():返回 list对应/WEB-INF/back\_page/product/list.jsp页面

## 第十步:创建OrderController类

* list():返回 list对应/WEB-INF/back\_page/order/list.jsp页面

# 品牌管理

## 创建品牌表 : bbs\_brand

**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';**

## 第一步:创建cn.itcast.core.web.Constants

**public interface Constants {**

**/\*\***

**\* 图片服务器**

**\*/**

**public static final String IMAGE\_URL = "http://localhost:8088/image-web/";**

**}**

## 创建cn.itcast.core.bean.product.Brand对象

**/\*\***

**\* 品牌对象**

**\* @author lixu**

**\* @Date [2014-3-28 下午04:38:53]**

**\*/**

**public class Brand implements Serializable {**

**/\*\***

**\* 序列化ID**

**\*/**

**private static final long serialVersionUID = 1L;**

**private Long id;**

**private String name;**

**private String description;**

**private String imgUrl;**

**private String webSite;**

**private Integer sort;**

**private Integer isDisplay;**

**//获取全路径**

**public String getAllUrl(){**

**return Constants.IMAGE\_URL + imgUrl;**

**}**

**//set get 省略**

## 创建cn.itcast.core.bean.product.BrandQuery对象

**/\*\***

**\* 品牌条件对象**

**\* @author lx**

**\***

**\*/**

**public class BrandQuery implements Serializable {**

**/\*\***

**\***

**\*/**

**private static final long serialVersionUID = 1L;**

**private Long id;**

**private String name;**

**private String description;**

**private String imgUrl;**

**private Integer sort;**

**private Integer isDisplay;**

**//set get 省略**

**//分页用到的 Limit参数**

**//当前页**

**private Integer pageNo;**

**//每页数 5条**

**private Integer pageSize = 10;**

**//开始行 == (pageNO - 1) \* pageSize;**

**private Integer startRow;**

**public Integer getPageNo() {**

**return pageNo;**

**}**

**public void setPageNo(Integer pageNo) {**

**//计算开始行**

**this.startRow = (pageNo - 1) \* pageSize;**

**this.pageNo = pageNo;**

**}**

**public Integer getPageSize() {**

**return pageSize;**

**}**

**public void setPageSize(Integer pageSize) {**

**//计算开始行**

**this.startRow = (pageNo - 1) \* pageSize;**

**this.pageSize = pageSize;**

**}**

**public Integer getStartRow() {**

**return startRow;**

**}**

**public void setStartRow(Integer startRow) {**

**this.startRow = startRow;**

**}**

## 第二步:创建Dao.xml

cn.itcast.core.dao.product.BrandDao.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">**

**<mapper namespace="cn.itcast.core.dao.product.BrandDao">**

**</mapper>**

## 第三步:创建Dao接口

cn.itcast.core.dao.product.BrandDao

**/\*\***

**\* 数据交互层**

**\* @author lx**

**\***

**\*/**

**public interface BrandDao {**

**}**

## 第四步:创建Service接口

cn.itcast.core.service.product.BrandService

**/\*\***

**\* 事务层,逻辑开发层 接口**

**\* @author lx**

**\***

**\*/**

**public interface BrandService {**

**}**

## 第五步:创建ServiceImpl实现类

cn.itcast.core.service.product.BrandServiceImpl

**/\*\***

**\* 事务层,逻辑开发层 实现类**

**\* @author lx**

**\***

**\*/**

**@Service**

**@Transactional**

**public class BrandServiceImpl implements BrandService{**

**}**

## 第六步:创建Controller

cn.itcast.core.controller.admin.BrandController

**/\*\***

**\* 品牌管理**

**\* @author lx**

**\***

**\*/**

**@Controller**

**public class BrandController {**

**}**

# 品牌列表之前的准备

## 如果出现中文乱码

如果名称条件出现中文乱码

在tomcat/conf/server.xml 的第70行,输入以下内容

**<Connector port="8080" protocol="HTTP/1.1"**

**connectionTimeout="20000"**

**redirectPort="8443" URIEncoding="UTF-8" />**

## 传递条件到Springmvc的方法形参时，应该去掉前后空格，同时如果传递的是空串，可转成Null值

建议做去前后空格处理

在classpath:springmvc-back.xml，springmvc-front.xml中配置如下内容

**<!-- 注解开发第二种配置文式 -->**

**<mvc:annotation-driven conversion-service="conversionService" />**

**<!-- 参数格式 字符串类型 yyyy-MM-dd HH-mm-ss 格式化 -->**

**<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">**

**<property name="converters">**

**<list>**

**<!-- 日期转换 -->**

**<bean class="cn.itcast.common.convertion.CustomDateConverter"/>**

**<!-- 参数 传递 前后去掉空格 -->**

**<bean class="cn.itcast.common.convertion.CustomTrimConverter"/>**

**</list>**

**</property>**

**</bean>**

## 去前后空格转换类

创建cn.itcast.common.convertion.CustomTrimConverter.java类，书写内容如下

**/\*\***

**\* 去掉前后空格**

**\* @author lx**

**\***

**\*/**

**public class CustomTrimConverter implements Converter<String, String> {**

**@Override**

**public String convert(String source) {**

**try {**

**if(null != source){**

**// " "**

**source = source.trim();// ""**

**if(source.equals("")){**

**return null;**

**}else{**

**return source;**

**}**

**}**

**} catch (Exception e) {**

**// TODO: handle exception**

**}**

**return null;**

**}**

**}**

## 转换日期类

创建cn.itcast.common.convertion.CustomDateConverter.java类，书写内容如下

**/\*\***

**\* 日期继承Converter接口**

**\* @author lx**

**\***

**\*/**

**public class CustomDateConverter implements Converter<String, Date>{**

**@Override**

**public Date convert(String source) {**

**//配置转换**

**try {**

**//判断是否为Null**

**if(null != source){**

**return new SimpleDateFormat("yyyy-MM-dd HH-mm-ss").parse(source);**

**}**

**} catch (Exception e) {**

**// TODO: handle exception**

**}**

**return null;**

**}**

**}**

# 品牌列表(数据加载)

## 导入测试数据

**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');**

## 第一步:编写Sql语句

**select id,name,description,img\_url,web\_site,sort,is\_display from bbs\_brand where is\_display = 1 and name = '依琦莲'**

**order by id desc**

**limit 0,5**

## 第二步:编写BrandDao.xml文件

**<?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">**

**<mapper namespace="cn.itcast.core.dao.product.BrandDao">**

**<!-- resultMap -->**

**<resultMap type="Brand" id="brand">**

**<result column="id" property="id"/>**

**<result column="name" property="name"/>**

**<result column="description" property="description"/>**

**<result column="img\_url" property="imgUrl"/>**

**<result column="sort" property="sort"/>**

**<result column="is\_display" property="isDisplay"/>**

**</resultMap>**

**<!-- 查询品牌 get\* -->**

**<select id="getBrandListWithPage" parameterType="Brand" resultMap="brand">**

**select id , name ,description,img\_url,sort,is\_display**

**from bbs\_brand**

**<where>**

**<if test="isDisplay != null">**

**is\_display = #{isDisplay}**

**</if>**

**<if test="name != null">**

**and name = #{name}**

**</if>**

**</where>**

**order by id desc**

**limit #{startRow},#{pageSize}**

**</select>**

**<!-- 查询总记录数 -->**

**<select id="getBrandCount" parameterType="Brand" resultType="Integer">**

**select count(1)**

**from bbs\_brand**

**<where>**

**<if test="isDisplay != null">**

**is\_display = #{isDisplay}**

**</if>**

**<if test="name != null">**

**and name = #{name}**

**</if>**

**</where>**

**</select></mapper>**

## 第三步:编写BrandDao接口

**/\*\***

**\* 数据交互层**

**\* @author lx**

**\***

**\*/**

**public interface BrandDao {**

**//查询所有品牌**

**//List集合**

**public List<Brand> getBrandListWithPage(Brand brand);**

**//查询总记录数**

**public int getBrandCount(Brand brand);**

**}**

## 第四步:编写BrandService接口

**/\*\***

**\* 事务层 逻辑开发层**

**\* @author lx**

**\***

**\*/**

**public interface BrandService {**

**//查询品牌分页对象**

**public Pagination getBrandListWithPage(BrandQuery brandQuery);**

**}**

## 第五步:编写BrandServiceImpl实现类

**/\*\***

**\* 事务层 逻辑开发层**

**\* @author lx**

**\***

**\*/**

**@Service**

**@Transactional**

**public class BrandServiceImpl implements BrandService {**

**@Resource**

**private BrandDao brandDao;**

**@Transactional(readOnly = true)**

**public Pagination getBrandListWithPage(BrandQuery brandQuery){**

**//1:起始页 startRow = (pageNo - 1)\*pageSize**

**//2:每页数**

**//3:总记录数**

**Pagination pagination = new Pagination(brandQuery.getPageNo(),brandQuery.getPageSize(),brandDao.getBrandCount(brandQuery));**

**//防止最后一页数据删除后，分页数据显示不正确**

**brandQuery.setPageNo(pagination.getPageNo());**

**//Brand集合**

**pagination.setList(brandDao.getBrandListWithPage(brandQuery));**

**return pagination;**

**}}**

## 第六步:编写BrandController

**/\*\***

**\* 品牌管理**

**\* 品牌列表**

**\* 品牌添加**

**\* 品牌修改**

**\* 品牌删除（批量） 单（同学）**

**\* @author lx**

**\***

**\*/**

**@Controller**

**public class BrandController {**

**@Autowired**

**private BrandService brandService;**

**//品牌列表**

**@RequestMapping(value = "/brand/list.do")**

**public String list(Integer pageNo,String name,Integer isDisplay,Model model){**

**//Dao Map Service JavaBean**

**BrandQuery brandQuery = new BrandQuery();**

**brandQuery.setPageNo(Pagination.cpn(pageNo));**

**brandQuery.setPageSize(3);**

**StringBuilder params = new StringBuilder();**

**//判断**

**if(null != name){**

**brandQuery.setName(name);**

**params.append("name=").append(name);**

**}**

**if(null != isDisplay){**

**brandQuery.setIsDisplay(isDisplay);**

**params.append("&isDisplay=").append(isDisplay);**

**}else{**

**brandQuery.setIsDisplay(1);**

**params.append("&isDisplay=").append(1);**

**}**

**//Shift + alt + L**

**Pagination pagination = brandService.selectBrandPaginationByQuery(brandQuery);**

**//分页封装**

**String url = "/brand/list.do";**

**pagination.pageView(url, params.toString());**

**model.addAttribute("pagination", pagination);**

**model.addAttribute("name", name);**

**model.addAttribute("isDisplay", isDisplay);**

**model.addAttribute("pageNo", brandQuery.getPageNo());//null 不能小于1**

**return "brand/list";**

**}**

**}**

## 第七步:编写back\_page/brand/list.jsp

**<form action="/brand/list.do" method="post" style="padding-top:5px;">**

**品牌名称: <input type="text" name="name" value="${name }"/>**

**<select name="isDisplay">**

**<option value="1" <c:if test="${isDisplay == 1 }">selected="selected"</c:if> >是</option>**

**<option value="0" <c:if test="${isDisplay == 0 }">selected="selected"</c:if>>不是</option>**

**</select>**

**<input type="submit" class="query" value="查询"/>**

**</form>**

**<c:forEach items="${brands }" var="brand">**

**<tr bgcolor="#ffffff" onmouseover="this.bgColor='#eeeeee'" onmouseout="this.bgColor='#ffffff'">**

**<td><input type="checkbox" name="ids" value="${brand.id }"/></td>**

**<td align="center">${brand.id }</td>**

**<td align="center">${brand.name }</td>**

**<td align="center"><img src="${brand.imgUrl}" width="40" height="40"/></td>**

**<td align="center"></td>**

**<td align="center">99</td>**

**<td align="center"><c:if test="${brand.isDisplay == 1 }">可用</c:if><c:if test="${brand.isDisplay == 0 }">不可用</c:if></td>**

**<td align="center">**

**<a href="javascript:void(0);" class="pn-opt">修改</a> | <a href=" javascript:void(0);" onclick="if(!confirm('您确定删除吗？')) {return false;}" class="pn-opt">删除</a>**

**</td>**

**</tr>**

**</c:forEach>**

**<div class="page pb15">**

**<span class="r inb\_a page\_b">**

**<c:forEach items="${pagination.pageView }" var="page">**

**${page }**

**</c:forEach>**

**</span>**

**</div>**

# 品牌添加

## 位置



## 第一步:编写back\_page/brand/list.jsp

**<div class="box-positon">**

**<div class="rpos">当前位置: 品牌管理 - 列表</div>**

**<form class="ropt">**

**<input class="add" type="button" value="添加"**

**<!-- 编写添加按钮的onclick事件为 toAdd.do -->**

**onclick="javascript:window.location.href='toAdd.do'"/>**

**</form>**

**</div>**

## 第二步:编写BrandController类

**//跳转添加页面**

**@RequestMapping(value = "/brand/toAdd.do")**

**public String toAdd(HttpServletRequest request,HttpServletResponse response,ModelMap model){**

**//指定页面视图**

**return "brand/add";**

**}**

## 第三步:编写back\_page/brand/add.jsp

**<script type="text/javascript" scr="/res/common/js/jquery.form.js"></script>**

**<script type="text/javascript">**

**//采用jquery.form.js异步上传图片,并结合<form>表单**

**function uploadFic(){**

**var options = {**

**//请求路径**

**url = "/upload/uploadPic.do",**

**dataType = "json",**

**type = "post",**

**success : function(data){**

**//处理结果**

**//将相对路径设置给隐藏域中,提交时用**

**$("#imgUrl").val(data.path);**

**//将全路径设置给img标签,显示图片用**

**#("#allImgUrl).attr("src",data.url);**

**}**

**}**

**$("#jvForm").ajaxSubmit(options);**

**}**

**</script>**

**<form id="jvForm" action="add.do" method="post" enctype="multipart/form-data">**

**……**

**<img src="" id="allImgUrl"/>**

**<input type="hidden" name="imgUrl" id="imgUrl"/>**

**<input type="file" name="uploadPic" onchange="uploadPic()"/>**

**</form>**

## 第四步:在springmvc-back.xml配置图片转换器

**<!-- 上传图片转换器 -->**

**<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">**

**<!-- 设置上传文件的最大尺寸为1MB -->**

**<property name="maxUploadSize" value="1048576"/>**

**</bean>**

## 第五步:创建UploadController

cn.itcast.core.controller.admin.UploadController

**//上传图片 通过@RequestParam(required = false) MultipartFile获取图片**

**@RequestMapping(value = "/upload/uploadPic.do")**

**public void uploadBrandPic(@RequestParam MultipartFile uploadPic,HttpServletResponse response** **,HttpServletRequest request){**

**//图片名称生成策略---采用时间格式(精确到毫秒)并追加随机3位(10以内)数字**

**//精确到毫秒**

**DateFormat df = new SimpleDateFormat("yyyyMMddHHmmssSSS");**

**String picName = df.format(new Date());**

**//随机再生成3位 10以内的数**

**Random r = new Random();**

**for(int i=0;i<3;i++){**

**picName += r.nextInt(10);**

**}**

**//获取扩展名**

**String originalFilename = uploadPic.getOriginalFilename();**

**String ext = FilenameUtils.getExtension(originalFilename);**

**//图片上传的路径**

**String path = "upload/" + picName + "." + ext;**

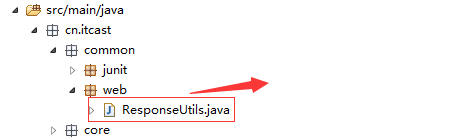
**//上传图片到指定位置 记得先创建upload文件夹 或是程序创建文件夹**

**pic.transferTo(new File(request.getSession().getServletContext().getRealPath("/") + path));**

**}**

## 第十步:创建ResponseUtils

### 位置



### 内容

**/\*\***

**\* Response帮助类**

**\* 支持JSON XML Text**

**\* @author lx**

**\***

**\*/**

**public class ResponseUtils {**

**//发送JSON**

**public void renderJson(HttpServletResponse response,String text){**

**render(response,"application/json;charset=UTF-8",text);**

**}**

**//发送xml**

**public void renderXml(HttpServletResponse response,String text){**

**render(response,"text/xml;charset=UTF-8",text);**

**}**

**//发送text**

**public void renderText(HttpServletResponse response,String text){**

**render(response,"text/plain;charset=UTF-8",text);**

**}**

**//发送**

**public void render(HttpServletResponse response,String contextType,String text){**

**//设置传输类型**

**response.setContentType(contextType);**

**//发送**

**try {**

**response.getWriter().write(text);**

**} catch (IOException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**}**

## 常量cn.itcast.core.web.Constants

**//图片服务器**

**public static final String IMG\_WEB = "http://localhost:8080/";**

## 第十一步: 编写UploadController

**//返回Json数据给页面(包括url回显路径,path保存数据库的路径)**

**JSONObject jo = new JSONObject();**

**jo.put("path", path);**

**//返回的是全路径 Url 常量里写的服务器的域名**

**jo.put("url", Constants.IMG\_WEB + path);**

**ResponseUtils.renderJson(response, jo.toString());**

## 第十二步:查看结果



## 第十三步:保存表单数据

Controller中代码

**//保存表单**

**@RequestMapping(value = "/brand/add.do")**

**public String add(Brand brand,HttpServletRequest request,HttpServletResponse response,ModelMap model){**

**//保存**

**brandService.addBrand(brand);**

**//重定向到品牌列表页面**

**return "redirect:/brand/list.do";**

**}**

Service中代码

**@Resource**

**BrandDao brandDao;**

**//保存数据**

**public Integer addBrand(Brand brand) {**

**return brandDao.addBrand(brand);**

**}**

Dao中代码

**//添加**

**public Integer addBrand(Brand brand);**

Dao.xml中代码

**<mapper namespace="cn.itcast.core.dao.product.BrandDao">**

**<!-- 添加 -->**

**<insert id="addBrand" parameterType="Brand">**

**INSERT INTO bbs\_brand**

**<trim prefix="(" suffix=")" >**

**id,**

**name,**

**description,**

**img\_url,**

**web\_site,**

**sort,**

**is\_display**

**</trim>**

**VALUES**

**<trim prefix="(" suffix=")" suffixOverrides=",">**

**#{id},**

**#{name},**

**#{description},**

**#{imgUrl},**

**#{webSite},**

**#{sort},**

**#{isDisplay}**

**</trim>**

**</insert>**

**</mapper>**

# 品牌全选

## 位置



**//全选**

**<script type="text/javascript">**

**//全选**

**function checkBox(name,checked){**

**//全选开始**

**$("input[name='" + name + "']").attr("checked",checked);**

**}**

**</script>**

**<form id="jvForm">**

**<!-- 列表头部 -->**

**<tr>**

**<td><input type="checkbox" onclick="checkBox('ids',this.checked)"/></td>**

**</tr>**

**<!-- 列表内容 -->**

**<tr>**

**<td><input type="checkbox" name="ids" value="${brand.id }"/></td>**

**</tr>**

**</form>**

# 品牌批量删除

## 位置



## 第一步:编写/back\_page/brand/list.jsp

**//添加onclick事件 方法名为:optDeletes();**

**<div style="margin-top:15px;"><input class="del-button" type="button" value="删除" onclick="optDeletes('${name}','${isDisplay }','${pageNo');"/></div>**

## 第二步:编写optDeletes()方法

**<script type="text/javascript">**

**//删除按钮**

**function optDeletes(name,isDisplay,pageNo) {**

**//判断checkbox是否至少有一个被选中**

**if($("input[name='ids']:checked").size()==0){**

**alert("请至少选择一个");**

**return;**

**}**

**//删除时需要确定**

**if(confirm("你确定删除吗? ")){**

**$("#jvForm").attr("action","deletes.do?name=" + name +"&isDisplay=" + isDisplay + "&pageNo=" + pageNo);**

**$("#jvForm").attr("method","post").submit();**

**}**

**}**

**</script>**

**<form id="jvForm" action="list.do" method="post">**

**</form>**

## 第三步:编写Sql语句

**delete from bbs\_brand where id in (?,?,…)**

## 第四步:编写BrandDao.xml

**<!-- 根据主键批量删除 -->**

**<delete id="deleteByKeys">**

**delete from bbs\_brand**

**where**

**id in**

**<foreach collection="array" separator="," item="id" open="(" close=")">**

**#{id}**

**</foreach>**

**</delete>**

## 第五步:编写BrandDao接口

**//批量删除**

**public void deleteByKeys(Integer[] ids);**

## 第六步:编写BrandService接口

**//批量删除**

**public void deleteByKeys(Integer[] ids);**

## 第七步:编写BrandServiceImpl实现类

**@Resource**

**private BrandDao brandDao;**

**public void deleteByKeys(Integer[] ids) {**

**return brandDao.deleteByKeys(ids);**

**}**

## 第八步:编写BrandController

**@Autowired**

**private BrandService brandService;**

**@RequestMapping(value = "/brand/deletes.do")**

**public String delete(Integer[] ids,String name,Integer isDisplay,ModelMap model){**

**//TODO 删除**

**brandService.deleteBrandByKeys(ids);**

**if(StringUtils.isNotBlank(name)){**

**model.addAttribute("name", name);**

**}**

**if(null != pageNo){**

**model.addAttribute("pageNo", pageNo);**

**}**

**if(null != isDisplay){**

**model.addAttribute("isDisplay", isDisplay);**

**}**

**return "redirect:/brand/list.do";**

**}**

# 品牌批量删除

## 位置



## 第一步:编写删除<a>标签

**<!-- a标签-->**

**<a class="pn-opt" onclick="optDelete('${name}','${isDisplay }','${pageNo }', '${brand.id}');" href="javascript:void(0)">删除</a>**

**//批量删除按钮**

**function optDelete(name,isDisplay,pageNo,brandId){**

**//**

**if($("input[name='ids']:checked").size() == 0){**

**alert("请至少选择一个");**

**return;**

**}**

**if(!confirm("你确定删除吗?")){**

**return;**

**}**

**//使用GET提交**

**window.location.href="/brand/delete.do?name=" + name + "&isDisplay=" + isDisplay + "&pageNo=" + pageNo + "&brandId=" + brandId;**

**}**

## 第二步:在BrandController中编写删除方法

**@Autowired**

**private BrandService brandService;**

**//删除品牌**

**@RequestMapping(value = "/brand/delete.do")**

**public String delete(Integer id,String name,Integer isDisplay,ModelMap model){**

**//批量删除**

**brandService.deleteBrands(id);**

**if(null != pageNo){**

**model.addAttribute("pageNo", pageNo);**

**}**

**if(null != name){**

**model.addAttribute("name", name);**

**}**

**if(null != isDisplay){**

**model.addAttribute("isDisplay", isDisplay);**

**}**

**//返回List页面**

**return "redirect:/brand/list.do";**

**}**

## 第三步:在BrandService中

**//删除品牌**

**public void deleteBrandByKeys (Integer ids);**

## 第四步:在BrandServiceImpl中

**@Autowired**

**private BrandDao brandDao;**

**//删除品牌**

**public void deleteBrandByKey (Integer ids){**

**return brandDao.deleteBrandByKeys(id);**

**}**

## 第五步:在BrandDao中

**//删除品牌**

**public void deleteBrandByKey (Integer id);**

## 第六步:在BrandDao.xml中

**<!-- 根据主键删除 -->**

**<delete id="deleteByKey" parameterType="hashMap">**

**DELETE FROM bbs\_brand**

**WHERE**

**id=#{id}**

**</delete>**

# 品牌单删（留给同学们练习）

# 品牌修改

## 位置



## 第一步:编写back\_page/brand/list.jsp

**//修改href**

**<a class="pn-opt" href="javascript:void(0)" onclick="window.location.href='/brand/toUpdate.do?id=${brand.id}'">修改</a>|**

## 第二步:编写Sql

**select id,name,description,img\_url,web\_site,sort,is\_display from bbs\_brand where id = ?**

## 第三步:编写BrandDao.xml

**<select id="getBrandByKey" parameterType="Integer" resultMap="brand">**

**select id,name,description,img\_url,web\_site,sort,is\_display from bbs\_brand where id = #{id}**

**</select>**

## 第四步:编写BrandDao接口

**public Brand getBrandByKey(Integer id);**

## 第五步:编写BrandService接口

**public Brand getBrandByKey(Integer id);**

## 第六步编写BrandServiceImpl实现类

**public Brand getBrandByKey(Integer id) {**

**// TODO Auto-generated method stub**

**return brandDao.getBrandByKey(id);**

**}**

## 第七步:编写BrandController

**//去修改页面**

**@RequestMapping(value = "/brand/toEdit.do")**

**public String toEdit(Integer id,ModelMap model){**

**model.addAttribute("brand", brandService.getBrandByKey(id));**

**return "brand/edit";**

**}**

## 第八步:编写back\_page/brand/edit.jsp

**<script type="text/javascript" scr="/res/common/js/jquery.form.js"></script>**

**<script type="text/javascript">**

**//采用jquery.form.js异步上传图片,并结合<form>表单**

**function uploadFic(){**

**var options = {**

**//请求路径**

**url = "/upload/uploadPic.do",**

**dataType = "json",**

**type = "post",**

**beforeSubmit : function(formData,jqForm,options){**

**//判断是否为图片**

**var f = jqForm[0];//将jqForm转成DOM对象**

**var v = f.logoPic.value;//获取DOM对象中name为logoPic的值**

**//获取扩展名,并转成小写**

**var ext = v.substring(v.length-3).toLowerCase();**

**//比对扩展名 jpg gif bmp png**

**if(ext != "jpg" && ext != "gif" && ext != "bmp" && ext != "png"){**

**alert("只允许上传图片!");**

**return false;**

**}**

**//校验提交的表单**

**return true;**

**},**

**success : function(data){**

**//处理结果**

**//将相对路径设置给隐藏域中,提交时用**

**$("#imgUrl").val(data.path);**

**//将全路径设置给img标签,显示图片用**

**#("#allImgUrl).attr("src",data.url);**

**}**

**}**

**$("#jvForm").ajaxSubmit(options);**

**}**

**</script>**

**<!-- 回显数据 action="edit.do" -->**

**<form id="jvForm" action="edit.do" method="post">**

**<!-- 隐藏Id -->**

**<input type="hidden" value="${brand.id }" name="id"/>**

**<table cellspacing="1" cellpadding="2" width="100%" border="0" class="pn-ftable">**

**<tbody>**

**<tr>**

**<td width="20%" class="pn-flabel pn-flabel-h">**

**<span class="pn-frequired">\*</span>**

**品牌名称:</td><td width="80%" class="pn-fcontent">**

**<input type="text" class="required" name="name" value="${brand.name }" maxlength="100"/>**

**</td>**

**</tr>**

**<tr>**

**<td width="20%" class="pn-flabel pn-flabel-h">**

**<span class="pn-frequired">\*</span>**

**上传商品图片(90x150尺寸):</td>**

**<td width="80%" class="pn-fcontent">**

**注:该尺寸图片必须为90x150。**

**</td>**

**</tr>**

**<tr>**

**<td width="20%" class="pn-flabel pn-flabel-h"></td>**

**<td width="80%" class="pn-fcontent">**

**<img width="100" height="100" id="allImgUrl" src="${brand.allImgUrl }"/>**

**<input type="file" />**

**<input type="hidden" name="imgUrl" id="imgUrl" value="${brand.imgUrl }"/>**

**</td>**

**</tr>**

**<tr>**

**<td width="20%" class="pn-flabel pn-flabel-h">**

**品牌描述:</td><td width="80%" class="pn-fcontent">**

**<input type="text" class="required" name="description" value="${brand.description }" maxlength="80" size="60"/>**

**</td>**

**</tr>**

**<tr>**

**<td width="20%" class="pn-flabel pn-flabel-h">**

**排序:</td><td width="80%" class="pn-fcontent">**

**<input type="text" class="required" name="sort" value="${brand.sort }" maxlength="80"/>**

**</td>**

**</tr>**

**<tr>**

**<td width="20%" class="pn-flabel pn-flabel-h">**

**是否可用:</td><td width="80%" class="pn-fcontent">**

**<input type="radio" name="isDisplay" value="1" <c:if test="${brand.isDisplay == 1 }">checked="checked"</c:if>/>可用**

**<input type="radio" name="isDisplay" value="0" <c:if test="${brand.isDisplay == 0 }">checked="checked"</c:if>/>不可用**

**</td>**

**</tr>**

**</tbody>**

**<tbody>**

**<tr>**

**<td class="pn-fbutton" colspan="2">**

**<input type="submit" class="submit" value="提交"/> &nbsp; <input type="reset" class="reset" value="重置"/>**

**</td>**

**</tr>**

**</tbody>**

**</table>**

**</form>**

## 第九步:提交

## 第十步:编写BrandDao.xml

**<update id="updateBrandByKey" parameterType="Brand">**

**update bbs\_brand**

**set**

**<trim suffixOverrides=",">**

**<if test="name != null">**

**name = #{name} ,**

**</if>**

**<if test="description != null">**

**description = #{description} ,**

**</if>**

**<if test="imgUrl != null">**

**img\_url = #{imgUrl} ,**

**</if>**

**<if test="sort != null">**

**sort = #{sort} ,**

**</if>**

**<if test="isDisplay != null">**

**is\_display = #{isDisplay}**

**</if>**

**</trim>**

**<where>**

**id = #{id}**

**</where>**

**</update>**

## 第十一步:编写BrandDao

**public void updateBrandByKey(Brand brand);**

## 第十二步:BrandService接口

**public void updateBrandByKey(Brand brand);**

## 第十三步:BrandServiceImpl实现类

**@Override**

**public void updateBrandByKey(Brand brand) {**

**brandDao.updateBrandByKey(brand);**

**}**

## 第十四步:BrandController

**@Autowired**

**private BrandService brandService;**

**//修改**

**@RequestMapping(value = "/brand/edit.do")**

**public String edit(Brand brand,ModelMap model){**

**brandService.updateBrandByKey(brand);**

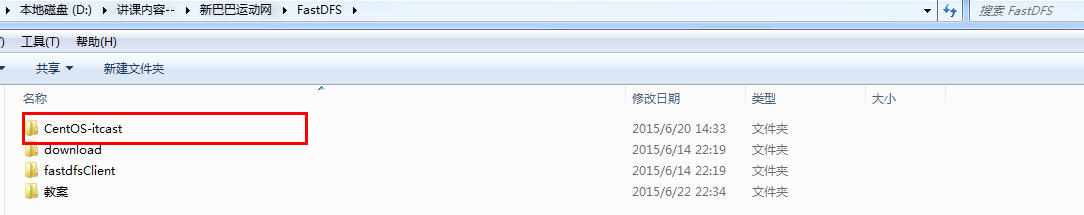
**return "redirect:/brand/list.do";**

**}**

# 图片服务器FastDFS（面试经常问）优化技术之一

## 资料位置：

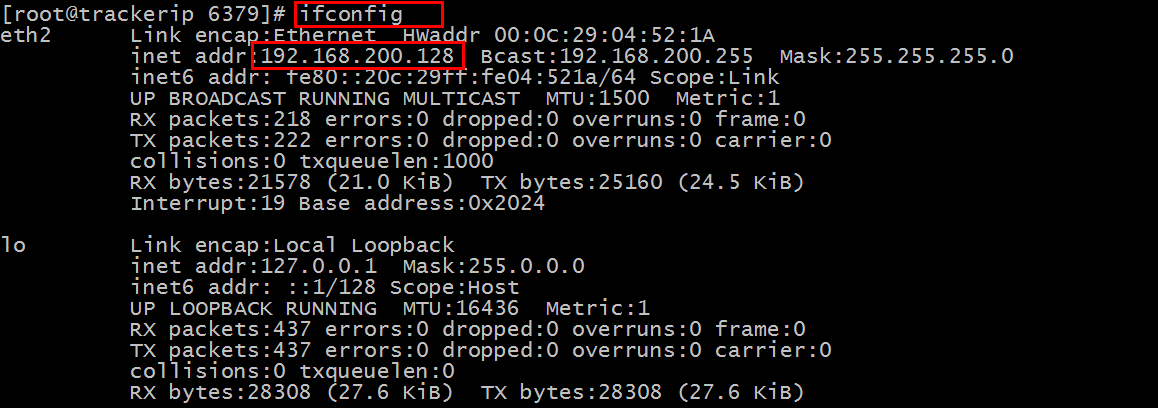
D:\讲课内容--\新巴巴运动网\FastDFS



原理可查看文档

## 前提：已经发放已经搭建好的FastDFS服务器Centos01到同学们手中

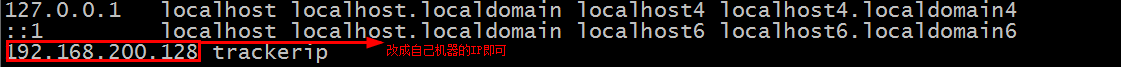
* 查看自己的机器的IP



* 开机后只需要配置hosts文件中的IP即可



* 打开之后



## FastDFS的Java接口工具类

**/\*\***

**\* 上传文件到FastDFS文件服务器**

**\* @author lx**

**\***

**\*/**

**public class FastDFSUtil {**

**public static String uploadFileToFastDFS(MultipartFile file){**

**//FastDFS 连接 服务器**

**//返回Tracker服务**

**TrackerServer trackerServer = null;**

**//返回的路径**

**String fileId = null;**

**try {**

**//读取配置文件**

**ClassPathResource classPathResource = new ClassPathResource("fdfs\_client.conf");**

**//全局配置文件加载**

**ClientGlobal.init(classPathResource.getClassLoader().getResource("fdfs\_client.conf").getPath());**

**//创建Tracker客户端**

**TrackerClient trackerClient = new TrackerClient();**

**//返回Tracker服务**

**trackerServer = trackerClient.getConnection();**

**StorageClient1 storageClient = new StorageClient1(trackerServer, null);**

**NameValuePair[] meta\_list = new NameValuePair[3];**

**String ext = FilenameUtils.getExtension(file.getOriginalFilename());**

**meta\_list[0] = new NameValuePair("fileName", file.getOriginalFilename());**

**meta\_list[1] = new NameValuePair("fileExtName", ext);**

**meta\_list[2] = new NameValuePair("fileLength", String.valueOf(file.getSize()));**

**fileId = storageClient.upload\_file1(file.getBytes(), ext, meta\_list);**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}finally{**

**if(null != trackerServer){**

**try {**

**trackerServer.close();**

**} catch (IOException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**}**

**return fileId;**

**}**

**}**

## FastDFS的连接服务器的配置文件

在classpath下创建fdfs\_client.conf文件

只需修改下面配置信息中红色部分的IP与自己的FastDFS服务器IP一致即可

**# 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**

## FastDFS在项目中的应用

将babasport-back项目中cn.itcast.core.controller.UploadController类中uploadPic方法修改为如下内容

**//上传图片 文件**

**@RequestMapping(value = "/upload/uploadPic.do")**

**public void uploadPic(@RequestParam MultipartFile pic,HttpServletResponse response,HttpServletRequest request) throws IOException{**

**//上传到FastDFS 文件系统上去**

**String path = FastDFSUtil.uploadFileToFastDFS(pic);**

**//返回 path url**

**JSONObject jo = new JSONObject();**

**jo.put("path", path);**

**//返回的是全路径 Url 常量里写的服务器的域名**

**jo.put("url", Constants.IMG\_WEB + path);**

**//JSON**

**response.setContentType("application/json;charset=UTF-8");**

**response.getWriter().write(jo.toString());**

**}**

将babasport-back项目中cn.itcast.core.web.Constants中IMG\_WEB修改为如下内容

**//图片服务器**

**public static final String IMG\_WEB = "http://192.168.200.128/";**

# 数据库设计工具PowerDesigner的使用

## 逆向工程来完成Mysql数据库表结构生成物理模型

参考:<<安装powerDesigner15反向工程过程.docx>>

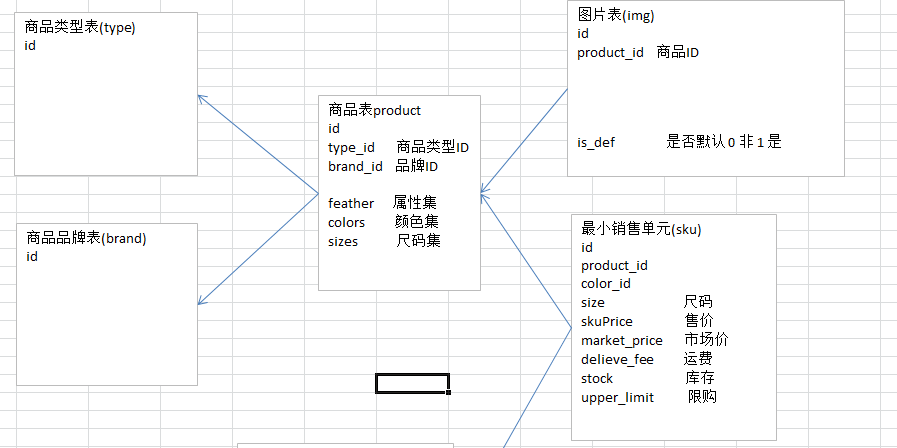
## 物理模型Common(注释字段)复制到Name字段,方便查看表字段

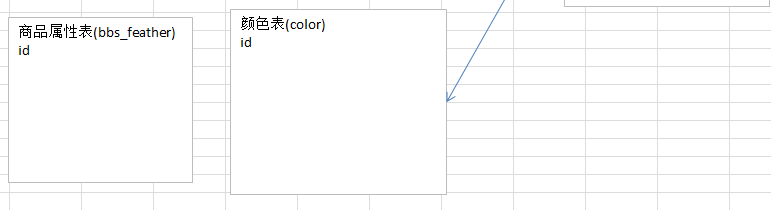
参考:<< PowerDesigner 把Comment写到name中 和把name写到Comment中 pd7以后版本可用.docx>>

## 制作模板,生成Html或Word来查看表字段及关联关系,与团队分享数据结构或给领导或客户方来查看

参考:<< powerDesigner生成Html及Word.docx>>

# 讲解表的作用及关联关系,讲解主要字段





# Mybatis逆向工程使用

## 代码所在位置

\新巴巴运动网\Dao--bean--mapper-代码生成器\generatorSqlmap-increase

此为完整版的逆向工程

使用之前更改generatorConfig.xml中的数据库名及密码

片段如下

<!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->

<jdbcConnection driverClass="com.mysql.jdbc.Driver"

connectionURL="jdbc:mysql://localhost:3306/babasport" userId="root"

password="root">

</jdbcConnection>

## 此逆向工程比Mybatis提供的工程优化内容如下

问题:

1)指定字段 id name (解决)

2)Limit 物理分页 (解决) 目前只支持Mysql

3)注释不好 (解决)

4) POJO 序列化 接口 (解决)

5) toString (解决)

6)条件对象名称希望是Query (解决)

7)生成模块包名(解决)

8)数据表名去前缀 成为 POJO类名(解决)

9)数据库表字段与POJO之间的类型转换(解决)

10)生成Mapper名改成习惯的Dao名

# 商品列表

## 说明:

1:初始页面加载下架商品

2:上下架条件处,要求默认是下架

3:条件处需要加载所有的可显示品牌

4:列表要求有分页 分页中要求显示总页数.并可指定页号进行查询

5:要求分页查看时必须携带条件

6:每次请求要求必须检查页号,如果页号为Null或小于0(包括0)必须默认为第1页

7:每次查询,条件如果更新,必须将页号显示为第1页

8:每次页面刷新,条件都要被选中

9:分页最好封装起来,便于以后复用

10:列表页面显示商品信息有限,要求点查看按钮可查看此商品的所有信息

## 第一步:创建ProductController类

**/\*\***

**\* 商品控制层 后台 商品管理**

**\* 商品列表**

**\* 上架**

**\* 查看**

**\* 添加**

**\* @author lx**

**\***

**\*/**

**@Controller**

**public class ProductController {**

**@Autowired**

**private ProductService productService;**

**@Autowired**

**private BrandService brandSerice;**

**//商品列表**

**@RequestMapping("/product/list.do")**

**public String list(Integer pageNo,String name , Integer brandId,Boolean isShow,Model model){**

**BrandQuery brandQuery = new BrandQuery();**

**//指定字段 id name**

**brandQuery.setFields("id,name");**

**//可见**

**brandQuery.setIsDisplay(1);**

**//加载页面所需要的品牌 是可见的**

**List<Brand> brands = brandSerice.selectBrandList(brandQuery);**

**//参数回显**

**model.addAttribute("brands", brands);**

**//查询商品 实例化 商品Query对象**

**ProductQuery productQuery = new ProductQuery();**

**//获取内部类**

**Criteria criteria = productQuery.createCriteria();**

**//分页拼接条件的**

**StringBuilder params = new StringBuilder();**

**//判断请求的参数是否为NUll " " 商品名**

**if(null != name){**

**criteria.andNameLike("%" + name + "%");**

**params.append("name=").append(name);**

**//参数回显**

**model.addAttribute("name", name);**

**}**

**//品牌ID where name = "" or ( brandId = 1 and is\_show = 1)**

**if(null != brandId){**

**criteria.andBrandIdEqualTo(brandId);**

**params.append("&").append("brandId=").append(brandId);**

**//参数回显**

**model.addAttribute("brandId", brandId);**

**}**

**//是否上下架 默认是下架 Boolean 布尔类型**

**if(null != isShow){**

**criteria.andIsShowEqualTo(isShow);**

**params.append("&").append("isShow=").append(isShow);**

**//参数回显**

**model.addAttribute("isShow", isShow);**

**}else{**

**criteria.andIsShowEqualTo(false);**

**params.append("&").append("isShow=").append(false);**

**//参数回显**

**model.addAttribute("isShow", false);**

**}**

**//查询3条**

**productQuery.setPageSize(3);**

**//当前页**

**productQuery.setPageNo(Pagination.cpn(pageNo));**

**//**

**productQuery.setOrderByClause("id desc");**

**//查询分页数据**

**//分页 = productService.select...(productQuery);**

**Pagination pagination = productService.selectByExample(productQuery);**

**//**

**String url = "/product/list.do";///product/list.do?name=''&brandId=1&isShow=1&pageNo=2**

**//分页展示**

**pagination.pageView(url, params.toString());**

**//回显**

**//参数回显**

**model.addAttribute("pagination", pagination);**

**return "product/list";**

**}**

## 第二步:修改ProductServiceImpl实现类

**/\*\***

**\* 商品**

**\* @author lx**

**\***

**\*/**

**@Service**

**@Transactional**

**public class ProductServiceImpl implements ProductService {**

**@Autowired**

**private ProductMapper productMapper;**

**@Autowired**

**private ImgMapper imgMapper;**

**@Autowired**

**private SkuMapper skuMapper;**

**@Override**

**public Pagination selectByExample(ProductQuery example) {**

**// TODO Auto-generated method stub**

**//创建 分页对象**

**Pagination pagination = new Pagination(**

**example.getPageNo(),example.getPageSize(),productMapper.countByExample(example));**

**List<Product> products = productMapper.selectByExample(example);**

**for (Product product : products) {**

**ImgQuery imgQuery = new ImgQuery();**

**//商品ID**

**Criteria imgQ = imgQuery.createCriteria();**

**imgQ.andProductIdEqualTo(product.getId());**

**imgQ.andIsDefEqualTo(true);**

**//是默认图片**

**//加载图片 只有一张图**

**List<Img> imgs = imgMapper.selectByExample(imgQuery);**

**product.setImg(imgs.get(0));**

**}**

**pagination.setList(products);**

**return pagination;**

**}**

## 第三步:back\_page/product/list.jsp

### 条件查询部分

如果名称条件出现中文乱码

在tomcat/conf/server.xml 的第70行,输入以下内容

**<Connector port="8080" protocol="HTTP/1.1"**

**connectionTimeout="20000"**

**redirectPort="8443" URIEncoding="UTF-8" />**



红色部分代码

**<form action="/product/list.do" method="post" style="padding-top:5px;">**

**名称: <input type="text" value="${name }" name="name"/>**

**<select name="brandId">**

**<option value="">请选择品牌</option>**

**<c:forEach items="${brands }" var="brand">**

**<option value="${brand.id }" <c:if test="${brandId == brand.id }">selected="selected"</c:if>>${brand.name }</option>**

**</c:forEach>**

**</select>**

**<select name="isShow">**

**<option value="1" <c:if test="${isShow == 1 }">selected="selected"</c:if>>上架</option>**

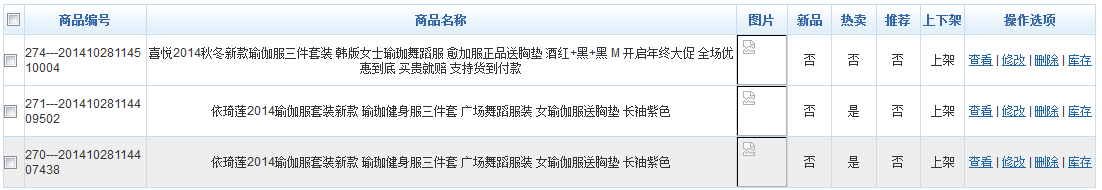
**<option value="0" <c:if test="${isShow == 0 }">selected="selected"</c:if>>下架</option>**

**</select>**

**<input type="submit" class="query" value="查询"/>**

**</form>**

### 内容部分



代码

**<table cellspacing="1" cellpadding="0" width="100%" border="0" class="pn-ltable">**

**<thead class="pn-lthead">**

**<tr>**

**<th width="20"><input type="checkbox" onclick="Pn.checkbox('ids',this.checked)"/></th>**

**<th>商品编号</th>**

**<th>商品名称</th>**

**<th>图片</th>**

**<th width="4%">新品</th>**

**<th width="4%">热卖</th>**

**<th width="4%">推荐</th>**

**<th width="4%">上下架</th>**

**<th width="12%">操作选项</th>**

**</tr>**

**</thead>**

**<tbody class="pn-ltbody">**

**<c:forEach items="${pagination.list }" var="entry">**

**<tr bgcolor="#ffffff" onmouseover="this.bgColor='#eeeeee'" onmouseout="this.bgColor='#ffffff'">**

**<td><input type="checkbox" name="ids" value="${entry.id }"/></td>**

**<td>${entry.id }---${entry.no }</td>**

**<td align="center">${entry.name }</td>**

**<td align="center"><img width="50" height="50" src="${entry.img.allUrl }"/></td>**

**<td align="center"><c:if test="${entry.isNew ==1}">是</c:if><c:if test="${entry.isNew ==0}">否</c:if></td>**

**<td align="center"><c:if test="${entry.isHot ==1}">是</c:if><c:if test="${entry.isHot ==0}">否</c:if></td>**

**<td align="center"><c:if test="${entry.isCommend ==1}">是</c:if><c:if test="${entry.isCommend ==0}">否</c:if></td>**

**<td align="center"><c:if test="${entry.isShow ==1}">上架</c:if><c:if test="${entry.isShow ==0}">下架</c:if></td>**

**<td align="center">**

**<a href="/product/detail.shtml?id=${entry.id }" target="\_black" class="pn-opt">查看</a> | <a href="#" class="pn-opt">修改</a> | <a href="#" onclick="if(!confirm('您确定删除吗？')) {return false;}" class="pn-opt">删除</a> | <a href="/sku/list.do?id=${entry.id }" class="pn-opt">库存</a>**

**</td>**

**</tr>**

**</c:forEach>**

**</tbody>**

**</table>**

### 分页部分



代码

**<div class="page pb15">**

**<span class="r inb\_a page\_b">**

**<c:forEach items="${pagination.pageView }" var="page">**

**${page }**

**</c:forEach>**

**</span>**

**</div>**

### 上下架



代码

**<div style="margin-top:15px;">**

**<input class="del-button" type="button" value="删除" onclick="optDelete();"/>**

**<input class="add" type="button" value="上架" onclick="show();"/>**

**<input class="del-button" type="button" value="下架" onclick="hide();"/>**

**</div>**

# 商品管理之添加页面

## 第一步:toAdd到添加页面

## 第二步:需要加载的数据有

1. 商品类型
2. 商品品牌
3. 材质
4. 颜色
5. 尺码(这里尺码少,可不加载)

**//去添加页面**

**@RequestMapping("/product/toAdd.do")**

**public String toAdd(Model model){**

**//类型**

**TypeQuery example = new TypeQuery();**

**cn.itcast.core.bean.product.TypeQuery.Criteria criteria = example.createCriteria();**

**criteria.andParentIdNotEqualTo(0);**

**List<Type> types = typeSerivce.selectType(example);**

**model.addAttribute("types", types);**

**//品牌**

**BrandQuery brandQuery = new BrandQuery();**

**//指定字段 id name**

**brandQuery.setFields("id,name");**

**//可见**

**brandQuery.setIsDisplay(1);**

**//加载页面所需要的品牌 是可见的**

**List<Brand> brands = brandSerice.selectBrandList(brandQuery);**

**model.addAttribute("brands", brands);**

**//材质\featureSeri**

**List<Feature> features = featureSerivce.selectFeatureByExample(null);**

**model.addAttribute("features", features);**

**//颜色**

**ColorQuery e = new ColorQuery();**

**//父ID不等于0**

**cn.itcast.core.bean.product.ColorQuery.Criteria createCriteria = e.createCriteria();**

**createCriteria.andParentIdNotEqualTo(0);**

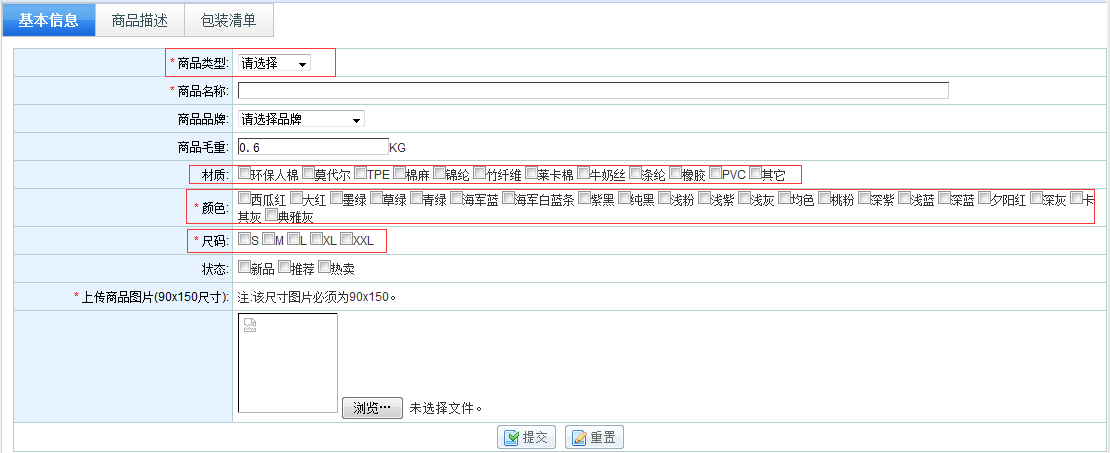
**List<Color> colors = colorSerivce.selectColorByExample(e);**

**model.addAttribute("colors", colors);**

**return "product/add";**

**}**

## 第三步:查看加载结果页面



## 第四步:切换Tab页



**<script type="text/javascript">**

**$(function(){**

**//暂定**

**var tObj;**

**//id为tabs 下的所有a标签 遍历**

**$("#tabs a").each(function(){**

**if($(this).attr("class").indexOf("here") == 0){tObj = $(this)}**

**$(this).click(function(){**

**var c = $(this).attr("class");**

**if(c.indexOf("here") == 0){return;}**

**//#tab\_2 3**

**var ref = $(this).attr("ref");**

**//#tab\_1**

**var ref\_t = tObj.attr("ref");**

**tObj.attr("class","nor");**

**$(this).attr("class","here");**

**$(ref\_t).hide();//隐藏**

**$(ref).show();//显示**

**tObj = $(this);**

**//加载Fck**

**if(ref == '#tab\_2'){**

**//加载Fck productdesc**

**var fck = new FCKeditor("productdesc");**

**fck.BasePath = "/res/fckeditor/";**

**fck.Height = "400";**

**fck.Config["ImageUploadURL"] = "/fck/uploadPic.do";**

**fck.ReplaceTextarea();**

**}**

**});**

**});**

**});**

**</script>**

**<h2 class="h2\_ch">**

**<span id="tabs">**

**<a href="javascript:void(0);" ref="#tab\_1" title="基本信息" class="here">基本信息</a>**

**<a href="javascript:void(0);" ref="#tab\_2" title="商品描述" class="nor">商品描述</a>**

**<a href="javascript:void(3);" ref="#tab\_3" title="商品参数" class="nor">包装清单</a>**

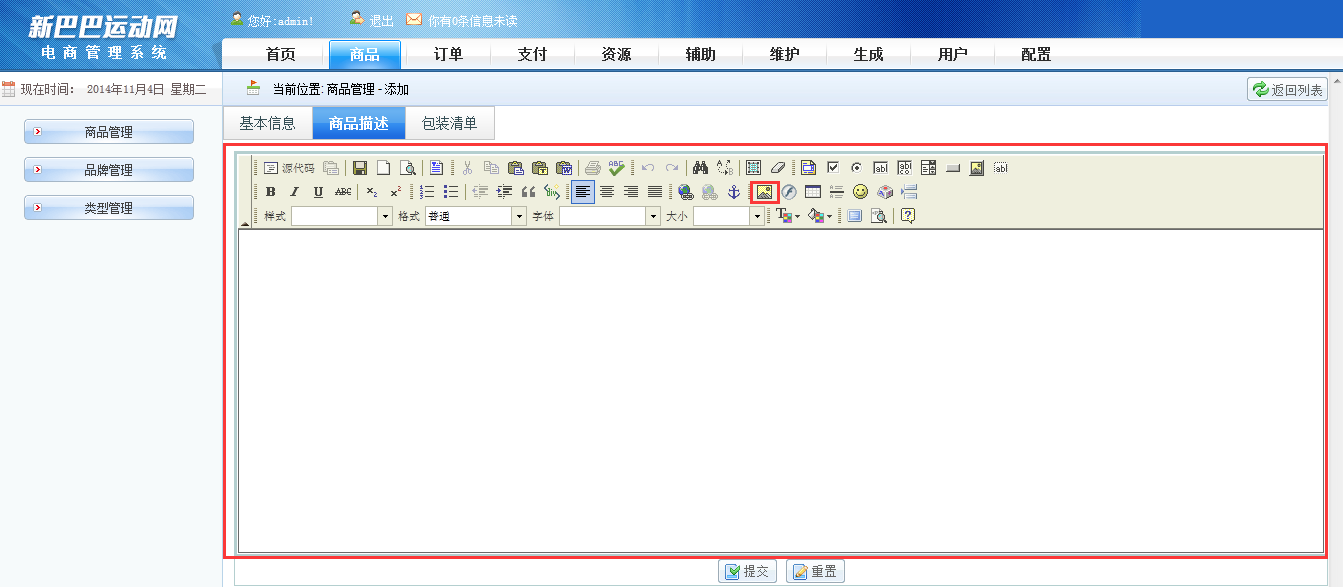
**</span>**

**</h2>**

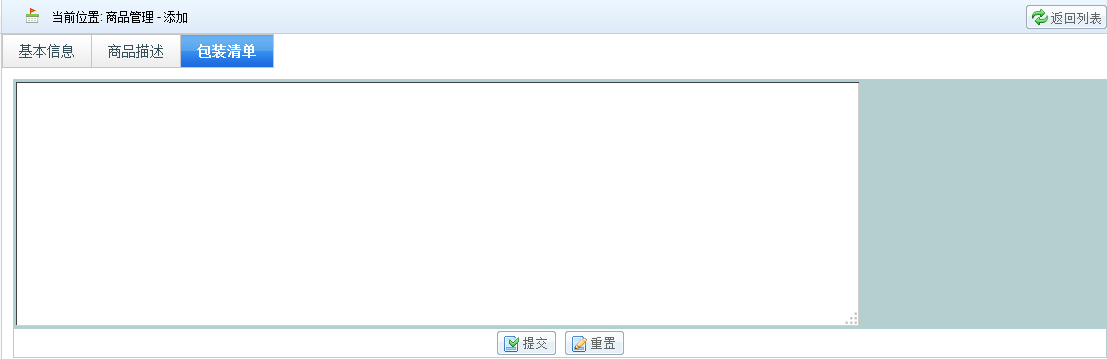
### 基础信息



### 商品描述



### 包装清单



# 富文本FCK

## 第一步:引入fckeditor.js文件

## 第二步:前端Js代码片段:

var fck = new FCKeditor("productdesc");//productdesc是textarea的ID

fck.BasePath = "/res/fckeditor/";//基本路径webapp/res/fckeditor/.

fck.Height = 400 ;

fck.Config["ImageUploadURL"] = "/product/uploadFck.do";//上传图片

//fck.ToolbarSet = "myTool";//自定义工具栏

fck.ReplaceTextarea();//转成Fck

## 第三步:后端Java代码片段:接收Fck图片到指定的图片服务器,由于Fck上传的图片没有fileName,无法使用注解的方式获取到上传的图片,故采用下面这段代码来获取

//强制转换request

MultipartHttpServletRequest mr = (MultipartHttpServletRequest) request;

//获得文件的集合

Map<String, MultipartFile> map = mr.getFileMap();

//获得map集合的key

Set<String> keys = map.keySet();

//获得唯一的一个key

String key = keys.iterator().next();

//获得文件

MultipartFile mf = map.get(key);

## 第四步:以上代码书写前提是在springmvc-back.xml中已经配置了如下代码

<!-- 上传图片转换器 -->

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<!-- 设置上传文件的最大尺寸为1MB -->

<property name="maxUploadSize" value="1048576"/>

</bean>

## 第五步:图片名称生成策略

DateFormat df = new SimpleDateFormat("yyyyMMddHHmmssSSS");

String picName = df.format(new Date());

Random r = new Random();

for(int i=0;i<3;i++){

picName += r.nextInt(10);

}

## 第六步:获取扩展名

String ext = FilenameUtils.getExtension(multipartFile.getOriginalFilename());

## 第七步:再采用day02中学习的Jersey将图片发送到另一个服务器上去即可

String path = "upload/" + picName + "." + ext;

String url = Constants.IMAGE\_URL + path;

//IMAGE\_URL = "http://localhost:8088/image-web/";

Client client = new Client();

WebResource resource = client.resource(url);

try {

resource.put(String.class, multipartFile.getBytes());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

## 第八步:将图片的Url返回给Fck

UploadResponse ok = UploadResponse.getOK(url);

try {

response.getWriter().print(ok);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

# 商品信息保存

## 第一步:提交表单后,需要在后台生成的数据有

* 商品:编号,添加时间
* 最小销售单元:市场价,售价,库存,运费,销量,购买限制,增加时间,状态(0:历史,1:最新),类型(0:赠品,1:普通)

## 第三步:在ProductController中

//商品添加

@RequestMapping(value = "/product/add.do")

public String add(Product product){

//保存商品 因为要控制事务 所以在商品的Service层一起保存

productService.addProduct(product);

return "redirect:/product/list.do";

}

## 第四步:在ProductServiceImpl实现类

//添加

public void addProduct(Product product){

//保存商品

//商品时间 当前时间

product.setCreateTime(new Date());

//默认 下架

product.setIsShow(false);

// 删除 true

product.setIsDel(true);

//保存

productDao.insertSelective(product);

//图片

Img img = product.getImg();

img.setProductId(product.getId());

img.setIsDef(true);

imgDao.insert(img);

//保存Sku

String[] colors = product.getColors().split(",");

for (String color : colors) {

String[] sizes = product.getSizes().split(",");

for (String size : sizes) {

//创建

Sku sku = new Sku();

//商品ID

sku.setProductId(product.getId());

//颜色ID

sku.setColorId(Long.parseLong(color));

//尺码

sku.setSize(size);

//运费

sku.setDeliveFee(10f);

//售价

sku.setPrice(0f);

//库存

sku.setStock(0);

//市场价

sku.setMarketPrice(0f);

//时间

sku.setCreateTime(new Date());

//普通

sku.setSkuType(true);

//保存

skuDao.insertSelective(sku);

}

}

}

# 商品保存优化之Redis应用

## 需求:

商品编号,订单号,采用Redis缓存服务器来生成,生成策略可采用从100开始每添加一款商品,商品编号加1的方式来生成,

例如:100+1=101 redis自带自增命令 ,可直接使用

Key Value : pno 101

Redis安装可参考: 参考：D:\讲课内容--\新巴巴运动网\redis缓存

## Redis加入项目中

第一步:加入二个Jar包 ,将二个包放在pom.xml最下方

<dependency>

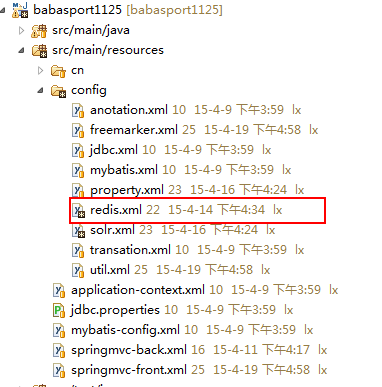
<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.7.0</version>

</dependency>

## 位置



第二步:在classpath:config/redis.xml中

<!-- 配置Redis服务器连接 -->

<bean id="jedis" class="redis.clients.jedis.Jedis">

<constructor-arg>

<value>192.168.200.149</value>

</constructor-arg>

</bean>

第三步:java接口使用

/\*\*

\* 测试Redis jedis

\* @author lx

\*

\*/

public class TestJedis {

@Test

public void testJedis() throws Exception {

Jedis jedis = new Jedis("192.168.200.149");

jedis.set("foo", "hehe");

//取值

jedis.get("foo");

//商品编号 增1

jedis.incr("pno");

//商品编号 减1

jedis.decr("foo");

//指定增多少

redis.incrBy("pno", 3);

//指定减多少

redis.incrBy("pno", 3);

//更多命令可参考Jedis的 Api

}

}

## Redis在项目中的应用之商品编号的生成

@Autowired

private Jedis jedis;

//添加

public void addProduct(Product product){

//保存商品

//商品编号 生成 Redis

// int 20亿 （14亿

Long pno = jedis.incr("pno");

product.setId(pno);

//商品时间 当前时间

product.setCreateTime(new Date());

//默认 下架

product.setIsShow(false);

// 删除 true

product.setIsDel(true);

//保存

productDao.insertSelective(product);

//图片

Img img = product.getImg();

img.setProductId(product.getId());

img.setIsDef(true);

imgDao.insert(img);

//保存Sku

String[] colors = product.getColors().split(",");

for (String color : colors) {

String[] sizes = product.getSizes().split(",");

for (String size : sizes) {

//创建

Sku sku = new Sku();

//商品ID

sku.setProductId(product.getId());

//颜色ID

sku.setColorId(Long.parseLong(color));

//尺码

sku.setSize(size);

//运费

sku.setDeliveFee(10f);

//售价

sku.setPrice(0f);

//库存

sku.setStock(0);

//市场价

sku.setMarketPrice(0f);

//时间

sku.setCreateTime(new Date());

//普通

sku.setSkuType(true);

//保存

skuDao.insertSelective(sku);

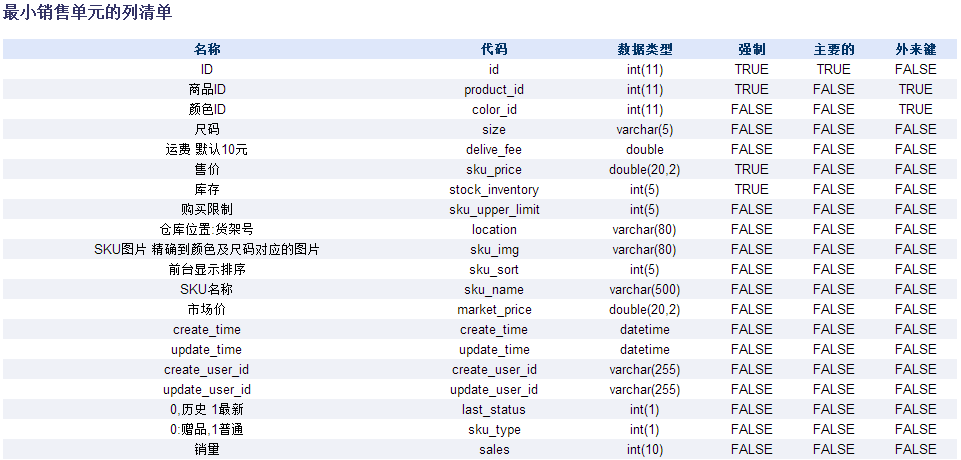
}

}

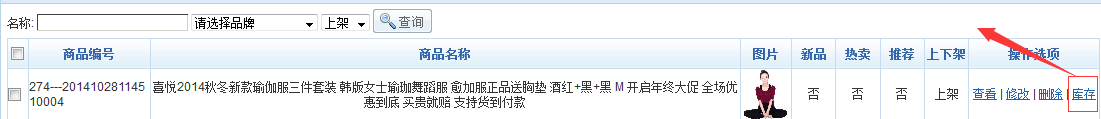
}

# 库存管理

## 第一步:查看bbs\_sku表字段



## 第二步:从商品列表页面跳转至库存管理页面



页面代码

<!-- 库存管理请求路径 -->

<a href="/sku/list.do?id=${entry.id }&no=${entry.no}" class="pn-opt">库存</a>

## 第三步:创建SkuController,并书写跳转库存管理,加载此商品的Sku数据

/\*\*

\* 库存管理

\* @author lx

\*

\*/

@Controller

public class SkuController {

@Autowired

private SkuService skuService;

//跳转库存管理页面

@RequestMapping(value = "/sku/list.do")

public String list(Integer id , String no,ModelMap model){

//创建Sku条件对象

SkuQuery skuQuery = new SkuQuery();

//设置商品ID

skuQuery.setProductId(id);

//返回结果集

List<Sku> skus = skuService.getSkuList(skuQuery);

model.addAttribute("skus", skus);

//返回商品编号

model.addAttribute("pno", no);

return "sku/list";

}

## 第四步:查看库存管理页面back\_page/sku/list.jsp



### 数据遍历部分代码

<c:forEach items="${skus}" var="sku">

<tr bgcolor="#ffffff" onmouseout="this.bgColor='#ffffff'" onmouseover="this.bgColor='#eeeeee'">

<td><input type="checkbox" value="73" name="ids"/></td>

<td>${pno }</td>

<td align="center">${sku.color.name }</td>

<td align="center">${sku.size }</td>

<td align="center"><input type="text" size="10" disabled="disabled" value="${sku.marketPrice }" id="m${sku.id }"/></td>

<td align="center"><input type="text" size="10" disabled="disabled" value="${sku.skuPrice }" id="p${sku.id }"/></td>

<td align="center"><input type="text" size="10" disabled="disabled" value="${sku.stockInventory }" id="i${sku.id }"/></td>

<td align="center"><input type="text" size="10" disabled="disabled" value="${sku.skuUpperLimit }" id="l${sku.id }"/></td>

<td align="center"><input type="text" size="10" disabled="disabled" value="${sku.deliveFee }" id="f${sku.id }"/></td>

<td align="center"><c:if test="${sku.skuType == 1}" >不是</c:if><c:if test="${sku.skuType == 0}" >是</c:if></td>

<td align="center"><a class="pn-opt" href="javascript:updateSku(${sku.id })">修改</a> | <a class="pn-opt" href="javascript:addSku(${sku.id })">保存</a></td>

</tr>

</c:forEach>

## 第五步:操作中的:修改按钮,保存按钮

### 编写sku/list.jsp jQuery脚本

<script type="text/javascript">

//修改按钮

function updateSku(skuId){

$("#m" + skuId).attr("disabled",false);

$("#p" + skuId).attr("disabled",false);

$("#i" + skuId).attr("disabled",false);

$("#l" + skuId).attr("disabled",false);

$("#f" + skuId).attr("disabled",false);

}

//保存按钮

function addSku(skuId){

var m = $("#m" + skuId).val();

var p = $("#p" + skuId).val();

var i = $("#i" + skuId).val();

var l = $("#l" + skuId).val();

var f = $("#f" + skuId).val();

//var ex = /^\d+$/; 整数

//var ex =/^[0-9]+([.]{1}[0-9]{1,2})?$/;小数点后两位

//ex.test(value)

var url = "/sku/add.do";

var params = {"id" : skuId, "marketPrice" : m, "skuPrice" : p, "stockInventory" : i, "skuUpperLimit" : l,"deliveFee" : f};

$.post(url,params,function(data){

//保存成功!

alert(data.message);

},"json");

$("#m" + skuId).attr("disabled",true);

$("#p" + skuId).attr("disabled",true);

$("#i" + skuId).attr("disabled",true);

$("#l" + skuId).attr("disabled",true);

$("#f" + skuId).attr("disabled",true);

}

</script>

## 第六步:编写SkuController

//保存Sku数据 保存按钮为更新操作

@RequestMapping(value = "/sku/add.do")

public void add(Sku sku,HttpServletResponse response){

skuService.updateSkuByKey(sku);

JSONObject jo = new JSONObject();

jo.put("message", "保存成功!");

ResponseUtils.renderJson(response, jo.toString());

}

product.setIsDel(true);

//保存商品 返回ID

productMapper.insert(product);

//保存图片

Img img = product.getImg();

//商品ID

img.setProductId(product.getId());

//默认

img.setIsDef(true);

imgMapper.insert(img);

//保存SKu

Sku sku = null;

for (String color : product.getColor().split(",")) {

//实例化

sku = new Sku();

//商品ID

sku.setProductId(product.getId());

//颜色ID

sku.setColorId(Integer.parseInt(color));

for(String size : product.getSize().split(",")){

sku.setSize(size);

//价格

sku.setSkuPrice(0.00);

//运费 默认10元

sku.setDeliveFee(10.00);

//库存

sku.setStockInventory(0);

//默认200件

sku.setSkuUpperLimit(200);

//市场价

sku.setMarketPrice(0.00);

//时间

sku.setCreateTime(new Date());

//

sku.setLastStatus(1);

//

sku.setSkuType(1);

sku.setSales(0);

//保存

skuMapper.insert(sku);

}

}

}

# 搭建Solr服务器

准备:

一台Centos6.4

关闭防火墙

service iptables stop

chkconfig iptables off

安装Jdk1.7并配置JDK的环境变量

第一步:上传solr-4.10.3.tar, apache-tomcat-7.0.47.tar.gz

第二步:mkdir /solr

第三步:tar –zxvf solr-4.10.3.tar –C /solr

第四步:tar –zxvf apache-tomcat-7.0.47.tar.gz –C /solr

第五步:复制solr的ext下的包

第六步:复制solr.war到tomcat/webapp/下

第七步:cd /solr/ apache-tomcat-7.0.47webapp

1. mkdir solr
2. unzip solr.war –d solr

第八步:配置solr/WEB-INF/web.xml 指定solr的目录路径

第九步:修改web.xml第40行 去掉注释

<!--

<env-entry>

<env-entry-name>solr/home</env-entry-name>

<env-entry-value>/put/your/solr/home/here</env-entry-value>

<env-entry-type>java.lang.String</env-entry-type>

</env-entry>

-->

修改为

<env-entry>

<env-entry-name>solr/home</env-entry-name>

<env-entry-value>/solr/solr-4.10.3/example/solr</env-entry-value>

<env-entry-type>java.lang.String</env-entry-type>

</env-entry>

第十步:启动Tomcat.查看日志

# 配置IK分词器

## 把IKAnalyzer2012FF\_hf1.zip上传到服务器

## 解压到指定目录

1. mkdir /solr/IK
2. unzip IKAnalyzer2012FF\_hf1.zip -d /solr/IK

## 复制IK到项目中

1. 复制IK.cfg.xml 到项目的classpath下
2. 复制IK的Jar包到/webapp/solr/WEB\_INF/lib下
3. 复制stop.dic停词到项目的classpath下

## 配置IK域类型,修改schema.xml文件

修改Solr的schema.xml文件，添加FieldType：

<!-- 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>

## 根据需求配置自定义域

<!--IKAnalyzer Field-->

<field name="name\_ik" type="text\_ik" indexed="true" stored="true" />

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

# Solr服务器java接口—solrj使用

/\*\*

\* Solr客户端

\* @author lx

\*

\*/

public class TestSolrJ {

//测试查询

@Test

public void testSolr() throws Exception {

String baseURL = "http://192.168.200.149:8080/solr";

//Solr

SolrServer solrServer = new HttpSolrServer(baseURL);

//查询所有数据

SolrQuery params = new SolrQuery();

params.set("q", "name:瑜伽服");//product\_name ik

//params.addFilterQuery("id:97");

params.setStart(0);

params.setRows(5);

params.setSort("id", ORDER.desc);

//高亮

params.setHighlight(true);

params.addHighlightField("name");

//<span style="color:red">name</span>

params.setHighlightSimplePre("<span style='color:red'>");

params.setHighlightSimplePost("</span>");

QueryResponse response = solrServer.query(params);

//getBrandList Brand

SolrDocumentList docs = response.getResults();

//发现几条

//

long numFound = docs.getNumFound();

System.out.println(numFound);

for(SolrDocument doc : docs){

//22

String id = (String) doc.get("id");

System.out.println(id);

//"name" == 13期瑜伽服

String name = (String) doc.get("name");

System.out.println(name);

//取高亮

Map<String, Map<String, List<String>>> highlighting = response.getHighlighting();

Map<String, List<String>> map = highlighting.get(id);

//就一个数据

List<String> list = map.get("name");

String s = list.get(0);

System.out.println(s);

}

}

//测试Solr的添加

@Test

public void testAdd() throws Exception {

String baseURL = "http://192.168.200.149:8080/solr";

//Solr

SolrServer solrServer = new HttpSolrServer(baseURL);

//添加

SolrInputDocument doc = new SolrInputDocument();

for(int id =4;id<500;id++){

doc.setField("id", id);

doc.setField("name", "13期卖的瑜伽服===" + id);

solrServer.add(doc);

solrServer.commit();

}

}

搭建集合请参考/solr资料/solrcloud

# :上架

## 第一步:back\_page/product/list.jsp

<script type="text/javascript">

function isShow() {

if(Pn.checkedCount('ids')<=0) {

alert("请至少选择一个!");

return;

}

if(!confirm("确定上架吗?")) {

return;

}

//jquery form 表单提交

$("#jvForm").attr("action","/product/isShow.do");

$("#jvForm").attr("method","post").submit();

}

</script>

## 第二步:ProductController

//上架

@RequestMapping("/product/isShow.do")

public String isShow(Integer[] ids,Model model){

//1:上架

productService.isShow(ids);

return "redirect:/product/list.do";

}

## 第三步:ProductServiceImpl

// 上架

public void isShow(Long[] ids) {

// 1：上架状态更变

Product product = null;

for (Long id : ids) {

product = new Product();

// 商品ID

product.setId(id);

// 设置上架状态

product.setIsShow(true);

productDao.updateByPrimaryKeySelective(product);

// 2：添加数据到Solr服务器

//先加载此ID的商品信息 名称 ID 图片Url

Product p = productDao.selectByPrimaryKey(id);

// 添加数据

SolrInputDocument doc = new SolrInputDocument();

//1)添加商品ID 数据库的ID与Solr服务器的ID是一致的

doc.setField("id", id);

//2)商品名称

doc.setField("name\_ik", p.getName());

//3)售价 SKu集合 默认将它的每个Sku的价格作为显示价格

SkuQuery example = new SkuQuery();

example.createCriteria().andProductIdEqualTo(id);

List<Sku> skus = skuDao.selectByExample(example);

doc.setField("price", skus.get(0).getPrice());

//4)图片的URL

ImgQuery iq = new ImgQuery();

iq.createCriteria().andProductIdEqualTo(id).andIsDefEqualTo(true);

List<Img> imgs = imgDao.selectByExample(iq);

doc.setField("url", imgs.get(0).getUrl());

//5)品牌ID

doc.setField("brandId", p.getBrandId());

//6)添加时间

doc.setField("last\_modified", new Date());

try {

solrServer.add(doc);

solrServer.commit();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

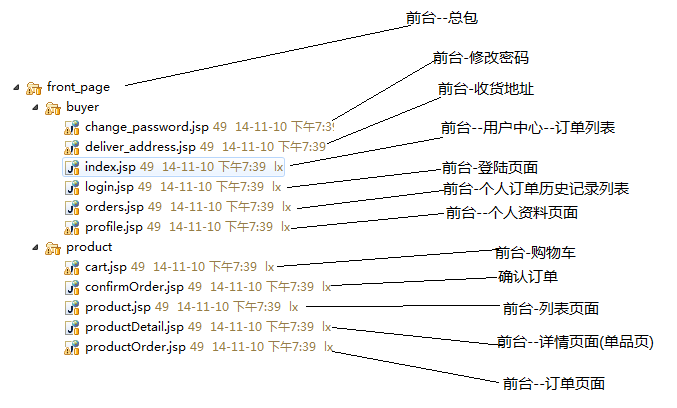
// 3：静态化商品详情页面（单品页）TODO

}

}

# 前台列表页面

## 准备工作:了解前台页面



## 说明

1:可采用Lucene + Sorl进行开发

2:此页面并发访问非常大,是整个网站压力较大的页面

3:由于此页面动态查询的条件过多,在优化技术中不能采用静态化

4:可以采用的技术有:页面缓存,二级缓存,分布式缓存

5:二级缓存在集群下,意义不大,分布式缓存为上上策,分布式缓存可用于全网站,后面再说

6:最终此页面采用页面缓存

页面缓存:

优点:效率比分布式缓存还高

缺点:数据不能时时更新,必须等到缓存时间过期才能更新,而分布式缓存可做到时时更新

针对以上情况:页面缓存的时间设置尤为重要,可在线上产品正式运行期间慢慢修正

## 注意:

1:由于同学们此时此刻尚未学习Lucene + Sorl,本项目仍然采用检索数据库来完成商品筛选

2:/product/display/list.shtml列表页面,需要加载的数据有:1)品牌2)类型3)材质

3:价格,适合人群在本项目中是直接写在页面上的,不从数据库加载

4:注意一点:a标签href不能使用,因为此标签href会请求后台二次

5:分页显示每页数需要添加到业务常量中来,后台列表页与前台列表页的商品数量是不会一样的.不能采用默认每页显示20条数据而要提取出来到业务常量中,统一定义,方便管理

## 第一步:输入<http://localhost:8080/> 进入首页面

## 第二步:需要在web.xml当中配置欢迎页,默认欢迎页在webapp 下的index.jsp

## 第三步:对index.jsp中配置重定向

<%@ page import="java.net.URLEncoder"%>

<%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>

<%response.sendRedirect("/product/display/list.shtml?keyword=" + URLEncoder.encode("瑜伽服","UTF-8")); %>

## 第四步:加载列表页面数据,带上分页即可

1:筛选条件大部分都是中文的,这里请求Url上会有大量中文出现

2:我们要对项目做中文编码转换的处理

3:在web.xml中添加由Spring提供的编码转换过滤器,此处已经在搭建项目环境时已经完成了

4:在Tomcat的conf/server.xml中8080端口处添加URIEncoding=”UTF-8”:

由于Tomcat默认情况下对请求的Url上中文编码为GBK,而我们的Jsp页面里的中文是以UTF-8进行编码的

请求Url是先被Tomcat监听到之后,才进入项目中的

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000" redirectPort="8443" URIEncoding="UTF-8"/>

5:商品筛选区—已选条件

<c:if test="${flag }">

<li>

<label>已选条件：</label>

<p class="sel">

<c:if test="${brandId != null }">

<a href="javascript:void(0);">

<em>品牌：</em>${brandName }

<cite title="关闭此筛选条件">X</cite></a>

</c:if>

<c:if test="${typeId != null}">

<a href="javascript:void(0);">

<em>类型：</em>${typeName }

<cite title="关闭此筛选条件">X</cite></a>

</c:if>

</p>

</li>

</c:if>

6:商品筛选区—品牌

<c:if test="${brandId == null }">

<li>

<b>品牌：</b>

<p>

<a href="javascript:void(0);" title="不限" class="here">不限</a>

<c:forEach items="${brands }" var="brand">

<a href="javascript:void(0);" onclick="javascript:window.location.href='/product/display/list.shtml?brandId=${brand.id }&brandName=${brand.name }&typeId=${typeId}&typeName=${typeName }'" title="${brand.name }">${brand.name }</a>

</c:forEach>

</p>

</li>

</c:if>

7:商品筛选区—类型

<c:if test="${typeId == null }">

<li>

<b>类型：</b>

<p>

<a href="javascript:void(0);" title="不限" class="here">不限</a>

<c:forEach items="${types }" var="type">

<a href="javascript:void(0);" title="${type.name }" onclick="javascript:window.location.href='/product/display/list.shtml?typeId=${type.id}&typeName=${type.name }&brandId=${brandId }&brandName=${brandName }'">${type.name }</a>

</c:forEach>

</p>

</li>

</c:if>

## 第五步:动态条件jquery

<script type="text/javascript">

//关键词

var keyword = '${param.keyword}';

//搜索

function searchKeyword(){

keyword = $("#keyword").val();

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword;

}

//全局变量

var brandId = '${brandId}';

var brandName = '${brandName}';

var price = '${price}';

//品牌过滤

function brandfq(brandId,brandName){

//判断价格存在

if(null != price && "" != price){

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&brandId=" + brandId + "&price=" + price + "&brandName=" + brandName;

}else{

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&brandId=" + brandId + "&brandName=" + brandName;

}

}

//价格

function pricefq(price){

if(null != brandId && "" != brandId){

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&price=" + price + "&brandId=" + brandId + "&brandName=" + brandName;

}else{

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&price=" + price;

}

}

//清理条件

function clearfq(key){

//品牌 价格

if(key == '品牌'){

//判断价格存在

if(null != price && "" != price){

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&price=" + price;

}else{

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword;

}

}else if(key == '价格'){

if(null != brandId && "" != brandId){

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword + "&brandId=" + brandId + "&brandName=" + brandName;

}else{

//发送发台

window.location.href = "/product/display/list.shtml?keyword=" + keyword;

}

}

}

//价格高到低

function orderbyPrice(){

var params = "";

if(brandId != ""){

params += "&brandId=" + brandId;

params += "&brandName=" + brandName;

}

params += "&orderby=" + 1;

//刷新页面 带着条件

window.location.href = "/product/display/list.shtml?keyword=" + keyword + params;

}

//上架时间排序

function orderbyTime(){

var params = "";

if(brandId != ""){

params += "&brandId=" + brandId;

params += "&brandName=" + brandName;

}

if(price != ""){

params += "&price=" + price;

}

params += "&orderby=" + 1;

//刷新页面 带着条件

window.location.href = "/product/display/list.shtml?keyword=" + keyword + params;

}

//去掉当前点击的条件

function clearQuery(q){

var params = "";

if(q == "品牌"){

if(typeId != ""){

params += "&typeId=" + typeId;

params += "&typeName=" + typeName;

}

if(price != ""){

params += "&price=" + price;

}

}else if(q == "类型"){

if(brandId != ""){

params += "&brandId=" + brandId;

params += "&brandName=" + brandName;

}

if(price != ""){

params += "&price=" + price;

}

}

//刷新页面 带着条件

window.location.href = "/product/display/list.shtml?keyword=" + keyword + params;

}

//登陆

function login(){

//去登陆页面 get

window.location.href = "/shopping/login.shtml?returnUrl=" + window.location.href;

}

//我的订单

function myOrder(){

window.location.href = "/buyer/index.shtml";

}

</script>

## 第六步:classpath:config/solr.properties

<!-- solrJ配置 托管给Spring处理 -->

<bean id="solrServer" class="org.apache.solr.client.solrj.impl.HttpSolrServer">

<constructor-arg>

<value>http://192.168.200.149:8080/solr</value>

</constructor-arg>

</bean>

## 第七步:FrontProductController

//商品列表页面

//第一个参数: 品牌ID

//第二个参数: 品牌名称

//第三个参数: 类型Id

//第四个参数: 类型名称

//第五个参数: 页号

@RequestMapping(value = "/product/display/list.shtml")

public String list(String keyword,Integer brandId,String brandName,String price,String orderby,Integer pageNo,ModelMap model){

//品牌 id, name

//1:品牌加载

BrandQuery brandQuery = new BrandQuery();

//1) 可见状态

brandQuery.setIsDisplay(1);

//2) 显示 id ,name

brandQuery.setFields("id,name");

//品牌集合

List<Brand> brands = brandService.getBrandList(brandQuery);

//设置Model

model.addAttribute("brands", brands);

//实例化商品条件对象

//创建商品Query对象

ProductQuery productQuery = new ProductQuery();

//页号

productQuery.setPageNo(Pagination.cpn(pageNo));

//每页数

productQuery.setPageSize(8);

//solr检搜商品

SolrQuery solrQuery = new SolrQuery();

//开始行

solrQuery.setStart(productQuery.getStartRow());

//每页数

solrQuery.setRows(productQuery.getPageSize());

//如果有指定排序规则 按照指定的排序规则来进行 否则按默认排序 默认是ID排序

if("1".equals(orderby)){

solrQuery.addSort("price", ORDER.desc);

}else{

//按照Id倒序

solrQuery.addSort("id", ORDER.desc);

}

//高亮

solrQuery.setHighlight(true);

solrQuery.addHighlightField("product\_name");

solrQuery.setHighlightSimplePre("<span style='color:red'>");

solrQuery.setHighlightSimplePost("</span>");

//分页在页面上展示的参数容器

StringBuilder params = new StringBuilder();

//1:步: 控制已选条件 显示 或隐藏

boolean flag = false;

if(StringUtils.isNotBlank(keyword)){

//设置查询条件

solrQuery.set("q", "product\_name:" + keyword);

params.append("&keyword=" + keyword);

}

if(StringUtils.isNotBlank(price)){

String[] p = price.split("-");

//判断长度

if(p.length == 1){

float pStart = new Float(p[0]);

float pEnd = new Float(99999999);

//价格区间

solrQuery.addFilterQuery("price:[" + pStart + " TO " + pEnd + "]");

}else if (p.length == 2){

float pStart = new Float(p[0]);

float pEnd = new Float(p[1]);

//价格区间

solrQuery.addFilterQuery("price:[" + pStart + " TO " + pEnd + "]");

}

}

//条件的Map

Map<String,String> querys = new LinkedHashMap<String,String>();

//1)品牌Id

if(null != brandId){

flag = true;

//过滤条件

//品牌ID

solrQuery.addFilterQuery("brandId:" + brandId);

model.addAttribute("brandId", brandId);

model.addAttribute("brandName", brandName);

//装条件

querys.put("品牌", brandName);

params.append("&brandId=").append(brandId).append("&brandName=").append(brandName);

}//2)类型ID

if(null != typeId){

flag = true;

//类型ID

solrQuery.addFilterQuery("typeId:" + typeId);

model.addAttribute("typeId", typeId);

model.addAttribute("typeName", typeName);

//装条件

querys.put("类型", typeName);

params.append("&typeId=").append(typeId).append("&typeName=").append(typeName);

}

//去Solr库查询 并返回 Response

QueryResponse response = null;

try {

response = solrServer.query(solrQuery);

} catch (SolrServerException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

//分页对象

//结果集 1) 10 条 2) 总条数

SolrDocumentList docs = response.getResults();

//展示在页面上 分页

Pagination pagination = new Pagination(productQuery.getPageNo(),productQuery.getPageSize(),(int)docs.getNumFound());

//创建商品集合

List<Product> products = new ArrayList<Product>();

//商品对象

Product product = null;

//遍历Solr集合中的值

for(SolrDocument doc : docs){

product = new Product();

String id = (String)doc.get("id");

//商品ID

product.setId(Integer.parseInt(id));

//商品名称

Map<String, Map<String, List<String>>> highlighting = response.getHighlighting();

Map<String, List<String>> map = highlighting.get(id);

List<String> list = map.get("product\_name");

String n = list.get(0);

product.setName(n);

//商品图片

Img img = new Img();

img.setUrl((String) doc.get("url"));

product.setImg(img);

//商品价格

product.setPrice((float) doc.get("price"));

//添加进商品集合中

products.add(product);

}

pagination.setList(products);

//分页在页面上展示

String url = "/product/display/list.shtml";

pagination.pageView(url, params.toString());

model.addAttribute("flag", flag);

model.addAttribute("pagination", pagination);

model.addAttribute("querys", querys);

model.addAttribute("keyword", keyword);

return "product/product";

}

# 商品详情页面

## 第一步:从前台商品列表页面跳转商品详情页面

<!-- 请求路径 -->

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

## 第二步:编写SkuDao.xml

<!-- 查询大于0的库存 条件商品Id-->

<select id="getStock" parameterType="SkuQuery" resultMap="sku">

<include refid="skuSelector"/>

<where>

stock\_inventory > 0 and product\_id = #{productId}

</where>

</select>

## 第三步:编写SkuDao.java

/\*\*

\* 库存大于0所有 条件是商品Id

\*/

public List<Sku> getStock(SkuQuery skuQuery);

## 第四步:编写SkuServiceImpl.java

//获取库存大于0的Sku

public List<Sku> getStock(SkuQuery skuQuery) {

// TODO Auto-generated method stub

List<Sku> skus = skuDao.getStock(skuQuery);

//颜色加载完结

for(Sku sku : skus){

sku.setColor(colorService.getColorByKey(sku.getColorId()));

}

return skus;

}

## 第五步:编写FrontProductController

cn.itcast.core.controller.FrontProductController

/\*\*

\* 前台商品管理

\* @author lx

\*

\*/

@Controller

public class FrontProductController {

//商品详情页面

@RequestMapping(value = "/product/detail.shtml")

public String detail(Integer id,ModelMap model){

//业务数据

//1商品图片加载

Product product = productService.getProductByKey(id);

//2:加载sku

SkuQuery skuQuery = new SkuQuery();

skuQuery.setProductId(id);

//3:此商品 库存大于0的

List<Sku> skus = skuService.getStock(skuQuery);

//去重复

List<Color> colors = new ArrayList<Color>();

//遍历SKu

for(Sku sku : skus){

//判断集合中是否已经有此颜色对象了

if(!colors.contains(sku.getColor())){

colors.add(sku.getColor());

}

}

//把不重复的List<Color> model到页面

model.addAttribute("colors", colors);

model.addAttribute("skus", skus);

model.addAttribute("product", product);

return "product/productDetail";

}

}

## 第六步:编写front\_page/product/productDetail.jsp

### 页面数据部分代码

<div class="w ofc mt">

<div class="l">

<div class="showPro">

<div class="big"><a id="showImg" class="cloud-zoom" href="${product.img.allUrl }" rel="adjustX:10,adjustY:-1"><img alt="" src="${product.img.allUrl }"></a></div>

</div>

</div>

<div class="r" style="width: 640px">

<ul class="uls form">

<li><h2>${product.name }</h2></li>

<li><label>巴 巴 价：</label><span class="word"><b class="f14 red mr" id="p\_id">￥128.00</b>(市场价:<del id="m\_id">￥150.00</del>)</span></li>

<li><label>商品评价：</label><span class="word"><span class="val\_no val3d4" title="4分">4分</span><var class="blue">(已有888人评价)</var></span></li>

<li><label>运　　费：</label><span class="word" id="f\_id">10元</span></li>

<li><label>库　　存：</label><span class="word" id="i\_id">100</span><span class="word" >件</span></li>

<li><label>选择颜色：</label>

<div id="colors" class="pre spec">

<c:forEach items="${colors }" var="color">

<a onclick="colorToRed(this,${color.id})" href="javascript:void(0)" title="${color.name }" class="changToWhite"><img width="25" height="25" data-img="1" src="/res/img/pic/ppp00.jpg" alt="${color.name }"><i>${color.name }</i></a>

</c:forEach>

</div>

</li>

<li id="sizes"><label>尺　　码：</label>

<a href="#none" class="not-allow" id="S" onclick="sizeToRed(this,'S')">S</a>

<a href="#none" class="not-allow" id="M" onclick="sizeToRed(this,'M')">M</a>

<a href="#none" class="not-allow" id="L" onclick="sizeToRed(this,'L')">L</a>

<a href="#none" class="not-allow" id="XL" onclick="sizeToRed(this,'XL')">XL</a>

<a href="#none" class="not-allow" id="XXL" onclick="sizeToRed(this,'XXL')">XXL</a>

</li>

<li><label>我 要 买：</label>

<a id="sub" class="inb arr" style="border: 1px solid #919191;width: 10px;height: 10px;line-height: 10px;text-align: center;" title="减" href="javascript:void(0);" >-</a>

<input id="num" type="text" value="1" name="" size="1" readonly="readonly">

<a id="add" class="inb arr" style="border: 1px solid #919191;width: 10px;height: 10px;line-height: 10px;text-align: center;" title="加" href="javascript:void(0);">+</a></li>

<li class="submit"><input type="button" value="" class="hand btn138x40" onclick="buy(${product.id});"/><input type="button" value="" class="hand btn138x40b" onclick="addCart(${product.id})"/></li>

</ul>

</div>

</div>

### Js部分代码

<script type="text/javascript">

//页面加载时触发

$(function(){

//页面加载时 就给 sub add - + 给二个按钮 注册 click事件

$("#sub").click(function(){

//点击-

//获取 num 值

var num = $("#num").val();

// num + 1

num--;

if(num <= 0){

alert("不允许买0件商品");

return ;

}

//将+后的值赋值回去

$("#num").val(num);

});

$("#add").click(function(){

//点击 +

//获取 num 值

var num = $("#num").val();

// num + 1

num++;

if(num >= buyLimit){

alert("不允许买" + buyLimit + "件商品");

return;

}

//将+后的值赋值回去

$("#num").val(num);

});

$("#colors a:first").trigger("click");

});

//限购

var buyLimit;

//市场价 售价 运费 库存 skuId 限购

//颜色ID

var colorId;

//最小销售单元 skuId

var skuId;

//点击颜色

function colorToRed(target,id){

//重置购买数量

$("#num").val(1);

$("#colors a").each(function(){

$(this).attr("class","changToWhite");

});

$("#sizes a").each(function(){

$(this).attr("class","not-allow");

});

$(target).attr("class","changToRed");

//赋值给颜色Id

colorId = id;

//设置标记

var flag = 0;

<c:forEach items="${skus }" var="sku">

if(id == '${sku.colorId}'){

//利用标记 =0 来判断 第一次进来

if(flag == 0){

//将第一个尺码标红

$("#" + '${sku.size}').attr("class","changToRed");

flag = 1;

//修改市场价

$("#market\_price").html('${sku.marketPrice}');

//售价

$("#sku\_price").html('${sku.skuPrice}');

//运费

$("#delive\_fee").html('${sku.deliveFee}');

//库存

$("#stockInventory").html('${sku.stockInventory}');

//赋值限购

buyLimit = '${sku.skuUpperLimit}';

//赋值skuId

skuId = '${sku.id}';

}else{

//剩余尺码标白

$("#" + '${sku.size}').attr("class","changToWhite");

}

}

</c:forEach>

}

//点击尺码

function sizeToRed(target,id){

//重置购买数量

$("#num").val(1);

$("#sizes a").each(function(){

var v = $(this).attr("class");

if(v != "not-allow"){

$(this).attr("class","changToWhite");

}

});

$(target).attr("class","changToRed");

//点击尺码 变价格等属性

<c:forEach items="${skus }" var="sku">

if(colorId == '${sku.colorId}' && id == '${sku.size}'){

//修改市场价

$("#market\_price").html('${sku.marketPrice}');

//售价

$("#sku\_price").html('${sku.skuPrice}');

//运费

$("#delive\_fee").html('${sku.deliveFee}');

//库存

$("#stockInventory").html('${sku.stockInventory}');

//赋值限购

buyLimit = '${sku.skuUpperLimit}';

//赋值skuId

skuId = '${sku.id}';

}

</c:forEach>

}

//加入购物车 //传入商品ID

function addCart(id){

//采用异步的方式 将商品ID 购买数量 限购 skuId 传递到后台

//更改购物车中的数据 更改Cookie中的数量

var url = "/shopping/addCart.shtml";

var params = {"skuId" : skuId,"buyLimit" : buyLimit,"amount" : $("#num").val()};

$.post(url,params,function(data){

alert("加入购物车成功!");

},"json");

}

//立即购买

function buy(id){

window.location.href = "/shopping/buyCart.shtml?skuId=" + skuId + "&amount=" + $("#num").val() + "&buyLimit=" + buyLimit + "&productId=" + id;

}

</script>

### CSS代码(禁止点击,红色边框,白色边框)

<style type="text/css">

.changToRed {

border: 2px solid #e4393c;

padding: 2px 4px;

float: left;

margin-right: 4px;

text-decoration: none;

}

.changToWhite {

border: 1px solid #ccc;

padding: 2px 4px;

float: left;

margin-right: 4px;

text-decoration: none;

}

.not-allow {

cursor: not-allowed;

color: #999;

border: 1px dashed #ccc;

padding: 2px 4px;

float: left;

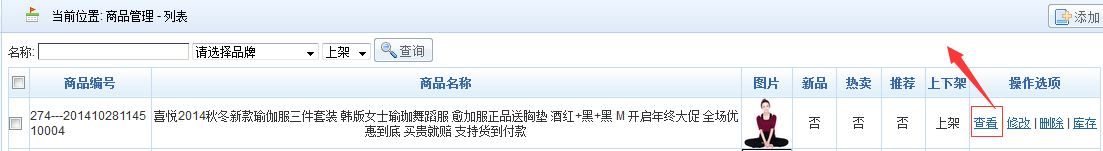
margin-right: 4px;

text-decoration: none;

}

</style>

## 第四步:后台商品列表页面back\_page/product/list.jsp



## 第五步:编写查看按钮

<a href="/product/detail.shtml?id=${entry.id }" target="\_black" class="pn-opt">查看</a> |

此按钮可方便查看自己添加的商品

# 静态化技术Freemarker

## 什么是Freemarker

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-2.3.9.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>

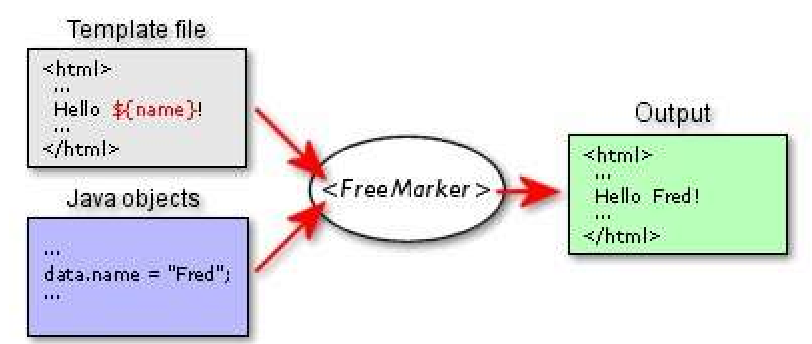
<groupId>freemarker</groupId>

<artifactId>freemarker</artifactId>

<version>2.3.18</version>

</dependency>

## Freemarker原理图



模板 + 数据模型 = 输出

## 第一步:Freemarker环境搭建

### 创建FMDemo类

//第一步:实例化Freemarker的配置类

Configuration conf = new Configuration();

//第二步:给配置类设置路径

String dir = "C:\\Users\\lx\\workspaces-new\\freemarker\\ftl\\";

conf.setDirectoryForTemplateLoading(new File(dir));

Template template = conf.getTemplate("freemarker.ftl");

//第三步:处理模板及数据之间 关系 将数据与模板合成一个html

//第四步: 输出html

Writer out = new FileWriter(new File(dir + "hello.html"));

// 定义数据

Map root = new HashMap();

root.put("world", "世界你好");

//执行生成

template.process(root, out);

//关流

out.flush();

out.close();

### 创建页面模板ftl/freemarker.ftl

${world}

## 第二步:应用例子测试

### 例子1 : 对象输出

//定义数据

Person p = new Person();

Map root = new HashMap();

root.put("person",p);

//页面

${person.id}=${person.name}

### 例子2 : 遍历List

//定义数据

Map root = new HashMap();

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>

//页面

<#list persons as p>

${p.id}/${p.name}

</#list>

### 例子3 : 遍历Map

//定义数据

Map root = new HashMap();

Map mx = new HashMap();

mx.put("fbb","范冰冰");

mx.put("lbb","李冰冰");

root.put("mx",mx);

//页面

第一种写法:

${mx.fbb}/${mx.lbb}

第二种写法:

<#list mx?keys as k>

${mx[k]}

</#list>

### 例子4 : 遍历List<Map>

//定义数据

Map root = new HashMap();

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);

//页面

第一种写法:

<#list maps as m>

${m.id1}/${m.id2}

</#list>

第二种写法:

<#list maps as m>

<#list m?keys as k>

${m[k]}

</#list>

</#list>

### 例子5 : 获取当前选代的索引

//定义数据

Map root = new HashMap();

List<String> persons = new ArrayList<String>();

persons.add("范冰冰");

persons.add("李冰冰");

persons.add("何灵");

root.put("persons", persons);

//页面

<#list persons as p>

${p\_index}

</#list>

### 例子6 : 在模版中进行赋值

1:<#assign x=0 />

${x}

2:<#assign x="${world}" />

${x}

3:<#assign x>世界太好了</#assign>

${x}

4:<#assign x>

<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as n>

${n}

</#list>

</#assign>

${x}

### 例子7 : #if

#### 1:== !=

<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as n>

<#if n != "星期一">

${n}

</#if>

</#list>

#### 2:按索引

<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as n>

<#if n\_index != 0>

${n}

</#if>

</#list>

#### 3: || &&

<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as n>

<#if (n\_index == 1) || (n\_index == 3)>

${n}

</#if>

</#list>

### 例子8 : #else

<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as n>

<#if (n\_index == 1) || (n\_index == 3)>

${n} --红色

<#else>

${n} --绿色

</#if>

</#list>

### 例子9 : 时间格式

//定义数据

Map root = new HashMap();

root.put("cur\_time",new Date());

//页面

1:date

${cur\_time?date}

2:datetime

${cur\_time?datetime}

3:time

${cur\_time?time}

### 例子10 : null处理

//定义数据

Map root = new HashMap();

root.put("val",null);

//页面

${val!"我是null!"}

### 例子11 : 宏定义

1:

<#macro table u>

${u}

</#macro>

<@table u=8 />

2:

<#macro table u>

${u}

<#nested/>

</#macro>

<@table u=8 >这是8</@table>

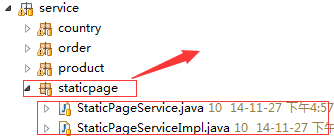
### 例子12 :

将另一个页面引入本页面时可用以下命令完成

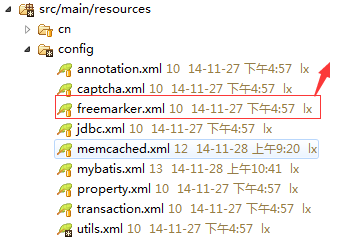
<#include "/include/head.html">

# 整合Freemarker到项目中

## 第一步:创建Service接口及实现类



## 第二步:在classpath下创建config/freemarker.xml



## 第三步:在StaticPageServiceImpl中声明Freemarker配置类

//声明 freemarker配置类

private Configuration conf;

//实现化配置类

public void setFreeMarkerConfigurer(FreeMarkerConfigurer freeMarkerConfigurer){

this.conf = freeMarkerConfigurer.getConfiguration();

}

## 第四步:使用Springmvc自带的Freemarker类实例化Freemarker

<!-- 配置 freemarker -->

<bean id="staticPageService" class="cn.itcast.core.service.staticpage.StaticPageServiceImpl">

<property name="freeMarkerConfigurer">

<bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">

<property name="templateLoaderPath" value="/WEB-INF/ftl/"/>

<property name="defaultEncoding" value="UTF-8"/>

</bean>

</property>

</bean>

## 第五步:StaticPageServiceImpl实现ServletContextAware接口

/\*\*

\* 静态化页面

\* @author lx

\*

\*/

public class StaticPageServiceImpl implements StaticPageService ,ServletContextAware{

//获取项目中的 全路径

public String get(String path){

return servletContext.getRealPath(path);

}

//获取到了 servletContext 容器的上下文

private ServletContext servletContext;

@Override

public void setServletContext(ServletContext servletContext) {

this.servletContext = servletContext;

}

}

## 第六步:路径设定

模板路径:/WEB-INF/ftl/productDetail.html

静态化生成html所在路径"/html/product/" + id + ".html"

## 第七步:静态化方法书写

//静态化开始

public void staticPageIndex(Map<String,Object> data,Integer id) {

//设置模板的文件夹路径

//获取模板 /WEB-INF/ftl/prodcutDetail.html

//输出路径 webapp/html/ + prouduct/ + id + ".html"

String tpl = "productDetail.html";

String outPath = get("/html/product/" + id + ".html");

File f = new File(outPath);

File p = f.getParentFile();

if(!p.exists()){

p.mkdirs();

}

try {

Writer out = new OutputStreamWriter(new FileOutputStream(f), "UTF-8");

Template template = conf.getTemplate(tpl);

template.process(data, out);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (TemplateException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

## 第八步:Map<String,Object> data加载

cn.itcast.core.controller.admin.ProductServiceImpl

// 上架

public void isShow(Long[] ids) throws Exception {

// 1：上架状态更变

Product product = null;

for (Long id : ids) {

product = new Product();

// 商品ID

product.setId(id);

// 设置上架状态

product.setIsShow(true);

productDao.updateByPrimaryKeySelective(product);

// 2：添加数据到Solr服务器

//先加载此ID的商品信息 名称 ID 图片Url

Product p = productDao.selectByPrimaryKey(id);

// 添加数据

SolrInputDocument doc = new SolrInputDocument();

//1)添加商品ID 数据库的ID与Solr服务器的ID是一致的

doc.setField("id", id);

//2)商品名称

doc.setField("name\_ik", p.getName());

//3)售价 SKu集合 默认将它的每个Sku的价格作为显示价格

SkuQuery example = new SkuQuery();

example.createCriteria().andProductIdEqualTo(id);

List<Sku> skus = skuDao.selectByExample(example);

doc.setField("price", skus.get(0).getPrice());

//4)图片的URL

ImgQuery iq = new ImgQuery();

iq.createCriteria().andProductIdEqualTo(id).andIsDefEqualTo(true);

List<Img> imgs = imgDao.selectByExample(iq);

doc.setField("url", imgs.get(0).getUrl());

//5)品牌ID

doc.setField("brandId", p.getBrandId());

//6)添加时间

doc.setField("last\_modified", new Date());

solrServer.add(doc);

solrServer.commit();

// 3：静态化商品详情页面（单品页）TODO

//加载数据 四张表

//通过ID查询商品

Map<String,Object> root = new HashMap<String,Object>();

Product pp = selectProductById(id);

root.put("product", pp);

//通过ID查询Sku表的 （有库存的）

List<Sku> sss = skuService.selectSkuListByProductIdAndStock(id);

root.put("skus", sss);

//去重

Set<Color> colors = new HashSet<Color>();

for (Sku sku : sss) {

colors.add(sku.getColor());

}

root.put("colors", colors);

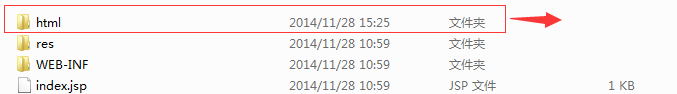
//静态化

staticPageService.index(root, id);

}

}

## 第九步:tomcat的发布目录下



## 第十步:编写前台商品列表页面

front\_page/product/product.jsp

修改前:

<a href="/product/detail.shtml?id=${entry.id }" title="瑜伽服" target="\_blank" class="pic">

从此页面跳转商品详情页面的路径

修改后:

<a href="/html/product/${entry.id }.html" title="瑜伽服" target="\_blank" class="pic">

## 第十一步:点击此按钮可直接跳转至静态页面



# CMS项目中添加静态化程序

## 创建Maven结构的CMS项目

## 利用HttpClient跨项目访问CMS项目

# 配置验证码

## 准备开始

cn.itcast.common.web.Constants

/\*\*

\* web 常量

\* @author lx

\*

\*/

public abstract class Constants {

/\*\* 用户 session 的 cookie 名称\*/

public static final String SESSION\_ID = "JSESSIONID";

}

## 第一步:创建SessionProvider接口

cn.itcast.common.web.session.SessionProvider

public interface SessionProvider {

//四个方法

//往Session设置值

//参数1:request

//参数2:Session中的Key

//参数3:Session中的Value

public void setAttribute(HttpServletRequest request,HttpServletResponse response,String name,Serializable value);

//从Session里取值

public Serializable getAttribute(HttpServletRequest request,HttpServletResponse response,String name);

//退出登陆

public void logout(HttpServletRequest request,HttpServletResponse response);

//获取SessionId

public String getSessionId(HttpServletRequest request,HttpServletResponse response);

}

## 第二步:创建HttpSessionProvider实现类

cn.itcast.common.web.session.HttpSessionProvider

/\*\*

\* Session提供类

\* @author lx

\*

\*/

public class HttpSessionProvider implements SessionProvider{

public void setAttribute(HttpServletRequest request,HttpServletResponse response, String name,

Serializable value) {

//没获取到 创建一个新的Session

HttpSession session = request.getSession();

//设置值 不用判断

session.setAttribute(name, value);

}

public Serializable getAttribute(HttpServletRequest request,HttpServletResponse response, String name) {

HttpSession session = request.getSession(false);

//判断是否为Null

if(null != session){

return (Serializable) session.getAttribute(name);//空指针

}

return null;

}

public void logout(HttpServletRequest request, HttpServletResponse response) {

// TODO Auto-generated method stub

HttpSession session = request.getSession(false);

if(null != session){

session.invalidate();

}

//Cookie 中的 JSESSIONID EEWWEWEWEFDFDTETETETE 32

Cookie cookie = new Cookie("JSESSIONID",null);

cookie.setMaxAge(0);

cookie.setPath("/");

response.addCookie(cookie);

}

public String getSessionId(HttpServletRequest request,

HttpServletResponse response) {

return request.getSession().getId();

}

}

## 第三步:创建classpath:config/utils.xml

<!-- session 提供类 -->

<bean id="sessionProvider" class="cn.itcast.common.web.session.HttpSessionProvider"/>

## 第四步:创建FrontCodeController类

/\*\*

\* 验证码

\* @author lx

\*

\*/

@Controller

public class FrontCodeController {

@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]);

}

sessionProvider.setAttribute(request,response, Constants.CODE\_NAME, sb.toString());

try {

ImageIO.write(img, "JPG", response.getOutputStream());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

## 第八步:编写登陆页面front\_page/buyer/login.jsp

<li>

<label for="captcha">验证码：</label>

<span class="bg\_text small">

<input type="text" id="captcha" name="captcha" maxLength="7"/>

</span>

<img src="/captcha.svl" 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>

# 用户登陆

## 准备

## js获取当前页面的URL

//获取当前窗口的Url

//returnUrl=<http://localhost:8080/shopping/buyCart.shtml?skuId=510&amount>=1

window.location.href

//获取当前窗口的主机名 例如:http://localhost:8080

window.location.host

//获取当前窗口的端口 例如: 8080

window.location.port

//获取当前窗口的路径 例如: /shopping/buyCart.shtml

window.location.pathname

//获取当前文档的Url

document.URL

//获取参数 例如: ?skuId=510&amount=1

window.location.search

//跳出当前窗口,打开新窗口

window.open(url);

在从/front\_page/product/product.jsp跳转到登陆页面

//跳转到登陆页面

function login(){

window.location.href = "/shopping/login.shtml?returnUrl=" + encodeURIComponent(window.location.href);

}

## 第一步:创建FrontProfileController类

cn.itcast.core.controller.FrontProfileController

/\*\*

\* 个人中心

\* 包括登陆 个人资料 收货地址

\* 验证码生成

\* @author lx

\*

\*/

@Controller

public class FrontProfileController {

}

## 第二步:编写前台登陆的跳转页面

//前台登陆页面跳转

@RequestMapping(value = "/shopping/login.shtml",method = RequestMethod.GET)

public String login(){

//跳转用户页面

return "buyer/login";

}

## 第三步:前台登陆页面POST提交方法

//前台登陆 提交登陆表单

@RequestMapping(value = "/shopping/login.shtml",method = RequestMethod.POST)

public String login(Buyer buyer,String captcha,String returnUrl,HttpServletRequest request,ModelMap model){

}

## 第四步:编写登陆页面front\_page/buyer/login.jsp

**登陆后返回的路径returnUrl**

**用户名**

**密码**

<form id="jvForm" action="/shopping/login.shtml" method="post">

<input type="hidden" name="returnUrl" value="${param.returnUrl}"/>

<ul class="uls form">

<li id="errorName" class="errorTip" <c:if test="${empty error }">style="display:none"</c:if>>${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" name="code" maxLength="7"/>

</span>

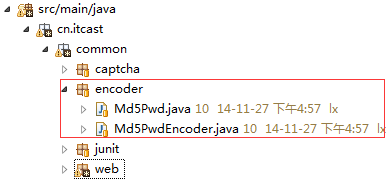
<img src="/shopping/getCodeImage.shtml" onclick="this.src='/shopping/getCodeImage.shtml?d='+new Date()" class="code" alt="换一张" /><a href="javascript:void(0);" onclick="$('.code').attr('src','/shopping/getCodeImage.shtml?d='+new Date())" title="换一张">换一张</a></li>

<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>

## 第五步:创建Md5提供类



## 第六步:编写Md5PwdEncoder

/\*\*

\* 密码Md5提供类

\* @author lx

\*

\*/

public class Md5PwdEncoder implements Md5Pwd{

//加密

public String encoder(String password) {

String algorithm = "MD5";

MessageDigest instance = null;

try {

instance = MessageDigest.getInstance(algorithm);

} catch (NoSuchAlgorithmException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

byte[] digest = instance.digest(password.getBytes());

char[] encodeHex = Hex.encodeHex(digest);

return new String(encodeHex);

}

//测试一下Md5

public static void main(String[] args) {

Md5PwdEncoder m = new Md5PwdEncoder();

String encoder = m.encoder("123456");

System.out.println(encoder);

}

}

## 第七步:编写classpath:config/utils.xml

<!-- md5工具类 -->

<bean id="md5Pwd" class=" cn.itcast.common.encoder.Md5PwdEncoder"/>

## 第八步:在FrontProfileController类中

//注入

@Autowired

private BuyerService buyerService;

@Autowired

private SessionProvider sessionProvider;

@Autowired

private Md5Pwd md5Pwd;

## 第九步:验证码是否正确,用户是否存在,密码是否正确.

//提交登陆

@RequestMapping(value = "/shopping/login.shtml",method=RequestMethod.POST)

public String login(String username,String password,String code,String returnUrl,Model model,HttpServletRequest request,HttpServletResponse response) throws UnsupportedEncodingException{

//验证码不能为空

if(null != code){

//验证码必须正确

String c = (String) request.getSession().getAttribute(Constants.CODE\_NAME);

if(c.equalsIgnoreCase(code)){

//用户名不能空

if(null != username){

//密码不能为空

if(null != password){

Buyer buyer = buyerService.selectBuyerById(username);

//用户名必须正确

if(null != buyer){

//密码必须正确

if(buyer.getPassword().equals(encodePassword(password))){

//用户放入Session中

request.getSession().setAttribute(Constants.BUYER\_NAME, buyer);

if(null != returnUrl){

//returnUrl : http://locahost:8080/product/display/list.shtml?keyword="瑜伽服"&price=0-79

URL url = null;

try {

url = new URL(returnUrl);

String query = url.getQuery();

//判断是否有条件

if(null != query){

String[] q = query.split("&");

for (String string : q) {

String[] split = string.split("=");

model.addAttribute(split[0], URLDecoder.decode(split[1],"UTF-8"));

}

}

} catch (MalformedURLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return "redirect:" + url.getPath();

}else{

//会员中心

return "/buyer/index.shtml";

}

}else{

model.addAttribute("error", "密码必须正确");

return "buyer/login";

}

}else{

model.addAttribute("error", "用户名必须正确");

return "buyer/login";

}

}else{

model.addAttribute("error", "密码不能为空");

return "buyer/login";

}

}else{

model.addAttribute("error", "用户名不能空");

return "buyer/login";

}

}else{

model.addAttribute("error", "验证码必须正确");

return "buyer/login";

}

}else{

model.addAttribute("error", "验证码不能为空");

return "buyer/login";

}

}

## 第十步:在登陆页面显示错误信息

<li id="errorName" class="errorTip" <c:if test="${empty error }">style="display:none"</c:if>>${error}</li>

## 第十一步:登陆成功后默认跳转个人中心

//跳转个人中心

@RequestMapping(value = "/buyer/index.shtml")

public String index(){

//加载订单

return "buyer/index";

}

## 第十二步:退出登陆

//退出登陆

@RequestMapping(value = "/shopping/logout.shtml")

public String logout(String returnUrl ,HttpServletRequest request,HttpServletResponse response){

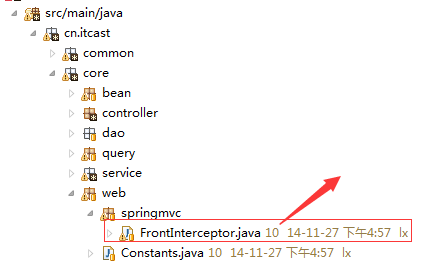
sessionProvider.logout(request, response);

return "redirect:" + returnUrl;

}

# Springmvc拦截器的配置及使用

## 第一步:创建FrontInterceptor



## 第二步:编写FrontInterceptor

/\*\*

\* 前台拦截器

\* 判断 用户是否登陆

\* 全站上下文

\* 全局信息

\* @author lx

\*/

public class FrontInterceptor implements HandlerInterceptor{

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

// TODO Auto-generated method stub

//方法前

return false;

}

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

// TODO Auto-generated method stub

}

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

throws Exception {

// TODO Auto-generated method stub

}

}

## 第三步:编写classpath:springmvc-front.xml

<!-- 处理器映射器 -->

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">

<property name="interceptors">

<ref bean="frontInterceptor"/>

</property>

</bean>

<!-- 处理器适配器 -->

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>

<!-- 创建一个拦截器类 -->

<bean id="frontInterceptor" class="cn.itcast.core.web.springmvc.FrontInterceptor"/>

## 第四步:测试拦截器是否有效

## 第五步:编写FrontInterceptor

@Autowired

private SessionProvider sessionProvider;

//拦截 包括/buyer 的请求路径

public static final String INTERCETOR\_URL = "/buyer/";

//返回路径

public static final String RETURNURL = "returnUrl";

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

//System.out.println("方法前");

if(adminId != null){

Buyer b = new Buyer();

b.setUsername("fbb2014");

sessionProvider.setAttribute(request, Constants.BUYER\_NAME, b);

request.setAttribute("isLogin", true);

}else{

Buyer buyer = (Buyer) sessionProvider.getAttribute(request, Constants.BUYER\_NAME);

if(null != buyer){

request.setAttribute("isLogin", true);

}else{

request.setAttribute("isLogin", false);

}

String requestURI = request.getRequestURI();

if(requestURI.contains(INTERCETOR\_URL)){

if(null == buyer){

response.sendRedirect("/shopping/login.shtml?" + RETURNURL + "=" + requestURI);

return false;

}

}

}

return true;

}

## 第六步:编写front\_page/buyer/login.jsp

要求从哪个页面跳转过来,登陆后跳转回哪个页面

<input type="hidden" name="returnUrl" value="${param.returnUrl}"/>

## 第七步:商品列表页页是否在登陆状态判断代码

<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)" 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="/buyer/index.shtml" title="我的订单">我的订单</a></li>

</c:if>

# 个人资料

## 第一步:编写FrontProfileController类

//跳转个人资料

@RequestMapping(value = "/buyer/profile.shtml")

public String profile(HttpServletRequest request,ModelMap model){

Buyer buyer = (Buyer) sessionProvider.getAttibuter(request, Constants.BUYER\_SESSION);

//

Buyer b = buyerService.getBuyerByKey(buyer.getUsername());

model.addAttribute("buyer", b);

//查省

List<Province> provinces = provinceService.getProvinceList(null);

//通过用户的省ID

CityQuery cityQuery = new CityQuery();

cityQuery.setProvince(b.getProvince());

//市

List<City> citys = cityService.getCityList(cityQuery);

TownQuery townQuery = new TownQuery();

townQuery.setCity(b.getCity());

//县

List<Town> towns = townService.getTownList(townQuery);

model.addAttribute("provinces", provinces);

model.addAttribute("citys", citys);

model.addAttribute("towns", towns);

return "buyer/profile";

}

## 第二步:front\_page/buyer/profile.jsp

<li>

<label for="username">用 户 名：</label>

<span class="word">fbb2016--${buyer.username }</span>

</li>

<li>

<label for="username">邮　　箱：</label>

<span class="word">112624349@qq.com--${buyer.email }</span>

</li>

<li>

<label for="realName">真实姓名：</label>

<span class="bg\_text"><input type="text" id="realName" name="realName" maxLength="32" value="${buyer.realName }" /></span>

<span class="pos"><span class="tip okTip">&nbsp;</span></span>

</li>

## 第三步:页面上性别判断

<li>

<label for="gender">性　　别：</label>

<span class="word">

<input type="radio" name="gender" value="SECRECY" <c:if test="${'SECRECY' == buyer.gender }">checked="checked"</c:if>/>保密

<input type="radio" name="gender" value="MAN" <c:if test="${'MAN' == buyer.gender }">checked="checked"</c:if>/>男

<input type="radio" name="gender" value="WOMAN" <c:if test="${'WOMAN' == buyer.gender }">checked="checked"</c:if>/>女

</span>

</li>

## 第四步:省,市,县级联

<li>

<label for="residence">居 住 地：</label>

<span class="word">

<select name="province" id="province" onchange="changeProvince(this.value)">

<option value="" selected>省/直辖市</option>

<c:forEach items="${provinces }" var="province">

<option value="${province.code }" <c:if test="${province.code == buyer.province }">selected</c:if>>${province.name }</option>

</c:forEach>

</select>

<select name="" id="city" onchange="changeCity(this.value)">

<option value="" selected>城市</option>

<c:forEach items="${citys }" var="city">

<option value="${city.code }" <c:if test="${city.code == buyer.city }">selected</c:if>>${city.name }</option>

</c:forEach>

</select>

<select name="" id="town">

<option value="" selected>县/区</option>

<c:forEach items="${towns }" var="town">

<option value="${town.code }" <c:if test="${town.code == buyer.town }">selected</c:if>>${town.name }</option>

</c:forEach>

</select>

</span>

</li>

## 第五步:省,市县级联Jquyer部分

//改变省

function changeProvince(value){

var url = "/buyer/city.shtml";

var params = {"province" : value};

$.post(url,params,function(data){

var citys = data.citys;

var html = '<option value="" selected>城市</option>';

for(var i=0;i<citys.length;i++){

html += '<option value="' + citys[i].code +'">' + citys[i].name + '</option>';

}

$("#city").html(html);

//改变县

var html1 = '<option value="" selected>县/区</option>';

$("#town").html(html1);

});

}

//改变城市

function changeCity(value){

var url = "/buyer/town.shtml";

var params = {"city" : value};

$.post(url,params,function(data){

var towns = data.towns;

var html = '<option value="" selected>县/区</option>';

for(var i=0;i<towns.length;i++){

html += '<option value="' + towns[i].code +'">' + towns[i].name + '</option>';

}

$("#town").html(html);

});

}

## 第六步:保存按钮(修改个人资料)

<li><label for="">&nbsp;</label><input type="button" onclick="updateProfile('${buyer.username}')" value="保存" class="hand btn66x23" /></li>

## 第七步:异步保存页面Js

//保存

function updateProfile(username){

var url = "/buyer/o\_profile.shtml";

//真实姓名

var realName = $.trim($("#realName").val());

//性别

var gender = $("input[name='gender']:checked").val();

//alert(gender);

//省编号

var province = $("#province").val();

var city = $("#city").val();

var town = $("#town").val();

//

//地址

var addr = $.trim($("#address").val());

var params = {"username" : username,"realName" : realName,"gender" : gender,"province" : province,"city" : city,"town" : town,"addr" : addr};

$.post(url,params,function(data){

alert(data.message);

},"json");

}

</script>

## 第八步:编写FrontProfileController类,保存方法

//保存个人资料

@RequestMapping(value = "/buyer/o\_profile.shtml")

public void o\_profile(Buyer buyer,HttpServletResponse response){

buyerService.updateBuyerByKey(buyer);

JSONObject jo = new JSONObject();

jo.put("message", "修改成功");

ResponseUtils.renderJson(response, jo.toString());

}

# 收货地址

## 第一步:跳转收货地址页面

//跳转收货地址

@RequestMapping(value = "/buyer/deliver\_address.shtml")

public String deliverAddRess(HttpServletRequest request,ModelMap model){

//加载已有地址

Buyer buyer = (Buyer) sessionProvider.getAttribute(request, Constants.BUYER\_NAME);

//设置地址条件对象

AddrQuery addrQuery = new AddrQuery();

addrQuery.setBuyerId(buyer.getUsername());

List<Addr> addrs = addrService.getAddrList(addrQuery);

model.addAttribute("addrs", addrs);

return "buyer/deliver\_address";

}

## 第二步:页面front\_page/buyer/deliver\_address.jsp

### 地址列表部分

<c:forEach items="${addrs }" var="addr">

<tr <c:if test="${addr.isDef == 1 }">class="here"</c:if>>

<td>${addr.name }</td>

<td>${addr.city }</td>

<td>${addr.addr }</td>

<td>${addr.phone }</td>

<td class="def blue">设为默认</td>

<td><a href="javascript:void(0);" title="修改" onclick="modify(${addr.id},'${addr.name }','${addr.addr }','${addr.phone }',${addr.isDef })" class="blue">[修改]</a><a href="javascript:void(0);" title="删除" onclick="del(this)" class="blue">[删除]</a></td>

</tr>

</c:forEach>

### 省,市,县级联异步查询Js部分

$(function(){

var url = "/buyer/province.shtml";

$.post(url,function(data){

var provinces = data.provinces;

var html = '<option value="" selected>省/直辖市</option>';

for(var i=0;i<provinces.length;i++){

html += '<option value="' + provinces[i].code +'">' + provinces[i].name + '</option>';

}

$("#province").html(html);

});

});

function changeProvince(value){

var url = "/buyer/city.shtml";

var params = {"province" : value};

$.post(url,params,function(data){

var citys = data.citys;

var html = '<option value="" selected>城市</option>';

for(var i=0;i<citys.length;i++){

html += '<option value="' + citys[i].code +'">' + citys[i].name + '</option>';

}

$("#city").html(html);

//更改

var html1 = '<option value="" selected>县/区</option>';

$("#town").html(html);

});

}

//改变城市

function changeCity(value){

var url = "/buyer/town.shtml";

var params = {"city" : value};

$.post(url,params,function(data){

var towns = data.towns;

var html = '<option value="" selected>县/区</option>';

for(var i=0;i<towns.length;i++){

html += '<option value="' + towns[i].code +'">' + towns[i].name + '</option>';

}

$("#town").html(html);

});

}

### 省,市,县级联异步查询页面部分

<li>

<label for="residence"><samp>\*</samp>地　　址：</label>

<span class="word">

<select name="province" id="province" onchange="changeProvince(this.value)">

<option value="" selected>省/直辖市</option>

<option value="1">1</option>

</select>

<select name="city" id="city" onchange="changeCity(this.value)">

<option value="" selected>城市</option>

<option value=""></option>

</select>

<select name="town" id="town">

<option value="" selected>县/区</option>

<option value=""></option>

</select>

</span>

</li>

## 省,市,县级联异步查询后台代码

#### 省

//加载省 市 县

@RequestMapping(value = "/buyer/province.shtml")

public void province(HttpServletResponse response){

List<Province> provinces = provinceService.getProvinceList(null);

JSONObject jo = new JSONObject();

jo.put("provinces", provinces);

ResponseUtils.renderJson(response, jo.toString());

}

#### 市

//加载市

@RequestMapping(value = "/buyer/city.shtml")

public void city(String province,HttpServletResponse response){

//设置城市条件

CityQuery cityQuery = new CityQuery();

cityQuery.setProvince(province);

List<City> citys = cityService.getCityList(cityQuery);

JSONObject jo = new JSONObject();

jo.put("citys", citys);

ResponseUtils.renderJson(response, jo.toString());

}

#### 县

//加载区

@RequestMapping(value = "/buyer/town.shtml")

public void town(String city,HttpServletResponse response){

//设置区条件

TownQuery townQuery = new TownQuery();

townQuery.setCity(city);

List<Town> towns = townService.getTownList(townQuery);

JSONObject jo = new JSONObject();

jo.put("towns", towns);

ResponseUtils.renderJson(response, jo.toString());

}

## 第三步:列表后的修改按钮

//修改按钮

function modify(id,name,addr,phone,isDef){

$("#id").val(id);

$("#username").val(name);

$("#addr").val(addr);

$("#telphone").val(phone);

if(isDef == 1){

$("#isDef").attr("checked",true);

$("#isDef").attr("disabled",true);

}

if(isDef == 0){

$("#isDef").attr("checked",false);

$("#isDef").attr("disabled",false);

}

}

## 第四步:保存按钮页面部分

//保存

function addAddr(){

var id = $("#id").val();

//收货人姓名

var name = $("#username").val();

//街道地址

var addr = $("#addr").val();

//手机号

var phone = $("#telphone").val();

//省的汉字

var province = $("#province").find("option:selected").text();

//市的汉字

var city = $("#city").find("option:selected").text();

//县的汉字

var town = $("#town").find("option:selected").text();

//未被中

var isDef = 0;

//是否为默认

if($("#isDef").attr("checked")){

isDef = 1;

}

var url = "/buyer/deliverAddr.shtml";

var params = {"id" : id , "name" : name, "city" : province + city + town,"addr" : addr,"phone" : phone,"isDef" : isDef};

$.post(url,params,function(data){

alert(data.message);

//刷新页面

window.location.href = "/buyer/deliver\_address.shtml";

},"json");

}

## 第五步:保存按钮后台代码ProfileController

//保存或修改收货地址

@RequestMapping(value = "/buyer/deliverAddr.shtml")

public void delieveAddr(Addr addr,HttpServletResponse response,HttpServletRequest request){

JSONObject jo = new JSONObject();

//根据Id是否为Null为判断 是修改还是保存

if(null != addr && null != addr.getId()){

//修改

addrService.updateAddrByKey(addr);

jo.put("message", "修改成功!");

}else{

//保存

Buyer buyer = (Buyer) sessionProvider.getAttribute(request, Constants.BUYER\_NAME);

addr.setBuyerId(buyer.getUsername());

addrService.addAddr(addr);

jo.put("message", "保存成功!");

}

ResponseUtils.renderJson(response, jo.toString());

}

## 第六步:从非默认收货地址修改为默认地址时?之前默认的收货地址变更为非默认

cn.itcast.core.service.user.AddrServiceImpl 中的添加地址或修改地址的方法都要加下面这段代码

if(addr.getIsDef() != 0){

AddrQuery addrQuery = new AddrQuery();

addrQuery.setBuyerId(addr.getBuyerId());

addrQuery.setFields("id");

List<Addr> addrs = addrDao.getAddrList(addrQuery);

for(Addr a : addrs){

a.setIsDef(0);

addrDao.updateAddrByKey(a);

}

}

# 购买按钮—跳转到购物车（非登陆）



## 第一步:要求加入购物车的在没有登陆的时候存储起来

* 可采用Cookie或Session或数据库进行存储
* 最后我们采用Cookie进行存储

## 第二步:编写front\_page/product/productDetail.jsp

### 购买按钮

//立即购买 参数商品ID

function buy(id){

window.location.href = "/shopping/buyCart.shtml?skuId=" + skuId + "&amount=" + $("#num").val() + "&buyLimit=" + buyLimit + "&productId=" + id;

}

## 第三步:创建FrontCartController类并编写购买方法

/\*\*

\* 购物车

\* @author lx

\*

\*/

@Controller

public class FrontCartController {

//购买

@RequestMapping(value = "/shopping/buyCart.shtml")

public String buyCart(Integer skuId,Integer amount,Integer buyLimit,Integer productId,HttpServletResponse response,HttpServletRequest request, ModelMap model){

}

}

## 第四步:编写Cookie进行数据存储测试

### 准备

/\*\*

\* 购物车在Cookie中的名称

\*/

public static final String BUYCART\_COOKIE = "buyCart\_cookie";

### 实例化Cookie -1关闭浏览器消毁 0立即消毁

//实例化Cookie,并指定Cookie的名称

Cookie cookie = new Cookie(Constants.BUYCART\_COOKIE,str.toString());

//1)给Cookie设置时间,如果不设置时间,默认下 时间为 -1 -1表示关闭浏览器会自动清理掉Cookie

cookie.setMaxAge(60\*60\*24);

cookie.setPath("/");

//将Cookie发送到浏览器

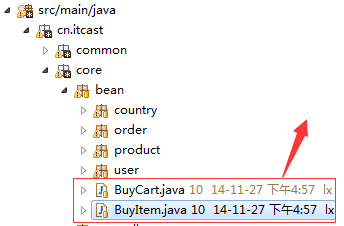
response.addCookie(cookie);

## 第五步:跳转购物车页面



## 第六步:创建购物车对象

### 位置



### 购物项对象BuyItem

/\*\*

\* 购物的商品,购买项

\* @author lx

\*

\*/

public class BuyItem {

//最小销售单元

private Sku sku ;

//买多少件 数量

private int amount = 1;

//省略set get ……

}

### 购物车对象BuyCart

/\*\*

\* 购物车

\* @author lx

\*

\*/

public class BuyCart {

//有多个购物项

private List<BuyItem> items = new ArrayList<BuyItem>();

//留言

private String note;

//最后一次加入购物车时的商品Id

private Integer productId;

//省略 set get ……

}

### 添加购物项

//添加购物项

public void addItem(BuyItem item){

if(items.contains(item)){

for(BuyItem it : items){

if(it.equals(item)){

int amount = item.getAmount() + it.getAmount();

if(amount >= it.getSku().getSkuUpperLimit()){

it.setAmount(it.getSku().getSkuUpperLimit() - 1);

}else{

it.setAmount(item.getAmount() + it.getAmount());

}

break;

}

}

}else{

items.add(item);

}

}

### 删除购物项

//删除购物项

public void removeItem(BuyItem item){

if(items.contains(item)){

items.remove(item);

}

}

### 消空购物车

//消空购物车

public void clearItem(){

items.clear();

}

### 商品数量

//商品数量

@JsonIgnore

public int getProductAmount(){

int result = 0;

for(BuyItem item : items){

result += item.getAmount();

}

return result;

}

### 商品金额

@JsonIgnore

public Double getProductPrice(){

Double result = 0.00;

for(BuyItem item : items){

result += item.getSku().getSkuPrice()\*item.getAmount();

}

return result;

}

### 商品运费

//商品运费

@JsonIgnore

public Double getFee(){

Double result = 0.00;

if(getProductPrice() < 39.00){

result = 10.00;

}

return result;

}

### 应付金额

//应付金额

@JsonIgnore

public Double getTotalPrice(){

Double result = 0.00;

result += getProductPrice() + getFee();

return result;

}

### 购物车是否有商品

//购物车是否有商品

@JsonIgnore

public boolean isExitItem(){

boolean result = false;

if(items.size() > 0){

result = true;

}

return result;

}

## 第七步:由于Cookie中不能装对象,采用将购物车序列化JSON格式

/\*\*

\* 测试将对象序列化成JSON格式

\* @author lx

\*

\*/

public class TestCookie {

public static void main(String[] args) throws JsonGenerationException, JsonMappingException, IOException {

//用户对象

Buyer b = new Buyer();

b.setUsername("fbb2014");

//对象序列化字符串

//

ObjectMapper om = new ObjectMapper();

om.setSerializationInclusion(Inclusion.NON\_NULL);

StringWriter str = new StringWriter();

//

om.writeValue(str, b);

System.out.println(str.toString());

//反序列化过程

//字符串转成对象

Buyer readValue = om.readValue(str.toString(), Buyer.class);

System.out.println(readValue.toString());

}

}

## 第八步:从Cookie中取出Value并反序列化为购物车对象

//购物车对象

BuyCart buyCart = null;

//把对象存储在Cookie中 需要将对象序列化成字符串

ObjectMapper om = new ObjectMapper();

//序列化的字符串中不要有Null

om.setSerializationInclusion(Inclusion.NON\_NULL);

Cookie[] cookies = request.getCookies();

if(null != cookies){

for(Cookie cookie : cookies){

if(Constants.BUYCART\_COOKIE.equals(cookie.getName())){

String value = cookie.getValue();

//System.out.println(value);

buyCart = om.readValue(value, BuyCart.class);

break;

}

}

}

## 第九步:如果是第一步加入购物车,没有Cookie

//实例化购物车 只为第一次购物

if(null == buyCart){

buyCart = new BuyCart();

}

## 第十步:将购物车序列化成JSON并存储在Cookie中

if(null != skuId){

//实例化SKU

Sku sku = new Sku();

//将SkuID设置给Sku对象

sku.setId(skuId);

//设置限购

sku.setSkuUpperLimit(buyLimit);

//实例化购物车

BuyItem item = new BuyItem();

//将Sku对象设置给购物项

item.setSku(sku);

//将商品数量设置给购物项

item.setAmount(amount);

//将商品Id加入购物车

buyCart.setProductId(productId);

//添加购物项进购物车

buyCart.addItem(item); //同类产品需要比较 如果SkuId一样 视为相同产品

//实例化一个字符串的流,存储序列化后的字符串

StringWriter str = new StringWriter();

//开始序列化

om.writeValue(str, buyCart);

//实例化Cookie,并指定Cookie的名称

Cookie cookie = new Cookie(Constants.BUYCART\_COOKIE,str.toString());

//1)给Cookie设置时间,如果不设置时间,默认下 时间为 -1 -1表示关闭浏览器会自动清理掉Cookie

cookie.setMaxAge(60\*60\*24);

cookie.setPath("/");

//将Cookie发送到浏览器

response.addCookie(cookie);

}

## 第十一步:把购物车装满,显示在购物车页面上

//将购物车里的数据装满,为了在购物车页面显示的数据全都有

//加载SkuId对应的属性

List<BuyItem> items = buyCart.getItems();

for(BuyItem it : items){

Sku skuByKey = skuService.getSkuByKey(it.getSku().getId());

it.setSku(skuByKey);

}

//将购物车model到购物车页面上去

model.addAttribute("buyCart", buyCart);

//将最后添加的商品ID 传回页面

model.addAttribute("productId", buyCart.getProductId());

return "product/cart";

## 第十二步:购物车页面front\_page/product/cart.jsp

### 显示商品部分的代码

<c:forEach items="${buyCart.items }" var="item">

<tr>

<td class="nwp pic">

<ul class="uls">

<li>

<a href="#" title="${item.sku.product.name }" class="pic"><img src="${item.sku.product.img.allUrl }" alt="${item.sku.product.name }" /></a>

<dl>

<dt><a href="#" title="${item.sku.product.name }">${item.sku.product.name }--${item.sku.color.name}--${item.sku.size }</a></dt>

<dd><span class="red">【赠品】</span>

<p class="box\_d bg\_gray2 gray"><a href="#" title="瑜伽丝带">瑜伽丝带</a><br /></p>

</dd>

</dl>

</li>

</ul>

</td>

<td>￥${item.sku.skuPrice}</td>

<td><a href="javascript:void(0);" title="减" class="inb arr" onclick="subProductAmount(${item.sku.id },${item.sku.skuUpperLimit})">-</a><input type="text" class="txts" size="1" name="" value="${item.amount }" readonly="readonly" id="num${item.sku.id }"/><a href="javascript:void(0);" title="加" class="inb arr" onclick="addProductAmount(${item.sku.id },${item.sku.skuUpperLimit})">+</a></td>

<td class="blue"><a href="javascript:void(0);" title="删除" onclick="delProduct(${item.sku.id})">删除</a></td>

</tr>

</c:forEach>

### 购物车统计处代码

<span class="r box\_gray">

<dl class="total">

<dt>购物车金额小计：<cite>(共<var id="productAmount">${buyCart.productAmount }</var>个商品)</cite></dt>

<dd><em class="l">商品金额：</em>￥<var id="productPrice">${buyCart.productPrice }</var>元</dd>

<dd><em class="l">运费：</em>￥<var id="fee">${buyCart.fee }</var>元</dd>

<dd class="orange"><em class="l">应付总额：</em>￥<var id="totalPrice">${buyCart.totalPrice }</var>元</dd>

<dd class="alg\_c"><input id="settleAccountId" type="button" value="结算" class="hand btn136x36a" onclick="trueBuy();"/></dd>

</dl>

</span>

# 购物车之登陆时

@Autowired

private Jedis jedis;

// 购物按钮

@RequestMapping(value = "/shopping/buyerCart.shtml")

public String buyerCart(Long skuId, Integer amount, Model model,

HttpServletRequest request, HttpServletResponse response)

throws JsonParseException, JsonMappingException, IOException {

// 购物车转成JSON格式

BuyerCart buyerCart = null;

// 下向所有程序非登陆

ObjectMapper om = new ObjectMapper();// 对象转JSON 对象（标准JavaBean）

// 不要NULL

om.setSerializationInclusion(Inclusion.NON\_NULL);

// 获取Cookie ，判断Cookie中是否有商品

Cookie[] cookies = request.getCookies();

// JSESSIONID

// 购物车 Constant.buyerCart\_NAME

if (null != cookies && cookies.length > 0) {

for (Cookie cookie : cookies) {

// 一样 这个就是购物车

if (cookie.getName().equals(Constants.BUYERCART\_NAME)) {

// 字符串类型

String value = cookie.getValue();

// 转成购物车对象

// 创建 JSON转换的对象

// 字符串转对象

buyerCart = om.readValue(value, BuyerCart.class);

break;

}

}

}

// 没有

if (null == buyerCart) {

// 第一次进来没有购物车

buyerCart = new BuyerCart();

}

// 登陆

// 判断 用户是否登陆

Buyer buyer = (Buyer) request.getSession().getAttribute(

Constants.USER\_NAME);

if (null != buyer) {

// 2.1 把当前的SkuID amount 加入Jedis中

if (null != skuId) {

// 有'

Sku sku = new Sku();

sku.setId(skuId);

// 购物项

BuyerItem buyerItem = new BuyerItem();

//多少件

buyerItem.setAmount(amount);

buyerItem.setSku(sku);

// Cotroller 购物车自己干

// 购物车中 把本次这款商品添加进来了

buyerCart.addItem(buyerItem);

}

// 3把当前用户的Redis里的商品拿出来放进购物车中 （null)

Map<String, String> hgetAll = jedis.hgetAll("buyerItem:" + buyer.getUsername());

if (null != hgetAll && hgetAll.size() > 0) {

Set<Entry<String, String>> entrySet = hgetAll.entrySet();

for (Entry<String, String> entry : entrySet) {

// 有'

Sku sku = new Sku();

sku.setId(Long.parseLong(entry.getKey()));

// 购物项

BuyerItem buyerItem = new BuyerItem();

// 多少件

buyerItem.setAmount(Integer.parseInt(entry.getValue()));

buyerItem.setSku(sku);

// Cotroller 购物车自己干

// 购物车中 把本次这款商品添加进来了

buyerCart.addItemAfter(buyerItem);

}

//清理掉原来的商品

jedis.del("buyerItem:" + buyer.getUsername());

}

// 1:判断 上面购物车中是否有商品

if (buyerCart.getItems().size() > 0) {

// 2:有：把上面的购物车中的商品放进Redis里 （当前用户的Redis里）

for (BuyerItem item : buyerCart.getItems()) {

// 当前用户的购物车在Redis里叫什么名 lpush rpush buyerCart:fbb2014 500 501

// 502

Boolean hexists = jedis.hexists(

"buyerItem:" + buyer.getUsername(), ""

+ item.getSku().getId());

if (hexists) {

// 530 数量 + amount 3

jedis.hincrBy("buyerItem:" + buyer.getUsername(), ""

+ item.getSku().getId(), item.getAmount());

} else {

// 530

jedis.hset("buyerItem:" + buyer.getUsername(), ""

+ item.getSku().getId(), "" + item.getAmount());

}

}

// 2.1 把Cookie删除

Cookie cookie = new Cookie(Constants.BUYERCART\_NAME, null);

cookie.setPath("/");

cookie.setMaxAge(0);

response.addCookie(cookie);

}

} else {

// 判断传递参数不能为Null

if (null != skuId) {

// 有'

Sku sku = new Sku();

sku.setId(skuId);

// 购物项

BuyerItem buyerItem = new BuyerItem();

// 多少件

buyerItem.setAmount(amount);

buyerItem.setSku(sku);

// Cotroller 购物车自己干

// 购物车中 把本次这款商品添加进来了

buyerCart.addItem(buyerItem);

// 字符串流 StringWriter toString

StringWriter w = new StringWriter();

om.writeValue(w, buyerCart);

// 3.4.5 更新用户浏览器的Cookie

Cookie cookie = new Cookie(Constants.BUYERCART\_NAME,

w.toString());

// 关闭浏览器要求还存在

// 1:JSESSIONID 存活 关闭浏览器就没有 Session对象失次

// 默认： -1 ：关闭浏览器就没有

// 0 : 立即，马上消毁

// 大于0 ： 秒 60\*60\*24

cookie.setMaxAge(60 \* 60 \* 24);

// 设置路径

// 默认 ：http://localhost:8080/shopping/buyerCart.shtml

// /shopping

// request.getCookies 一堆的Cookie 购物车那个Cookie /shopping/

// /buyer/\*.shtml

cookie.setPath("/");

// 更新了用户的浏览器

response.addCookie(cookie);

}

}

// 3.4.6 把当前购物车中的数据装满（在页面上显示）

List<BuyerItem> items = buyerCart.getItems();

// 表示购物车中商品

if (items.size() > 0) {

// 装数据

for (BuyerItem buyerItem : items) {

Sku sku = skuService.selectSkuById(buyerItem.getSku().getId());

// 数据加载完成

buyerItem.setSku(sku);

}

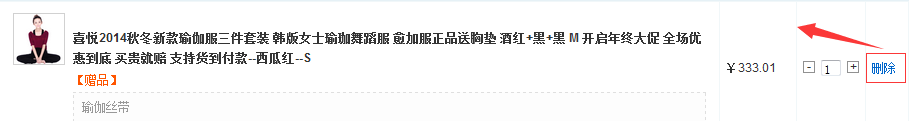
}

model.addAttribute("buyerCart", buyerCart);

return "product/cart";

}

# 购物车之删除



### 页面

<a href="javascript:void(0);" title="删除" onclick="delProduct(${item.sku.id})">删除</a>

### Js

//删除商品

function delProduct(skuId){

if(confirm("你确定要删除此商品吗?")){

window.location.href = "/shopping/delItem.shtml?skuId=" + skuId;

}

}

### 后台:删除购物项之后刷新页面重新加载购物中的数据

//删除购物车中的购物项

@RequestMapping(value = "/shopping/delItem.shtml")

public String delItem(Integer skuId,HttpServletRequest request,HttpServletResponse response) throws JsonParseException, JsonMappingException, IOException{

//购物车对象

BuyCart buyCart = null;

//把对象存储在Cookie中 需要将对象序列化成字符串

ObjectMapper om = new ObjectMapper();

//序列化的字符串中不要有Null

om.setSerializationInclusion(Inclusion.NON\_NULL);

Cookie[] cookies = request.getCookies();

if(null != cookies){

for(Cookie cookie : cookies){

if(Constants.BUYCART\_COOKIE.equals(cookie.getName())){

String value = cookie.getValue();

//System.out.println(value);

buyCart = om.readValue(value, BuyCart.class);

}

}

}

//实例化购物车 只为第一次购物

if(null != buyCart){

Sku sku = new Sku();

sku.setId(skuId);

BuyItem item = new BuyItem();

item.setSku(sku);

buyCart.removeItem(item);

//实例化一个字符串的流,存储序列化后的字符串

StringWriter str = new StringWriter();

//开始序列化

om.writeValue(str, buyCart);

//实例化Cookie,并指定Cookie的名称

Cookie cookie = new Cookie(Constants.BUYCART\_COOKIE,str.toString());

//1)给Cookie设置时间,如果不设置时间,默认下 时间为 -1 -1表示关闭浏览器会自动清理掉Cookie

cookie.setMaxAge(60\*60\*24);

cookie.setPath("/");

//将Cookie发送到浏览器

response.addCookie(cookie);

}

return "redirect:/shopping/buyCart.shtml";

}

# 购物车之+ - 按钮

### 页面

<td><a href="javascript:void(0);" title="减" class="inb arr" onclick="subProductAmount(${item.sku.id })">-</a><input type="text" class="txts" size="1" name="" value="${item.amount }" readonly="readonly" id="num${item.sku.id }"/><a href="javascript:void(0);" title="加" class="inb arr" onclick="addProductAmount(${item.sku.id },${item.sku.skuUpperLimit})">+</a></td>

### Js +

//添加商品数量 (skuid , 限购)

function addProductAmount(skuId,upperLimit){

var num = $("#num" + skuId).val();

//alert(num);

num++;

if(num >= upperLimit){

alert("此款商品不允许买" + upperLimit + "件");

return ;

}

$("#num" + skuId).val(num);

//更改购物车中的数据 更改Cookie中的数量

var url = "/shopping/addCart.shtml";

var params = {"skuId" : skuId,"buyLimit" : upperLimit,"amount" : 1};

$.post(url,params,function(data){

$("#productAmount").html(data.productAmount);

$("#productPrice").html(data.productPrice);

$("#fee").html(data.fee);

$("#totalPrice").html(data.totalPrice);

},"json");

}

### Js -

//减少商品数量

function subProductAmount(skuId,upperLimit){

var num = $("#num" + skuId).val();

num--;

if(num == 0){

delProduct(skuId);

return;

}

$("#num" + skuId).val(num);

//更改购物车中的数据 更改Cookie中的数量

var url = "/shopping/addCart.shtml";

var params = {"skuId" : skuId,"buyLimit" : upperLimit,"amount" : -1};

$.post(url,params,function(data){

$("#productAmount").html(data.productAmount);

$("#productPrice").html(data.productPrice);

$("#fee").html(data.fee);

$("#totalPrice").html(data.totalPrice);

},"json");

}

### 后台

//添加购物车中的商品数量

@RequestMapping(value = "/shopping/addCart.shtml")

public void addCart(Integer skuId,Integer amount,Integer buyLimit,HttpServletRequest request,HttpServletResponse response) throws JsonParseException, JsonMappingException, IOException{

//实例化Sku

Sku sku = new Sku();

sku.setId(skuId);

sku.setSkuUpperLimit(buyLimit);

//实例化购物项

BuyItem item = new BuyItem();

item.setSku(sku);

item.setAmount(amount);

//获取Cookie中的购物车

//购物车对象

BuyCart buyCart = null;

//把对象存储在Cookie中 需要将对象序列化成字符串

ObjectMapper om = new ObjectMapper();

//序列化的字符串中不要有Null

om.setSerializationInclusion(Inclusion.NON\_NULL);

Cookie[] cookies = request.getCookies();

if(null != cookies){

for(Cookie cookie : cookies){

if(Constants.BUYCART\_COOKIE.equals(cookie.getName())){

String value = cookie.getValue();

//System.out.println(value);

buyCart = om.readValue(value, BuyCart.class);

}

}

}

//实例化购物车 只为第一次购物

if(null == buyCart){

buyCart = new BuyCart();

}

//添加购物项进购物车

buyCart.addItem(item); //同类产品需要比较 如果SkuId一样 视为相同产品

//实例化一个字符串的流,存储序列化后的字符串

StringWriter str = new StringWriter();

//开始序列化

om.writeValue(str, buyCart);

//实例化Cookie,并指定Cookie的名称

Cookie cookie = new Cookie(Constants.BUYCART\_COOKIE,str.toString());

//1)给Cookie设置时间,如果不设置时间,默认下 时间为 -1 -1表示关闭浏览器会自动清理掉Cookie

cookie.setMaxAge(60\*60\*24);

cookie.setPath("/");

//将Cookie发送到浏览器

response.addCookie(cookie);

//将购物车里的数据装满,为了在购物车页面显示的数据全都有

//加载SkuId对应的属性

List<BuyItem> items = buyCart.getItems();

for(BuyItem it : items){

Sku skuByKey = skuService.getSkuByKey(it.getSku().getId());

it.setSku(skuByKey);

}

DecimalFormat df = new DecimalFormat(".00");

//回调数据

JSONObject jo = new JSONObject();

//商品数量

jo.put("productAmount", buyCart.getProductAmount());

//商品金额

jo.put("productPrice", df.format(buyCart.getProductPrice()));

//商品运费

jo.put("fee", df.format(buyCart.getFee()));

//商品应付金额

jo.put("totalPrice", df.format(buyCart.getTotalPrice()));

ResponseUtils.renderJson(response, jo.toString());

}

# 购物车之消空购物车

### 页面

<input type="button" value="清空购物车" title="清空购物车" class="hand btn100x26c" onclick="clearCart()" />

### Js

//消空购物车

function clearCart(){

window.location.href = "/shopping/clearCart.shtml";

}

### 后台

//消空购物车

@RequestMapping(value = "/shopping/clearCart.shtml")

public String clearCart(HttpServletRequest request, HttpServletResponse response) throws JsonParseException, JsonMappingException, IOException{

//清理Cookie

Cookie cookie = new Cookie(Constants.BUYCART\_COOKIE,null);

//设置时间,让浏览器器上的对应的Cookie马上失效 设置0马上失效

cookie.setMaxAge(0);

cookie.setPath("/");

response.addCookie(cookie);

return "redirect:/shopping/buyCart.shtml";

}

### 消空后页面显示部分

<c:if test="${fn:length(buyCart.items) == 0 }">

<div class="w ofc case">

<div class="confirm">

<div class="tl"></div><div class="tr"></div>

<div class="ofc pb40" style="text-align: center;height: 200px;margin-top: 80px">

<a href="http://localhost:8080" style="color: red;font-size: 30px;">去首页</a>挑选喜欢的商品

</div>

</div>

</div>

</c:if>

### 不消空时显示的部分

<c:if test="${fn:length(buyCart.items) != 0 }">

<div class="w ofc case">

……

</c:if>

# 购物车之继续购物

<input type="button" value="继续购物" title="继续购物" class="hand btn100x26c" onclick="window.open('/product/detail.shtml?id=${productId}')" />

# 购物车之结算按钮

### 页面

<input id="settleAccountId" type="button" value="结算" class="hand btn136x36a" onclick="trueBuy();"/>

### Js

//结算

function trueBuy(){

window.location.href = "/buyer/productOrder.shtml";

}

### 判断用户是否登陆

--此功能已经在用户登陆处已经完成了,可复用

--结算请求路径需要以/buyer/开始

### 后台

//结算

@RequestMapping(value = "/buyer/productOrder.shtml")

public String productOrder(HttpServletRequest request,ModelMap model,HttpServletResponse response) throws JsonParseException, JsonMappingException, IOException{

//1:判断登陆 返回购物车

//购物项设置进购物车中 转换对象到字符串 要求:标准的JavaBean

ObjectMapper om = new ObjectMapper();

om.setSerializationInclusion(Inclusion.NON\_NULL);

//1)应该从Cookie找到是否存在购物车

//如果存在了 从Cookie取出此购物车

//如果不存在 再创建购物车

//从request 当中找Cookie 找购物车

Cookie[] cookies = request.getCookies(); //null JSESSIONID

BuyerCart buyerCart = null;

//浏览器存多个

//现在浏览器存了 二个的 JSESSONID buyerCart\_cookie

if(null != cookies && cookies.length > 0){

for(Cookie c : cookies){

if(Constants.BUYERCART\_COOKIE.equals(c.getName())){

String value = c.getValue();

try {

buyerCart = om.readValue(value, BuyerCart.class);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

break;

}

}

}

### 判断购物车是否存在商品,如果有商品是否还有库存

////2:判断购物车是否有商品

if(null != buyerCart && buyerCart.getItems().size() >0){

//3:判断商品是否还有存在

List<BuyerItem> items = buyerCart.getItems();//5个购物项 3个

//标记 购物车购物项有多少个

Integer l = items.size();

//判断库存 > 0

for(BuyerItem item : items){

Sku sku = skuService.getSkuByKey(item.getSku().getId());

if(sku.getStockInventory() <= 0){//第一款发现没有库存 第二款也没有

//购物车移除购物项

buyerCart.deleteItem(item);

}

}

//清除没有库存商品之后的数量

Integer i = buyerCart.getItems().size();

//判断是否有库存 <0 的商品已经被清理了

if(l > i){

//对象转字符串

//创建个容器 字符串流

StringWriter sw = new StringWriter();

try {

om.writeValue(sw, buyerCart);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

//改更Cookie

Cookie cookie = new Cookie(Constants.BUYERCART\_COOKIE,sw.toString());

cookie.setMaxAge(60\*60\*24);

cookie.setPath("/");

response.addCookie(cookie);

//重定向回购物车页面

return "redirect:/shopping/buyerCart.shtml";

}else{

//跳转到订单页面

//加载默认的收货地址

// 用户Id

// 默认 是 1

AddrQuery addrQuery = new AddrQuery();

Buyer buyer = (Buyer) sessionProvider.getAttibuter(request, Constants.BUYER\_SESSION);

addrQuery.setBuyerId(buyer.getUsername());

addrQuery.setIsDef(1);

//默认地址

Addr addr = addrService.getAddrList(addrQuery).get(0);

//

model.addAttribute("addr", addr);

//把购物车装满 显示在订单页面上

//3:把购物车内容是不是显示在购物车页面上

//先加载好购物车中所有的数据 再转化成字符串 放在Cookie中

//把购物车数据加载满 再显示在购物车页面上

//购物车这个对象 但是这个对象中没有东西

for(BuyerItem item : buyerCart.getItems()){

//sku

Sku skuByKey = skuService.getSkuByKey(item.getSku().getId());

item.setSku(skuByKey);

}

model.addAttribute("buyerCart", buyerCart);

return "product/productOrder";

}

}else{

//重定向回购物车页面

return "redirect:/shopping/buyerCart.shtml";

}

# 订单页面

front\_page/product/productOrder.jsp

### 收货地址

<h2 class="h2 h2\_r mt"><em title="收货人信息">收货人信息 [<a href="#"><font style="color:blue;">修改</font></a>]</em><cite></cite></h2>

<div class="box bg\_white">

<dl class="distr">

<dd>${addr.name } ${addr.phone } <span style="margin-left: 30px">${addr.city }${addr.addr }</span></dd>

</dl>

</div>

### 商品清单

<tbody>

<c:forEach items="${buyCart.items }" var="item">

<tr>

<td>${item.sku.product.no}</td>

<td class="img48x20">

<span class="inb"><img src="${item.sku.product.img.allUrl}" /></span>

<a href="/product/detail.shtml?id=${item.sku.product.id }" target="block">${fn:substring(item.sku.product.name,0,30)}--${item.sku.color.name}--${item.sku.size}</a>

</td>

<td>￥${item.sku.skuPrice}元</td>

<td>x${item.amount}</td>

<td>${item.sku.stockInventory}</td>

</tr>

</c:forEach>

</tbody>

### 订单金额统计

<div class="confirm mt">

<div class="tl"></div><div class="tr"></div>

<div class="ofc">

<div class="r">

<dl class="total">

<dt>订单金额：<cite>(共<var id="totalNum">${buyCart.productAmount }</var>个商品)</cite></dt>

<dd><em class="l">商品金额：</em>￥<var>${buyCart.productPrice }</var></dd>

<dd><em class="l">返现：</em>￥<var>0.00</var></dd>

<dd><em class="l">运费：</em>￥<var>${buyCart.fee }</var></dd>

<dd class="orange"><em class="l">应付总额：</em>￥<var id="totalMoney">${buyCart.totalPrice }</var></dd>

<dd class="alg\_c"><input id="submitOrderID" type="submit" value="提交订单" class="hand btn136x36a"/></dd>

</dl>

</div>

</div>

</div>

# 提交订单

### 页面提交方式

<form action="/buyer/confirmOrder.shtml" method="post">

</form>

### 后台 Controller层

//提交订单

@RequestMapping(value = "/buyer/confirmOrder.shtml")

public String confirmOrder(Order order,HttpServletRequest request,HttpServletResponse response,ModelMap model) throws JsonParseException, JsonMappingException, IOException{

### 订单号

//订单号:

DateFormat df = new SimpleDateFormat("yyyyMMddHHmmssSSS");

String oid = df.format(new Date());

//设置订单号

order.setOid(oid);

### 从Cookie中取出购物车

//从Cookie中获取出购物车 购物车中有运费 应付金额 订单金额

//购物车对象

BuyCart buyCart = null;

//把对象存储在Cookie中 需要将对象序列化成字符串

ObjectMapper om = new ObjectMapper();

//序列化的字符串中不要有Null

om.setSerializationInclusion(Inclusion.NON\_NULL);

Cookie[] cookies = request.getCookies();

if(null != cookies){

for(Cookie cookie : cookies){

if(Constants.BUYCART\_COOKIE.equals(cookie.getName())){

String value = cookie.getValue();

//System.out.println(value);

buyCart = om.readValue(value, BuyCart.class);

}

}

}

### 把购物车数据装满

//将购物车里的数据装满,为保存订单及订单子项表

//加载SkuId对应的属性

List<BuyItem> items = buyCart.getItems();

for(BuyItem it : items){

Sku skuByKey = skuService.getSkuByKey(it.getSku().getId());

it.setSku(skuByKey);

}

### 订单对象Order数据装满

//从购物车中获取运费

Double fee = buyCart.getFee();

order.setDeliverFee(fee);

//订单金额 应付金额

Double totalPrice = buyCart.getTotalPrice();

order.setTotalPrice(totalPrice);

//应付金额

order.setPayableFee(totalPrice);

//支付状态

if(order.getPaymentWay().equals(0)){

//货到付款 支付方式为货到付款

order.setIsPaiy(0);

}else{

//支付方式为待付款

order.setIsPaiy(1);

}

//订单状态 提交订单

order.setState(0);

//生成时间

order.setCreateDate(new Date());

//附言

order.setNote(buyCart.getNote());

//订单的购买者

Buyer buyer = (Buyer) sessionProvider.getAttribute(request, Constants.BUYER\_NAME);

//设置订单的购买者ID

order.setBuyerId(buyer.getUsername());

### 订单子项表集合对象

//订单详情对象

List<Detail> details = new ArrayList<Detail>();

//实例化订单详情对象

Detail detail = null;

for(BuyItem item : buyCart.getItems()){

//每遍历一次实例化一个对象

detail = new Detail();

//设置SkuID

detail.setSkuId(item.getSku().getId());

//设置购买数量

detail.setAmount(item.getAmount());

//设置商品销售价

detail.setSkuPrice(item.getSku().getSkuPrice());

//设置商品ID

detail.setProductId(item.getSku().getProductId());

//添加进入订单详情集合

details.add(detail);

}

### 在订单对象中创建订单子项对象(订单详情对象)

/\*\*

\* 订单 --主

\* @author lixu

\*

\*/

public class Order implements Serializable {

/\*\*

\* 序列化ID

\*/

private static final long serialVersionUID = 1L;

private Integer id;

private String oid;

private Double deliverFee;

private Double payableFee;

private Double totalPrice;

private Integer paymentWay;

private Integer paymentCash;

private Integer delivery;

private Integer isConfirm;

private Integer isPaiy;

private Integer state;

private Date createDate;

private String note;

private String buyerId;

//添加订单详情对象集合

private List<Detail> details;

### 订单子项集合set进订单对象中

//订单集合对象设置给订单对象中

order.setDetails(details);

### 保存订单

//保存订单

orderService.addOrder(order);

### 后台Service层

public Integer addOrder(Order order) {

//保存订单

Integer o = orderDao.addOrder(order);

//准备订单详情所需要的数据

for(Detail detail : order.getDetails()){

//通过商品获取商品名称及编号

Product product = productService.getProductByKey(detail.getProductId());

//商品名称

detail.setProductName(product.getName());

//商品编号

detail.setProductNo(product.getNo());

//设置订单ID

detail.setOrderId(order.getId());

//保存订单详情

detailService.addDetail(detail);

}

return o;

}

### 后台 Dao.xml 保存订单后把订单ID返回,存储在订单对象的id属性中

<!-- 添加 -->

<insert id="addOrder" parameterType="Order" useGeneratedKeys="true" keyProperty="id">

### 消空Cookie

//消空Cookie

Cookie cookie = new Cookie(Constants.BUYCART\_COOKIE,null);

cookie.setMaxAge(0);

cookie.setPath("/");

response.addCookie(cookie);

### 订单确认及页面跳转至订单确认页面

//订单的确认信息

model.addAttribute("order", order);

//跳转至订单确认页面

return "product/confirmOrder";

# 订单确认页面

### front\_page/product/confirmOrder.jsp

<div class="confirm">

<div class="tl"></div><div class="tr"></div>

<div class="ofc">

<p class="okMsg">您的订单已成功提交，完成支付后，我们将立即发货！</p>

<table cellspacing="0" summary="" class="tab tab5">

<tr>

<th>您的订单号</th>

<td><var class="blue"><b>${order.oid }</b></var></td>

<th>应付现金</th>

<td><var class="red"><b>￥${order.payableFee }</b></var>&nbsp;元</td>

<th>支付方式</th>

<td>${order.paymentWayName }</td>

</tr>

<tr>

<th>配送方式</th>

<td>快递</td>

<th>预计到货时间</th>

<td>2014年11月20日</td>

<th></th>

<td></td>

</tr>

</table>

</div>

</div>

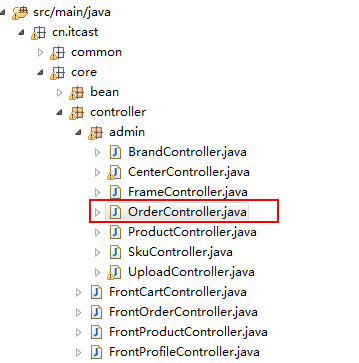
# 后台订单管理

## 订单状态流转

可查看<<订单状态流转图>>

## 准备

位置



Mybatis从数据库查询出的时间格式,可做如下处理

javaType="java.sql.Timestamp" jdbcType="TIMESTAMP"

## 订单列表

### 第一步:编写OrderController类

@Autowired

private OrderService orderService;

//订单列表

//支付状态

//订单状态 view.do

@RequestMapping(value = "/order/list.do")

public String list(Integer isPaiy ,Integer state,ModelMap model){

//创建订单Query对象

OrderQuery orderQuery = new OrderQuery();

//根据条件查询

if(null != isPaiy){

orderQuery.setIsPaiy(isPaiy);

}

//订单状态

if(null != state){

orderQuery.setState(state);

}

//

List<Order> orders = orderService.getOrderList(orderQuery);

//显示在页面

model.addAttribute("orders", orders);

return "order/list";

}

### 第二步:编写Order.java

/\*\*

\* 订单 --主

\* @author lixu

\* @Date [2014-3-28 下午04:38:53]

\*/

public class Order implements Serializable {

/\*\*

\* 序列化ID

\*/

private static final long serialVersionUID = 1L;

private Integer id;

private String oid;

private Double deliverFee;

private Double payableFee;

private Double totalPrice;

private Integer paymentWay;

private Integer paymentCash;//支付要求

private Integer delivery;

private Integer isConfirm;

private Integer isPaiy;

private Integer state;

private Date createDate;

private String note;

private String buyerId;

/\*\* 特殊处理字段 \*/

/\*\* 支付方式 \*/

public String getPaymentWayName() {

switch (paymentWay) {

case 0: return "货到到付";

case 1: return "在线支付";

case 2: return "公司转帐";

case 3: return "邮局汇款";

default: return "";

}

}

/\*\* 支付要求 必须是货到付款时 才会有支付要求 分为 0:现金 1:POS机 \*/

public String getPaymentCashName() {

if(null == paymentCash) return null;

switch (paymentCash) {

case 0: return "现金";

case 1: return "POS机";

default: return "";

}

}

/\*\* 支付状态 //支付状态 :0到付,1待付款,2已付款,3待退款,4退款成功,5退款失败 \*/

public String getIsPaiyName() {

switch (isPaiy) {

case 0: return "货到到付";

case 1: return "待付款";

case 2: return "已付款";

case 3: return "待退款";

case 4: return "退款成功";

case 5: return "退款失败";

default: return "";

}

}

/\*\* 订单状态 //订单状态 0:提交订单 1:仓库配货 2:商品出库 3:等待收货 4:完成 5:待退货 6已退货 7已取消 \*/

public String getStateName() {

switch (state) {

case 0: return "提交订单";

case 1: return "仓库配货";

case 2: return "商品出库";

case 3: return "等待收货";

case 4: return "已完成";

case 5: return "待退货";

case 6: return "已退货";

case 7: return "已取消";

default: return "";

}

}

/\*\* 送货时间 // 1:工作日，双休日，假日均可送货 2:只双休日，假日送货 3:只工作日送货（双休日，节假日不送） \*/

public String getDeliveryName() {

switch (delivery) {

case 1: return "工作日，双休日，假日均可送货";

case 2: return "只双休日，假日送货";

case 3: return "只工作日送货（双休日，节假日不送）";

default: return "";

}

}

/\*\* 电话确认 1:是 0:否 \*/

public String getIsConfirmName() {

switch (isConfirm) {

case 0: return "否";

case 1: return "是";

default: return "";

}

}

### 第三步:/back\_page/order/list.jsp

<c:forEach items="${orders }" var="order">

<tr bgcolor="#ffffff" onmouseover="this.bgColor='#eeeeee'" onmouseout="this.bgColor='#ffffff'">

<td><input type="checkbox" name="ids" value="${order.id }"/></td>

<td align="center">${order.id }--${order.oid }</td>

<td align="center">128.0</td>

<td align="center">0.0</td>

<td align="center">128.11</td>

<td align="center">${order.paymentWayName }</td>

<td align="center">${order.paymentCashName }</td>

<td align="center">${order.isPaiyName }</td>

<td align="center">${order.stateName }</td>

<td align="center">${fn:substring(order.createDate,0,19) }</td>

<td align="center"></td>

<td align="center">

<a href="view.do?id=${order.id }" class="pn-opt">查看</a>

</td>

</tr>

</c:forEach>

## 订单查看

### 第一步:编写OrderController

//查看

@RequestMapping(value = "/order/view.do")

public String view(Integer id,ModelMap model){

//1:查订单 根据Id

Order order = orderService.getOrderByKey(id);

//显示在页面

model.addAttribute("order", order);

//2:订单中用户Id 查默认收货地址

AddrQuery addrQuery = new AddrQuery();

addrQuery.setBuyerId(order.getBuyerId());

addrQuery.setIsDef(1);

Addr addr = addrService.getAddrList(addrQuery).get(0);

//显示在页面

model.addAttribute("addr", addr);

//3:订单ID 可以查订单详情

DetailQuery detailQuery = new DetailQuery();

detailQuery.setOrderId(id);

List<Detail> details = detailService.getDetailList(detailQuery);

//显示在页面

model.addAttribute("details", details);

return "order/view";

}

@Autowired

private AddrService addrService;

@Autowired

private DetailService detailService;

### 第二步:编写/back\_page/order/view.jsp

<table cellspacing="1" cellpadding="2" border="0" width="100%" class="pn-ftable">

<tbody>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">收货人:</td>

<td width="38%" colspan="1" class="pn-fcontent">${addr.name }</td>

<td width="12%" class="pn-flabel pn-flabel-h">手机:</td>

<td width="38%" colspan="1" class="pn-fcontent">${addr.phone }</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">所在地区:</td>

<td width="38%" colspan="1" class="pn-fcontent">${addr.city }</td>

<td width="12%" class="pn-flabel pn-flabel-h">地址:</td>

<td width="38%" colspan="1" class="pn-fcontent">${addr.addr }</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">订单号:</td>

<td width="38%" colspan="1" class="pn-fcontent">${order.oid }</td>

<td width="12%" class="pn-flabel pn-flabel-h">运费:</td>

<td width="38%" colspan="1" class="pn-fcontent">${order.deliverFee }元</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">订单金额:</td>

<td width="38%" colspan="1" class="pn-fcontent">${order.totalPrice }元</td>

<td width="12%" class="pn-flabel pn-flabel-h">应付金额:</td>

<td width="38%" colspan="1" class="pn-fcontent">${order.payableFee }元</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">支付方式:</td>

<td width="38%" colspan="1" class="pn-fcontent">货到到付</td>

<td width="12%" class="pn-flabel pn-flabel-h">付款要求:</td>

<td width="38%" colspan="1" class="pn-fcontent">现金</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">支付状态:</td>

<td width="38%" colspan="1" class="pn-fcontent">货到到付</td>

<td width="12%" class="pn-flabel pn-flabel-h">订单状态:</td>

<td width="38%" colspan="1" class="pn-fcontent">提交订单</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">送货时间:</td>

<td width="38%" colspan="1" class="pn-fcontent">只工作日送货（双休日，节假日不送）</td>

<td width="12%" class="pn-flabel pn-flabel-h">电话确认:</td>

<td width="38%" colspan="1" class="pn-fcontent">否</td>

</tr>

<tr>

<td width="12%" class="pn-flabel pn-flabel-h">创建时间:</td>

<td width="38%" colspan="1" class="pn-fcontent">${order.createDate }</td>

<td width="12%" class="pn-flabel pn-flabel-h">备注:</td>

<td width="38%" colspan="1" class="pn-fcontent"></td>

</tr>

</tbody>

</table>

<table cellspacing="1" cellpadding="0" border="0" width="100%" class="pn-ltable">

<thead class="pn-ltbody">

<tr>

<th>商品编号</th>

<th>商品名称</th>

<th>商品颜色</th>

<th>商品尺码</th>

<th>商品销售价</th>

<th>购买数量</th>

</tr>

</thead>

<tbody class="pn-ltbody">

<c:forEach items="${details }" var="detail">

<tr bgcolor="#ffffff" onmouseover="this.bgColor='#eeeeee'" onmouseout="this.bgColor='#ffffff'">

<td align="center">${detail.productNo}</td>

<td align="center">${detail.productName}</td>

<td align="center">${detail.color}</td>

<td align="center">${detail.size}</td>

<td align="center">${detail.skuPrice}元</td>

<td align="center">${detail.amount}件</td>

</tr>

</c:forEach>

</tbody>

</table>

# Hudson/Jenkins使用

## 第一步:新解压一个Tomcat

## 第二步:更改端口号不与当前服务器冲突

### 在tomcat/conf/server.xml更改

#### 22行

<Server port="8555" shutdown="SHUTDOWN">

#### 70行

<Connector port="8888" protocol="HTTP/1.1"

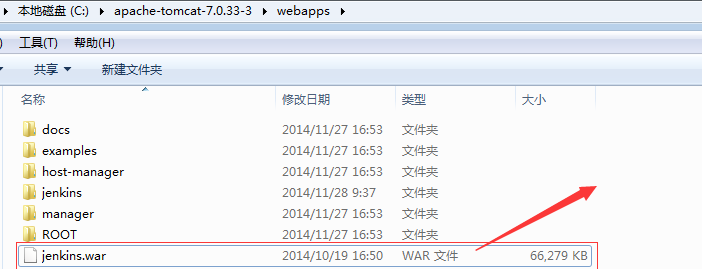
connectionTimeout="20000"

redirectPort="8443" />

#### 90行

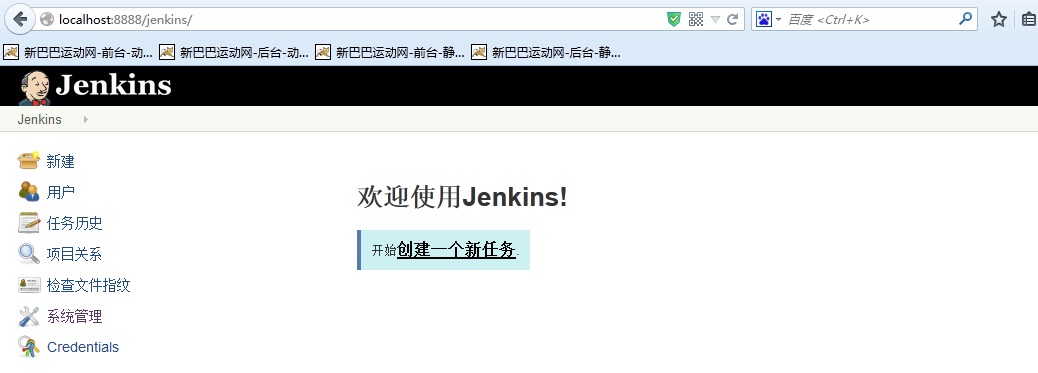
<Connector port="8999" protocol="AJP/1.3" redirectPort="8443" />

## 第三步:把hudson.war或jenkins.war放在tomcat的webapp下



## 第四步:启动Tomcat

## 第五步:访问<http://localhost:8888/jenkins>

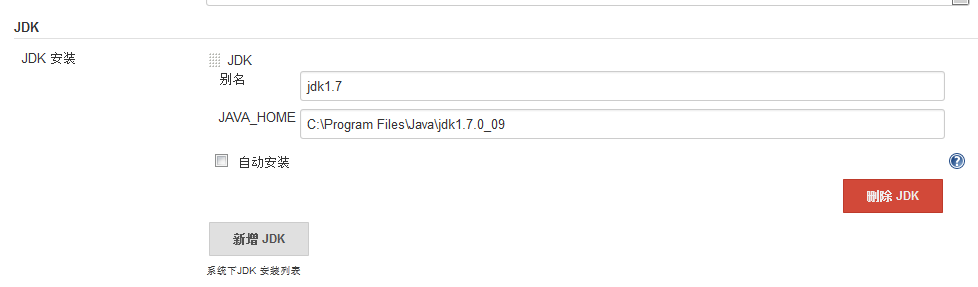


## 第六步:系统管理

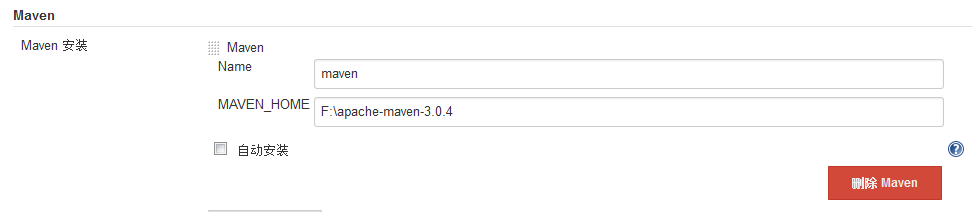


## 第七步:系统设置

### 设置Jdk



### 设置Maven

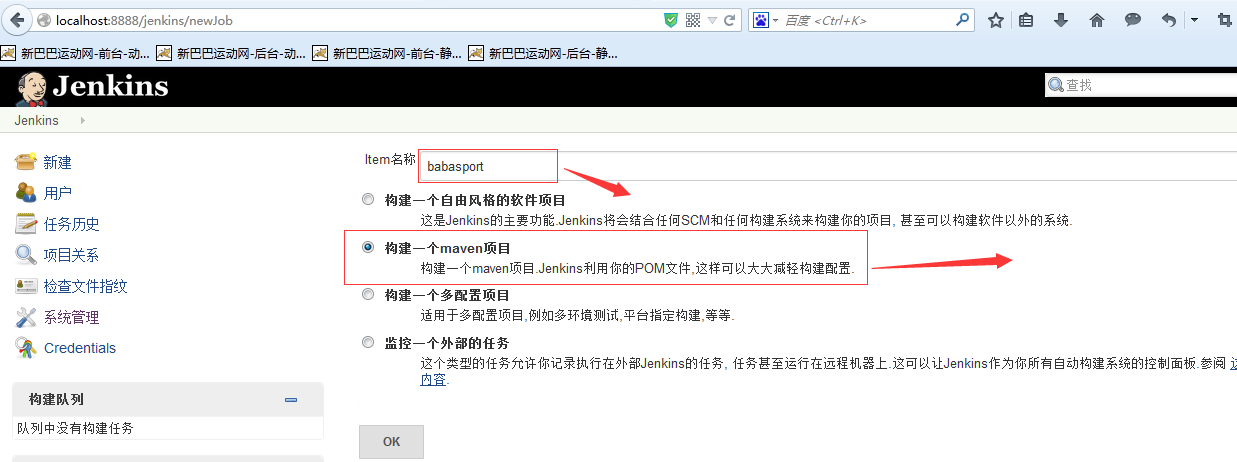


### 保存设置

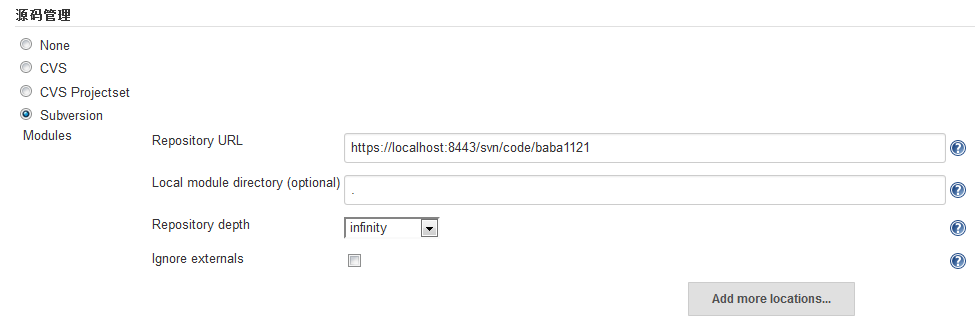
## 第八步:开始创建一个新任务



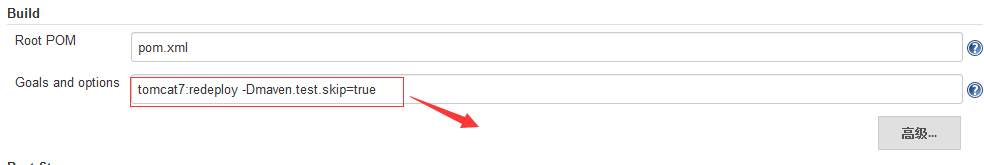
## 第九步:填写item名称,选择一个maven项目



## 第十步:设置SVN



## 第十一步:设置Maven命令



## 第十二步:保存

## 第十三步:立即构建



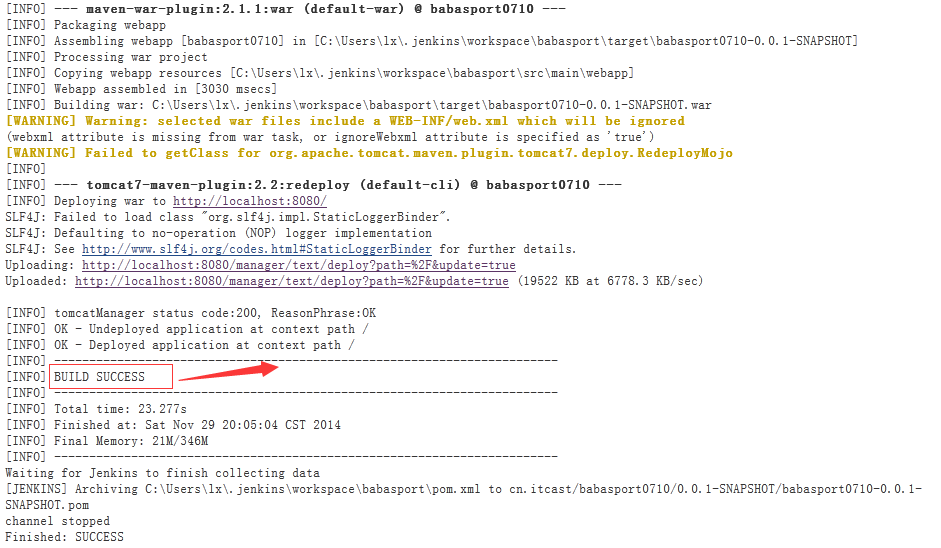
## 第十四步:查看构建历史



## 第十五步:查看控制台输出



## 第十六步:查看构建结果



# 整体网站架构分析

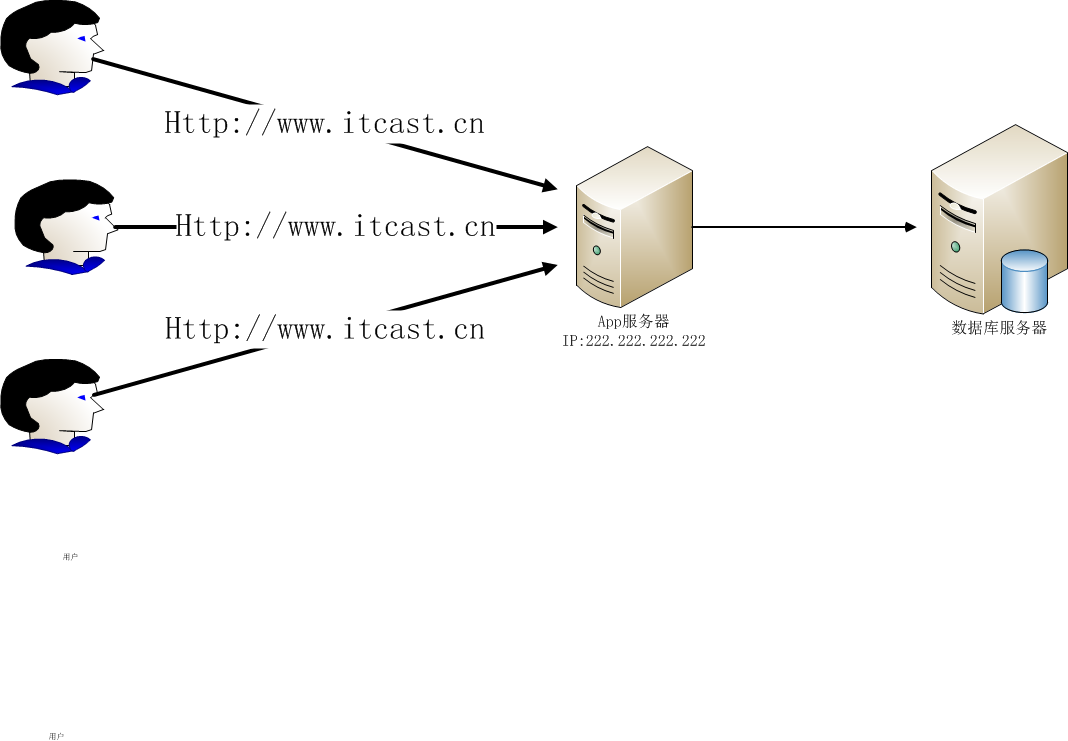
参考<<整体网站架构分析>>

# 高并发

## 什么是高并发呢？

多个进程或线程同时(或着说在同一段时间内)访问同一资源会产生并发问题。

## 高并发原理图



## 初期解决方案

### 系统或服务器级别的解决方案

1. **增大服务器的CPU。**
2. **增加内存条。**
3. **增加硬盘个数，对硬盘做Raid5。**
4. **换掉免费的Tomcat，使用商用weblogic([美国](http://baike.baidu.com/view/2398.htm)Oracle公司出品的)**
5. **增加到二块网卡。**
6. **聘请系统架构师优化Linux内核**
7. **甚至花高价直接购买高性能服务器**

随着业务的不断增加，服务器性能很快又到达瓶颈

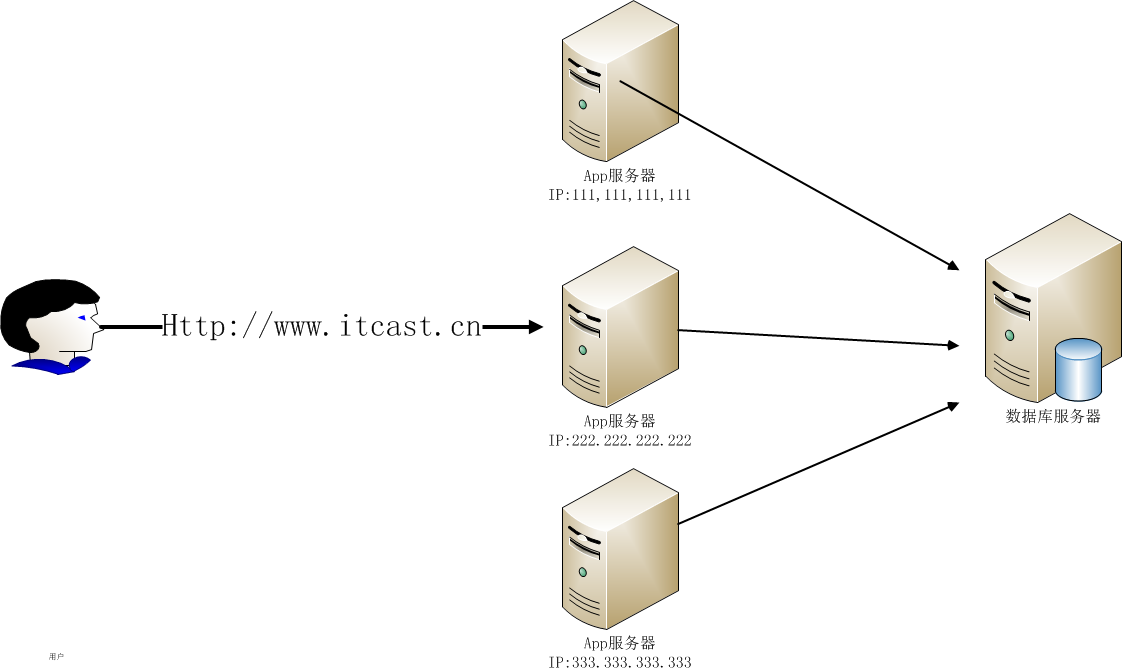
## 应用级别的解决方案

1. **网页HTML 静态化（需要CMS项目支持）**
2. **图片服务器分离（常用解决方案）**
3. **缓存（常用解决方案） 上上策为分布式缓存**
4. **镜像（下载较多）**

随着业务的不断增加，服务器性能很快又到达瓶颈

## 能否增加服务器数量？

### 解决办法：增加App服务器



## 随之出现更多问题

**问题1：用户访问IP多了 怎么解决？**

**问题2：数据库出现瓶颈 怎么办？**

## 解决用户IP多了的办法是?

### 开始:准备采用DNS

#### 什么是DNS

DNS（Domain Name System，域名系统），因特网上作为域名和[IP地址](http://baike.baidu.com/view/3930.htm)相互映射的一个[分布式数据库](http://baike.baidu.com/view/68389.htm)，能够使用户更方便的访问[互联网](http://baike.baidu.com/view/6825.htm)，而不用去记住能够被机器直接读取的IP数串。通过[主机](http://baike.baidu.com/view/23880.htm)名，最终得到该主机名对应的IP地址的过程叫做域名解析（或主机名解析）。DNS协议运行在UDP协议之上，使用端口号53。

#### 解决IP多—DNS

DNS服务器可以解决IP多了的问题

[http://www.itcast.cn](http://www.itcast.cn/) : 192.168.1.100

192.168.1.101

192.168.1.102

……更多

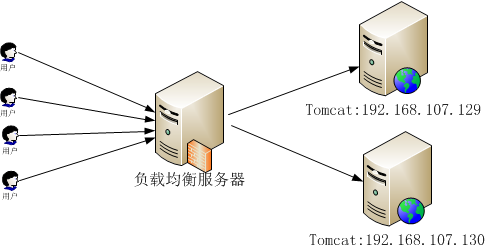
缺点：虽然循环复用 DNS 是一个普遍使用的在 Web 服务器上负载平衡的解决方案，但是，该方式有它自身的缺陷。循环复用 DNS将传入的 IP 请求映射到定义的一系列循环形式的服务器。一旦发生服务器故障，循环复用 DNS 继续把请求发送到这个故障服务器，一直到把该服务器从 DNS 中移走为止。这样许多用户必须等到 DNS 连接超时以后才能成功地访问目标网站

### 终级解决方案

#### 采用负载均衡技术

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）。

#### 负载均衡工作原理分析



#### 总结:负载均衡功能

1:转发请求

2:故障移除

3:恢复添加

### 负载均衡种类

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.

### 负载均衡--软件解决

**主流软件**

1: apache + JK

2: nginx

3: lvs + keepalived

#### 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。

#### 为什么选择LVS？ 优势

　1、抗负载能力强，因为lvs工作方式的逻辑是非常之简单，而且工作在网络4层仅做请求分发之用，没有流量，所以在效率上基本不需要太过考虑。在我手里的 lvs，仅仅出过一次问题：在并发最高的一小段时间内均衡器出现丢包现象，据分析为网络问题，即网卡或linux2.4内核的承载能力已到上限，内存和 cpu方面基本无消耗。

　2、配置性低，这通常是一大劣势，但同时也是一大优势，因为没有太多可配置的选项，所以除了增减服务器，并不需要经常去触碰它，大大减少了人为出错的几率。

　3、工作稳定，因为其本身抗负载能力很强，所以稳定性高也是顺理成章，另外各种lvs都有完整的双机热备方案，所以一点不用担心均衡器本身会出什么问题，节点出现故障的话，lvs会自动判别，所以系统整体是非常稳定的。

　4、无流量，上面已经有所提及了。lvs仅仅分发请求，而流量并不从它本身出去，所以可以利用它这点来做一些线路分流之用。没有流量同时也保住了均衡器的IO性能不会受到大流量的影响。

　5、基本上能支持所有应用，因为lvs工作在4层，所以它可以对几乎所有应用做负载均衡，包括http、数据库、聊天室等等。

#### LVS 对比 Nginx

1：负载度 LVS KO Nginx

2：功能多少 Nginx KO LVS

3：稳定度 LVS KO Nginx

4：服务器性能要求 LVS KO Nginx

### 效率最高的负载均衡技术

调度器的实现技术中,IP负载均衡技术是效率最高的,IP虚拟服务器软件(IPVS)是在linux内核中实现的｡

#### IPVS软件实现了三种IP负载均衡技术

1：VS/NAT

2： VS/TUN

3： VS/DR

#### LVS中提供了八种不同的调度算法

1:轮叫调度（Round-Robin Scheduling）  
2: 加权轮叫调度（Weighted Round-Robin Scheduling）  
3:最小连接调度（Least-Connection Scheduling）  
4:加权最小连接调度（Weighted Least-Connection Scheduling）  
5:基于局部性的最少链接（Locality-Based Least Connections Scheduling）  
6:带复制的基于局部性最少链接（Locality-Based Least Connections with Replication Scheduling）  
7:目标地址散列调度（Destination Hashing Scheduling）  
8:源地址散列调度（Source Hashing Scheduling）  
9:最短预期延时调度（Shortest Expected Delay Scheduling）  
10:不排队调度（Never Queue Scheduling）  
 对应: rr|wrr|lc|wlc|lblc|lblcr|dh|sh|sed|nq

# Nginx搭建负载均衡环境

## 环境准备

* 系统：Centos6（四台）
* 负载均衡：Nginx + Keepalived （主）
* 负载均衡：Nginx + Keepalived （备）
* 服务器1：Tomcat
* 服务器2：Tomcat

### 关闭防火墙

[root@itcast01 ~]# service iptables status

[root@itcast01 ~]# service iptables stop

[root@itcast01 ~]# chkconfig iptables --list

[root@itcast01 ~]# chkconfig iptables off

## 配置二台Tomcat服务器

此处同学们应该是没有问题的，故不再详细说明

## 配置负载均衡主机

### 安装Nginx反向代理服务器

D:\讲课内容--\新巴巴运动网\nginx高并发解决\nginx安装包



上传以上rpm软件包

Rpm –U openssl-1.0.1e-…..

Rpm –ivh nginx-1.8.0-1…..

### 配置Ngixn的配置文件

根据上边的需求在nginx.conf文件中配置负载均衡，如下：

http{

upstream tomcat\_server\_pool{

server 192.168.101.5:8080 weight=10;

server 192.168.101.6:8080 weight=10;

}

server {

listen 80;

server\_name aaa.test.com;

location / {

proxy\_pass http://tomcat\_server\_pool;

index index.jsp index.html index.htm;

}

}

}

### 测试

请求Nginx所在机器的IP看看能不能转发二台Tomcat服务器

### 配置Nginx的高可用

Nginx通过Keepalived实现高可用

### Keepalived的工作原理

keepalived是以VRRP协议为实现基础的，VRRP全称Virtual Router Redundancy Protocol，即虚拟路由冗余协议。

虚拟路由冗余协议，可以认为是实现路由器高可用的协议，即将N台提供相同功能的路由器组成一个路由器组，这个组里面有一个master和多个backup，master上面有一个对外提供服务的vip（该路由器所在局域网内其他机器的默认路由为该vip），master会发组播，当backup收不到VRRP包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master。这样的话就可以保证路由器的高可用了。

keepalived主要有三个模块，分别是core、check和VRRP。core模块为keepalived的核心，负责主进程的启动、维护以及全局配置文件的加载和解析。check负责健康检查，包括常见的各种检查方式。VRRP模块是来实现VRRP协议的。

### 安装Keepalived

D:\讲课内容--\新巴巴运动网\nginx高并发解决\高可用\keepalived\安装包



上传以上软件

rpm –ivh keepalived….

### 修改主nginx下/etc/keepalived/keepalived.conf文件

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from XXX@XXX.com #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_instance VI\_1 {

state MASTER #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 100 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

### 备nginx

修改备nginx下/etc/keepalived/keepalived.conf文件

**配置备nginx时需要注意：需要修改state为BACKUP , priority比MASTER低，virtual\_router\_id和master的值一致**

! Configuration File for keepalived

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from XXX@XXX.com #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_instance VI\_1 {

state BACKUP #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 99 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

### 测试

主备nginx都启动keepalived及nginx。

### 解决nginx进程和keepalived不同时存在问题

### 问题描述

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。

### nginx进程检测脚本

在主nginx上需要编写nginx进程检测脚本（check\_nginx.sh），判断nginx进程是否存在，如果nginx不存在就将keepalived进程杀掉，check\_nginx.sh内容如下：

#!/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

将check\_nginx.sh拷贝至/etc/keepalived下，

给此脚本赋予权限

chmod 700 check\_nginx.sh

脚本测试：

将nginx停止，将keepalived启动，执行脚本：sh /etc/keepalived/check\_nginx.sh



从执行可以看出自动将keepalived进程kill掉了。

### 修改keepalived.conf

修改主nginx的keepalived.conf，添加脚本定义检测：

注意下边红色标识地方：

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from miaoruntu@itcast.cn #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_script check\_nginx {

script "/etc/keepalived/check\_nginx.sh" ##监控脚本

interval 2 ##时间间隔，2秒

weight 2 ##权重

}

vrrp\_instance VI\_1 {

state MASTER #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 100 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

track\_script {

check\_nginx #监控脚本

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

修改后重启keepalived

### 测试

回到负载均衡高可用的初始状态，保证主、备上的keepalived、nginx全部启动。

停止主nginx服务

观察keepalived日志：

tail -f /var/log/keepalived.log



查看keepalived进程已经不存在。

查看eth0已经没有绑定vip。

# LVS搭建负载均衡的环境

### 准备

* 系统：Centos6 (三台)
* 负载均衡：LVS + keepalived
* 服务器1：Http
* 服务器2：Http

### ip配置

1、VIP（virtual ip）：用来提供virtual server服务的ip地址。分别绑定在Director一个物理网卡上（对外接收请求包）和RS的回环设备上（回环设备需要绑定两个ip，一个是127.0.0.1，另一个就是vip）。

2、DIP（director ip）：与vip绑定在一个物理网卡上，用来转发请求包到RS的RIP对应的mac上，此设备可以通过arp请求获取RIP对应的mac地址。

3、RIP（real server ip）：绑定在RS上的一个物理网卡上，用来接收从Directory转发过来的请求包。

### 关闭防火墙

[root@itcast01 ~]# service iptables status

[root@itcast01 ~]# service iptables stop

[root@itcast01 ~]# chkconfig iptables --list

[root@itcast01 ~]# chkconfig iptables off

### 第一步:在二台真实服务器网上上添加LO:0接口,并设置虚拟IP在此接口上

[root@itcast01 ~]# cd /etc/init.d/

[root@itcast01 ~]#touch realserver

[root@itcast01 ~]#vim realserver

SNS\_VIP=192.168.8.150

. /etc/rc.d/init.d/functions

case "$1" in

start)

ifconfig lo:0 $SNS\_VIP netmask 255.255.255.255 broadcast $SNS\_VIP

/sbin/route add -host $SNS\_VIP dev lo:0

echo "1" >/proc/sys/net/ipv4/conf/lo/arp\_ignore

echo "2" >/proc/sys/net/ipv4/conf/lo/arp\_announce

echo "1" >/proc/sys/net/ipv4/conf/all/arp\_ignore

echo "2" >/proc/sys/net/ipv4/conf/all/arp\_announce

sysctl -p >/dev/null 2>&1

echo "RealServer Start OK"

;;

stop)

ifconfig lo:0 down

route del $SNS\_VIP >/dev/null 2>&1

echo "0" >/proc/sys/net/ipv4/conf/lo/arp\_ignore

echo "0" >/proc/sys/net/ipv4/conf/lo/arp\_announce

echo "0" >/proc/sys/net/ipv4/conf/all/arp\_ignore

echo "0" >/proc/sys/net/ipv4/conf/all/arp\_announce

echo "RealServer Stoped"

;;

\*)

echo "Usage: $0 {start|stop}"

exit 1

esac

exit 0

以上参数说明

SNS\_VIP=192.168.1.98 #定义VIP变量

. /etc/rc.d/init.d/functions #导脚本库

case "$1" in #case语句 $1传递给该shell脚本的第一个参数

start)

ifconfig lo:0 $SNS\_VIP netmask 255.255.255.255 broadcast $SNS\_VIP #设置Lo：0 VIP netmask 及广播

/sbin/route add -host $SNS\_VIP dev lo:0 ##route del 增加本地路由

echo "1" >/proc/sys/net/ipv4/conf/lo/arp\_ignore

echo "2" >/proc/sys/net/ipv4/conf/lo/arp\_announce

echo "1" >/proc/sys/net/ipv4/conf/all/arp\_ignore

echo "2" >/proc/sys/net/ipv4/conf/all/arp\_announce

sysctl -p >/dev/null 2>&1 # -p <file> (default /etc/sysctl.conf) 将标准信息输入设备空文件

echo "RealServer Start OK"

;;

stop)

ifconfig lo:0 down

route del $SNS\_VIP >/dev/null 2>&1 #route del 删除本地路由

echo "0" >/proc/sys/net/ipv4/conf/lo/arp\_ignore

echo "0" >/proc/sys/net/ipv4/conf/lo/arp\_announce

echo "0" >/proc/sys/net/ipv4/conf/all/arp\_ignore

echo "0" >/proc/sys/net/ipv4/conf/all/arp\_announce

echo "RealServer Stoped"

;;

\*)

echo "Usage: $0 {start|stop}" #$0 是脚本本身的名字

exit 1 #表示进程正常退出

esac #case结束

exit 0 #表示进程非正常退出

[root@itcast01 ~]#chmod 750 realserver #给此脚本赋予权限

[root@itcast01 ~]#service realserver start #开启此脚本

RealServer Start OK #表示启动成功!

[root@itcast01 ~]#ifconfig #查看一下是否真的成功添加了lo:0接口

### 第二步:开启二台真实服务器上自带的Apache公司的Http服务器

[root@itcast01 ~]#service httpd start #开启Http服务器

[root@itcast01 ~]#cd /var/www/html/ #进入Http服务器默认的目录

[root@itcast01 ~]#touch index.html #创建Http服务器默认访问页面

[root@itcast01 ~]#vim index.html #编辑Http服务器默认访问页面

I am http IP : 真实服务器自己的IP(例如:192.168.200.201)

用浏览器访问一下:http://192.168.200.201 默认端口80

查看结果是否是自己编辑的: I am http IP : 真实服务器自己的IP(例如:192.168.200.201)

二台真实服务器一样

### 第三步:开始搭建负载均衡

1:安装 ipvsadm keepalived

命令:yum –y install ipvsadm keepalived

2:配置 keepalived

命令:vim /etc/keepalived/keepalived.conf

global\_defs {

# notification\_email {

# admin@toxingwang.com

# }

# notification\_email\_from master@toxingwang.com

# smtp\_server smtp.exmail.qq.com

# smtp\_connect\_timeout 30

router\_id LVS\_DEVEL

}

vrrp\_instance VI\_1 {

state MASTER

interface eth0

virtual\_router\_id 51

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.200.201

}

}

virtual\_server 192.168.200.201 80 {

delay\_loop 6

lb\_algo wrr

lb\_kind DR

nat\_mask 255.255.255.0

persistence\_timeout 0

protocol TCP

real\_server 192.168.200.139 80 {

weight 3

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

real\_server 192.168.200.140 80 {

weight 3

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

}

以上参数说明

global\_defs { ##全局配置部分

# notification\_email { ##下面几行均为全局通知配置，可以实现出现问题后报警，但功能有限，因此注释掉，并采用Nagios监视lvs运行情况

# admin@toxingwang.com

# }

# notification\_email\_from master@toxingwang.com

# smtp\_server smtp.exmail.qq.com

# smtp\_connect\_timeout 30

router\_id LVS\_DEVEL ##设置lvs的id，在一个网络内应该是唯一的

}

vrrp\_instance VI\_1 { ##设置vrrp组，唯一且同一LVS服务器组要相同

state MASTER ##备份LVS服务器设置为BACKUP

interface eth0 # #设置对外服务的接口

virtual\_router\_id 51 ##设置虚拟路由标识

priority 100 #设置优先级，数值越大，优先级越高，backup设置为99，这样就能实现当master宕机后自动将backup变为master，而当原master恢复正常时，则现在的master再次变为backup。

advert\_int 1 ##设置同步时间间隔

authentication { ##设置验证类型和密码，master和buckup一定要设置一样

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress { ##设置VIP，可以多个，每个占一行

192.168.18.60

}

}

virtual\_server 192.168.18.60 80 {

delay\_loop 6 ##健康检查时间间隔，单位s

lb\_algo wrr ##负载均衡调度算法设置为加权轮叫

lb\_kind DR ##负载均衡转发规则

nat\_mask 255.255.255.0 ##网络掩码，DR模式要保障真实服务器和lvs在同一网段

persistence\_timeout 50 ##会话保持时间，单位s

protocol TCP ##协议

real\_server 192.168.18.61 80 { ##真实服务器配置，80表示端口

weight 3 ##权重

TCP\_CHECK { ##服务器检测方式设置 keepalived的健康检查方式 有：HTTP\_GET|SSL\_GET|TCP\_CHECK|SMTP\_CHECK|MISC\_CHECK

connect\_timeout 0 ##连接超时时间

nb\_get\_retry 3 ##失败重试次数

delay\_before\_retry 3 ##失败重试的间隔时间

connect\_port 80 ##连接的后端端口

}

}

real\_server 192.168.18.62 80 {

weight 3

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

}

#### 启动keepalived服务

[root@lvs1 ~]# service keepalived start

**注：**由于keepalived配置文件有语法错误也能启动，因此看到启动了lvs服务，不代表配置文件没有错误，如果遇到lvs不能正常转发，及时跟踪日志进行处理。

#### 日志跟踪方法

1、开两个ssh窗口连接到lvs服务器，第一个窗口运行如下命令：

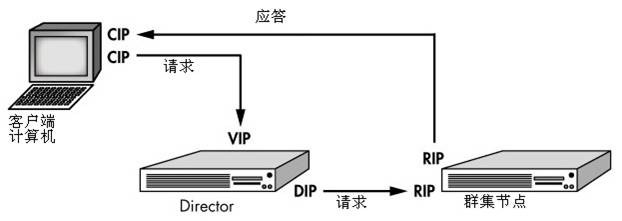
[root@lvs1 ~]# tail -f /var/log/message

2、第二个窗口重新启动keepalived服务，同时观察窗口1中日志的变化，然后根据日志提示解决即可。

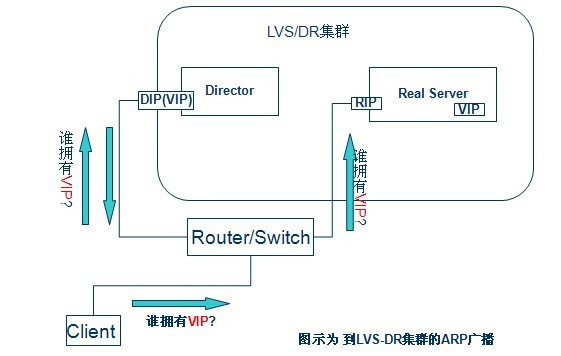
#### 浏览器访问虚拟IP查看结果

二台真实服务器分别出现在页面上,表明结果是正确的

## 内部原理图



## 防止丢数据包原理图



# 高可用

高可用性H.A.（High Availability）指的是通过尽量缩短因日常维护操作（计划）和突发的系统崩溃（非计划）所导致的停机时间，以提高系统和应用的可用性。它与被认为是不间断操作的[容错技术](http://baike.baidu.com/view/391720.htm" \t "_blank)有所不同。HA系统是目前企业防止核心计算机系统因故障停机的最有效手段。

一：网络高可用

由于[网络存储](http://baike.baidu.com/view/600264.htm" \t "_blank)的快速发展，[网络冗余](http://baike.baidu.com/view/983061.htm" \t "_blank)技术被不断提升，提高IT系统的高可用性的关键应用就是网络高可用性，网络高可用性与网络高可靠性是有区别的，网络高可用性是通过匹配[冗余](http://baike.baidu.com/view/104445.htm" \t "_blank)的[网络设备](http://baike.baidu.com/view/1158081.htm" \t "_blank)实现网络设备的冗余，达到高可用的目的。

比如冗余的[交换机](http://baike.baidu.com/view/1077.htm" \t "_blank)，冗余的[路由器](http://baike.baidu.com/view/1360.htm" \t "_blank)等

二：服务器高可用

服务器高可用主要使用的是[服务器集群](http://baike.baidu.com/view/292866.htm" \t "_blank)软件或高可用软件来实现。

三：存储高可用

使用软件或硬件技术实现存储的高度可用性。其主要技术指标是存储切换功能，数据复制功能，数据[快照](http://baike.baidu.com/view/16510.htm" \t "_blank)功能等。当一台[存储](http://baike.baidu.com/view/87682.htm" \t "_blank)出现故障时，另一台备用的存储可以快速切换，达一存储不停机的目的。

## LVS备机搭建

global\_defs {

# notification\_email {

# admin@toxingwang.com

# }

# notification\_email\_from master@toxingwang.com

# smtp\_server smtp.exmail.qq.com

# smtp\_connect\_timeout 30

router\_id LVS\_DEVEL

}

vrrp\_instance VI\_1 {

state BACKUP

interface eth0

virtual\_router\_id 51

priority 99

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.200.201

}

}

virtual\_server 192.168.200.201 80 {

delay\_loop 6

lb\_algo wrr

lb\_kind DR

nat\_mask 255.255.255.0

persistence\_timeout 0

protocol TCP

real\_server 192.168.200.139 80 {

weight 3

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

real\_server 192.168.200.140 80 {

weight 3

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

}

## 演示备机操作

1：Master挂了，无备机，演示

2：Master挂了，备机自动接管

3：Master好了，备机进入待机状态

# 大数据量

## 场景：

### 我说的大数据量处理是指同时需要对数据进行检索查询，同时有高并发的增删改操作；

## 大数据量的处理

例如腾讯，盛大，动辄数以亿计的帐号,怎么能这么快呢， 于是找到了互联网现在对数据处理的发展：  
  
对于大数据量处理，如果是互联网处理的话，一般分为下面阶段：  
  
第一阶段，所有数据都装入一个数据库，当数据量大了肯定就会出现问题，就像刚刚说的查询，于是想办法  
  
第二阶段，那时肯定想做缓存机制，确实可以，如加上缓存Memcached，但缓存也是治标不治本，数据量太大了也是不行，于是  
  
第三阶段，master-slave模式，进行主从数据库，master提供写，slave进行读，这个适合于有写造成数据库卡的方法，还是不行，于是  
  
第四阶段，垂直分库，这个意义还是不大，于是  
  
第五阶段，进行水平分库，这个不错，记得以前从兴也是按这个分时间水平分库，其实可以分的更细点估计效果更好

## 问题2：数据库出现瓶颈 怎么办？

以Mysql为例：

1：对Mysql进行优化（重点讲解）

2：缓存,主流缓存Memcached，redis…

3: mysql读写分离 + 主从复制

4：Oracle

5：Oracle读写分离 + 主从复制

6：Oracle Partition 分区

7：Oracle RAC集群（终级解决方案）

此方案：非常贵，即使是淘宝，京东这样的大公司，也是很难受的。

### 主要方案:水平拆分,库表散列

待续…

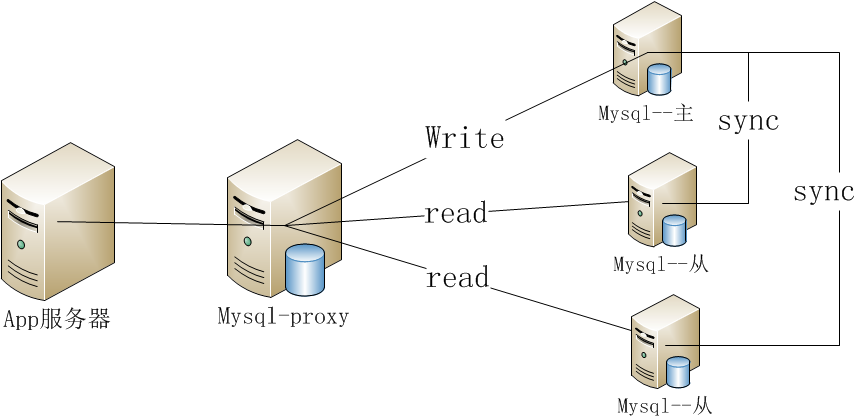
### 备选方案:Mysql主从复制与读写分离

MySQL主从复制(Master-Slave)与读写分离(MySQL-Proxy)实践

Mysql作为目前世界上使用最广泛的免费数据库，相信所有从事系统运维的工程师都一定接触过。但在实际的生产环境中，由单台Mysql作为独立的数据库是完全不能满足实际需求的，无论是在安全性，高可用性以及高并发等各个方面。

因此，一般来说都是通过 主从复制（Master-Slave）的方式来同步数据，再通过读写分离（MySQL-Proxy）来提升数据库的并发负载能力 这样的方案来进行部署与实施的。

### 原理架构图



## 开始搭建主从复制

服务器二台：

分别安装二台Mysql数据库

1：安装命令

yum –y install mysql-server

2:配置登陆用户的密码

演示此操作

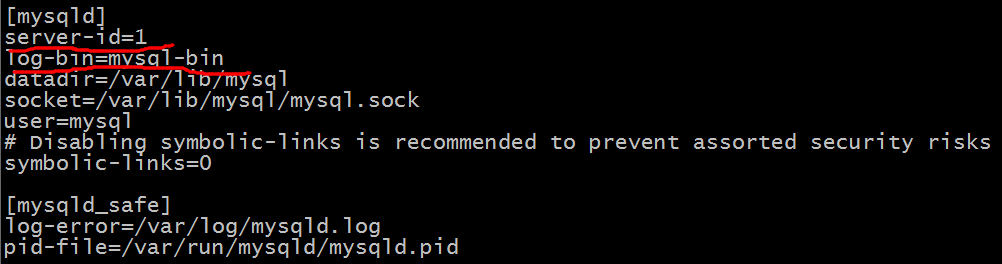
3：配置允许第三方机器访问本机Mysql

演示此操作

场景描述：

主数据库服务器：192.168.1.112，MySQL已经安装，并且无应用数据。  
从数据库服务器：192.168.1.115，MySQL已经安装，并且无应用数据。

#### Mysql配置

1. vim /etc/my.cnf
2. 
3. 接下来确认slave和master的上的server\_id是否正确。可以分别在slave和master上运行 SHOW VARIABLES LIKE 'server\_id'; 来查看server\_id是否和你配置的一样。
4. 分别重新启动master,slaver的二台mysql服务
5. 登陆
6. 输入
7. Mysql> SHOW VARIABLES LIKE 'server\_id';
8. 来查看server\_id是否和你配置的一样。
9. master输入
10. Mysql> show master status;  
    +------------------+----------+--------------+------------------+  
    | File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |  
    +------------------+----------+--------------+------------------+  
    | mysql-bin.000001 | 106 | | |  
    +------------------+----------+--------------+------------------+
11. 记录下 FILE 及 Position 的值，在后面进行从服务器操作的时候需要用到。
12. **配置从服务器**

change master to

master\_host='192.168.0.104',

master\_user='root',

master\_password='root',

master\_log\_file='mysql-bin.000001',

master\_log\_pos=106;

1. 正确执行后启动Slave同步进程
2. 启动slave

mysql> start slave;

1. 查看slave状态

mysql> show slave status\G

1. 其中Slave\_IO\_Running 与 Slave\_SQL\_Running 的值都必须为YES，才表明状态正常。

#### 测试主从复制

**1:先确定主，从库上没有任何自定义表**

**2：主服务器上的操作**  
在主服务器上创建数据库itcast\_db  
mysql> create database itcast\_db;

在主服务器上创建表itcast\_tb  
mysql> create table itcast\_tb(id int(3),name char(10));

在主服务器上的表itcast\_tb中插入记录  
mysql> insert into itcast\_tb values (01, "itcast01");

**3:从服务器上查看是否已经同步？**

## 总结:搭建过程中可能引起失败的原因

1:server\_id 配置的一样或是配置的没有更新到Mysql数据中来

2:防火墙拦截了3306端口

3:用户与密码不正确

4:Mysql不允许其它机器访问

## 把对象存放到Redis服务器中

/\*\*

\* 测试Redis jedis

\* @author lx

\*

\*/

public class TestJedis extends SpringJunitTest {

@Autowired

private JedisConnectionFactory jedisConnectionFactory;

@Test

public void testJedis() throws Exception {

//获取Jedis

Jedis redis = jedisConnectionFactory.getShardInfo().createResource();

Brand brand = new Brand();

brand.setName("中文");

List<Brand> brands = new ArrayList<Brand>();

brands.add(brand);

Pagination p = new Pagination(1,10,20);

p.setList(brands);

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ObjectOutputStream oos = new ObjectOutputStream(baos);

oos.writeObject(p);

redis.set("cnbrand".getBytes(), baos.toByteArray());

byte[] sss = redis.get("cnbrand".getBytes());

ByteArrayInputStream ba = new ByteArrayInputStream(sss);

ObjectInputStream ois = new ObjectInputStream(ba);

Pagination pagination = (Pagination) ois.readObject();

List<Brand> brs = (List<Brand>) pagination.getList();

for (Brand brand2 : brs) {

System.out.println(brand2);

}

//设置过期时间 单位是秒

redis.setex("mapbrand1".getBytes(), 15, baos.toByteArray());

//删除

//redis.del("mapbrand1");

Boolean exists = redis.exists("mapbrand1");

System.out.println(exists);

}

# 手动配置Spring切面增,删,改,查,操作分布式缓存Redis

## 在redis.xml中配置如下

### 配置内容

<!-- 切面对象 -->

<bean id="cacheInteceptror" class="cn.itcast.common.web.aop.CacheInteceptor"></bean>

<!-- 手动配置切面 环绕切面 -->

<aop:config>

<!-- 切面 -->

<aop:aspect ref="cacheInteceptror">

<!-- 环绕方式 -->

<aop:around method="doAround" pointcut="execution(\* cn.itcast.core.service.product.BrandServiceImpl.select\*(..))"/>

<!-- 方法后 -->

<aop:after method="doAfter" pointcut="execution(\* cn.itcast.core.service.product.BrandServiceImpl.add\*(..))"/>

<aop:after method="doAfter" pointcut="execution(\* cn.itcast.core.service.product.BrandServiceImpl.delete\*(..))"/>

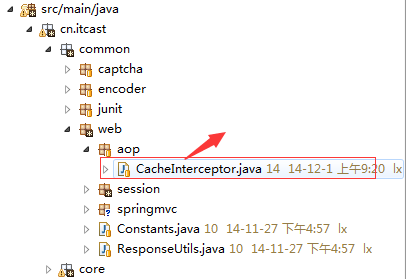
<aop:after method="doAfter" pointcut="execution(\* cn.itcast.core.service.product.BrandServiceImpl.update\*(..))"/>

</aop:aspect>

</aop:config>

## 创建CacheInterceptor切面对象

### 位置



### 内容

#### 准备

/\*\*

\* 基于Spring aop 的 切面对象

\* 处理切面有三种

\* after

\* before

\* around

\* @author lx

\*

\*/

public class CacheInterceptor{

@Autowired

private JedisConnectionFactory jedisConnectionFactory; //缓存默认超时时间

private static final int TIMEOUT = 3600000;

#### 生成唯一的Key策略

//生成Key 例如:cn.itcast.core.service.BrandServiceImpl….

public String getCacheKey(ProceedingJoinPoint pjp){

StringBuilder cacheKey = new StringBuilder();

//包类+类名 cn.itcast.core.dao.BrandServiceImpl .selectBrandById

String packageAndClassName = pjp.getTarget().getClass().getName();

cacheKey.append(packageAndClassName);

//方法名

String methodName = pjp.getSignature().getName();

cacheKey.append(".").append(methodName);

//参数[] 参数对象的转成JSON串

ObjectMapper om = new ObjectMapper();

om.setSerializationInclusion(Inclusion.NON\_NULL);

Object[] args = pjp.getArgs();

for (Object object : args) {

StringWriter str = new StringWriter();

try {

om.writeValue(str, object);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

cacheKey.append(".").append(str.toString());

}

//生成的Key

return cacheKey.toString();

}

#### 查询时切面方法

//查询切面 环绕切面

public Object doAround(ProceedingJoinPoint pjp) throws Throwable{

//判断 Redis当中是否有缓存数据

Jedis jedis = jedisConnectionFactory.getShardInfo().createResource();

//List 分页对象 Brand对象

//从Jedis中取值 要求 Key生成的策略 包名+类名 + 方法 名 + 参数对象的转成JSON串

String cacheKey = getCacheKey(pjp);

byte[] bs = jedis.get(cacheKey.getBytes());

if(null == bs){

//从Service中取 返回Service 分页对象 List Brand

Object proceed = pjp.proceed();

//放到Redis中一份

//二进制流

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ObjectOutputStream oos = new ObjectOutputStream(baos);

//写入流

oos.writeObject(proceed);

//二进制流

jedis.set(cacheKey.getBytes(), baos.toByteArray());

//

bs = jedis.get(cacheKey.getBytes());

}

//返回Controller层

//转成品牌对象

ByteArrayInputStream bais = new ByteArrayInputStream(bs);

ObjectInputStream ois = new ObjectInputStream(bais);

return ois.readObject();

}

#### 变更时切面方法

//方法后

public void doAfter(JoinPoint jp){

//1:获取当前访问的包名+类名

String packageAndClassName = jp.getTarget().getClass().getName();

//2:去redis 服务器 查包名+类名开始的 所有的Key

//判断 Redis当中是否有缓存数据

Jedis jedis = jedisConnectionFactory.getShardInfo().createResource();

//所有符合的KeyS

Set<String> keys = jedis.keys(packageAndClassName + "\*");

for (String key : keys) {

//删除此Key及Value

jedis.del(key);

}

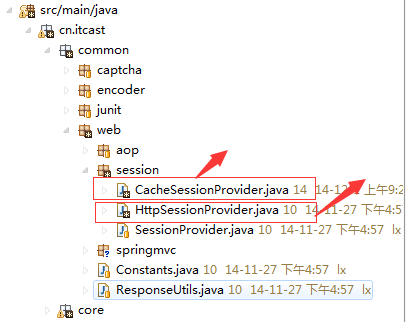
}

#### 测试

1. 测试Controller层访问Service层时,是否能进入切面对象
2. 第一次查询时,通过打断点,可看到进入Service查询数据库数据,第二次查询时,通过打断点,可看到不再进入Service,而是去缓存服务器查数据
3. 数据变更时,在Service方法后清除此数据库表对应的Service层所有查询时缓存下来的数据

# 共享Session解决

## 位置



## CacheSessionProvider提供类的内容

### 注入对象

/\*\*

\* 远程Session

\*

\* @author lx

\*

\*/

public class CacheSessionProvider implements SessionProvider {

@Autowired

private JedisConnectionFactory jedisConnectionFactory;

//设置一个Session超时时间

private int exp = 30;

public void setExp(int exp) {

this.exp = exp;

}

### 往Session中设置值

// 往Session

@SuppressWarnings("unchecked")

@Override

public void setAttribute(HttpServletRequest request,

HttpServletResponse response, String name, Serializable value)

throws Exception {

//获取Jedis

Jedis jedis = jedisConnectionFactory.getShardInfo().createResource();

//从Redis中获取Session对象的二进制数组

byte[] bs = jedis.get(getSessionId(request, response).getBytes());

//实例化一个Session

Map<String, Serializable> session = null;

if(null != bs){

// 转成品牌对象

ByteArrayInputStream bais = new ByteArrayInputStream(bs);

ObjectInputStream ois = new ObjectInputStream(bais);

//从Redis中取出来的

session = (Map<String, Serializable>) ois.readObject();

//之前存储的 code\_session : code验证码

session.put(name, value);

}else{

session = new HashMap<String,Serializable>();

//上前存储了 buyer\_session : 用户对象

session.put(name, value);

}

// 二进制流

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ObjectOutputStream oos;

// 写入流

oos = new ObjectOutputStream(baos);

oos.writeObject(session);

// Redis 1M JSESSIONID Session 30分钟

jedis.setex(getSessionId(request, response).getBytes(), 60 \* exp,

baos.toByteArray());

}

### 获取Session中的值

@Override

@SuppressWarnings("unchecked")

public Serializable getAttribute(HttpServletRequest request,

HttpServletResponse response, String name) throws Exception {

Jedis jedis = jedisConnectionFactory.getShardInfo().createResource();

// TODO Auto-generated method stub

// 在本地Session中存储值 1M

byte[] bs = jedis.get(getSessionId(request, response).getBytes());

if(null != bs){

// 转成品牌对象

ByteArrayInputStream bais = new ByteArrayInputStream(bs);

ObjectInputStream ois = new ObjectInputStream(bais);

//从Redis中取出来的

Map<String, Serializable> session = (Map<String, Serializable>) ois.readObject();

//判断 是否为NUll

if(null != session){

//之前存储的 code\_session : code验证码

return session.get(name);

}

}

return null;

}

### 退出登陆

@Override

public void logout(HttpServletRequest request, HttpServletResponse response) {

// TODO Auto-generated method stub

//判断此Key在redis 当中有没有 getSessionId(request,response)

Jedis jedis = jedisConnectionFactory.getShardInfo().createResource();

if(jedis.exists(getSessionId(request,response))){

//删除

jedis.del(getSessionId(request,response));

}

}

### 获取SessionId

@Override

public String getSessionId(HttpServletRequest request,

HttpServletResponse response) {

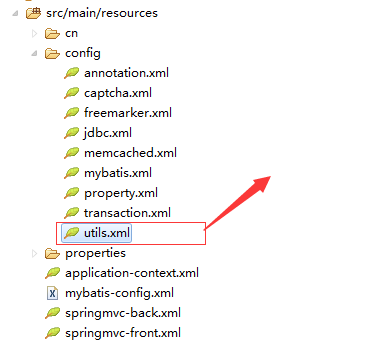
// TODO Auto-generated method stub

return request.getSession().getId();

}

## 把CacheSessionProvider交给Spring实例类

### 位置



### 内容

<bean id="sessionProvider" class="cn.itcast.common.web.session.CacheSessionProvider">

<property name="exp" value="60"/>

</bean>

### 测试

# 项目总结