

网上商城实战篇

今日内容介绍

- ◆ 分类管理：查询所有分类
- ◆ 商品管理

今日内容学习目标

- ◆ JavaWeb 知识巩固

第 1 章 查询所有分类

1.1 分析



1.2 准备工作

- 创建数据库表、JavaBean、Dao 接口和实现类、Service 接口和实现类
- 步骤 1：创建分类表

```
-- 2.1 创建分类表
CREATE TABLE `category` (
  `cid` varchar(32) NOT NULL,
  `cname` varchar(20) DEFAULT NULL, #分类名称
  PRIMARY KEY (`cid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- 2.2 初始化分类默认数据
INSERT INTO `category` VALUES ('1','手机数码'), ('172934bd636d485c98fd2d3d9cccd409','运动户外'), ('2','电脑办公'), ('3','家具家居')
```

```
''), ('4', '鞋靴箱包'), ('5', '图书音像'), ('59f56ba6ccb84cb591c66298766b83b5', 'aaaa'), ('6', '母婴孕婴'), ('afdba41a139b4320a74904485bdb7719', '汽车用品');
```

- 步骤 2: 创建 JavaBean Category

```
└─ cn.itcast.store.domain
   └─ Category.java
   └─ User.java
```

```
public class Category {

    private String cid;
    private String cname;
```

- 步骤 3: 编写 dao 接口、及实现类

```
/**
 * 分类模块 dao 层的接口
 *
 */
public interface CategoryDao {

}

/**
 * 分类模块 dao 层的实现类
 *
 */
public class CategoryDaoImpl implements CategoryDao {

}
```

- 步骤 4: 编写 service 即可及实现类

```
/**
 * 分类模块 service 层的接口
 *
 */
public interface CategoryService {

}

package cn.itcast.store.service.impl;

/**
 * 分类模块 service 层的接口
```

```
*/  
  
public class CategoryServiceImpl implements CategoryService {  
  
}
```

1.3 代码实现

- 步骤 1: 完善 IndexServlet，显示/jsp/index.jsp 前查询分类

```
public class IndexServlet extends BaseServlet {  
    private static final long serialVersionUID = 1L;  
  
    public String execute(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // 查询所有的分类  
        CategoryService categoryService = new CategoryServiceImpl();  
        List<Category> allCategory = categoryService.findAll();  
        // 将查询结果存放 request 作用域  
        request.setAttribute("allCategory", allCategory);  
  
        return "/jsp/index.jsp";  
    }  
  
}
```

- 步骤 2: 完善 CategoryService，提供 findAll()方法

```
//接口  
public interface CategoryService {  
    /**  
     * 查询所有  
     * @return  
     */  
    List<Category> findAll();  
}  
  
//实现类  
public class CategoryServiceImpl implements CategoryService {  
  
    private CategoryDao categoryDao = new CategoryDaoImpl();  
    @Override  
    public List<Category> findAll() {  
        return categoryDao.findAll();  
    }  
}
```

```
}
```

● 步骤 3：完善 CategoryDao，提供 findAll()方法

```
//接口
/**
 * 分类模块 dao 层的接口
 *
 */
public interface CategoryDao {

    /**
     * 查询所有
     * @return
     */
    List<Category> findAll() throws SQLException;

}

//实现类
/**
 * 分类模块 dao 层的实现类
 *
 */
public class CategoryDaoImpl implements CategoryDao {

    @Override
    public List<Category> findAll() throws SQLException{
        QueryRunner queryRunner = new QueryRunner(JDBCUtils.getDataSource());
        String sql = "select * from category";
        return queryRunner.query(sql,
            new BeanListHandler<Category>(Category.class));
    }

}
```

● 步骤 4：遍历数据

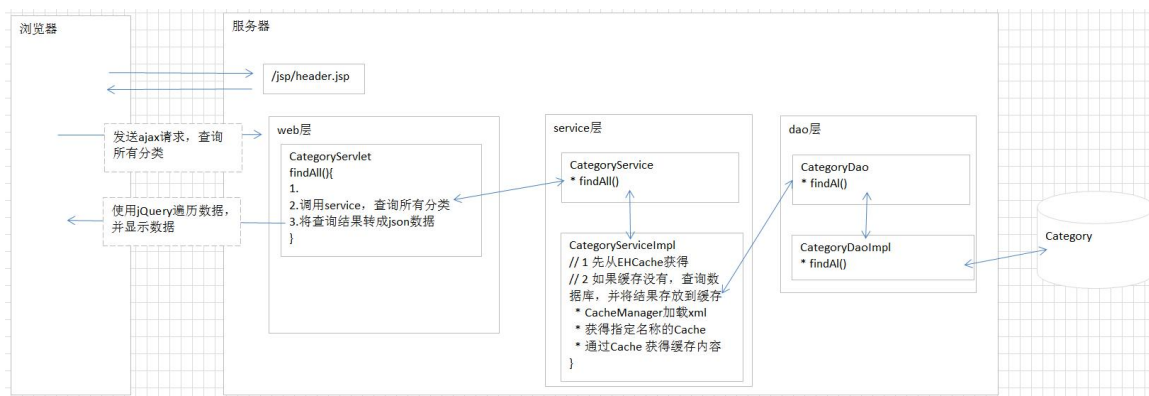
```
header.jsp
50< class="nav navbar-nav">
51 <!--激活 <li class='active'> --%>
52< <c:forEach items="${allCategory}" var="category">
53     <li value="${category.cid}"><a href="#">${category.cname}</a></li>
54 </c:forEach>
55 <!--激活 <li class='active'> --%>
    <c:forEach items="${allCategory}" var="category">
```

```
<li value="${category.cid}"><a href="#">${category.cname}</a></li>
</c:forEach>
```

1.4 优化：Ajax 异步加载

1.4.1 分析

当访问首页时可以显示分类导航条，但访问其他模块时无法访问显示分类。通过比较程序我们发现，显示首页前我们查询了所有分类，显示登录等其他模块时我们没有查询分类。为了所有模块都可以显示分类，我们需要发送 ajax 单独查询分类。



1.4.2 代码实现

- 步骤 1：修改 IndexServlet，将查询分类代码注释
- 步骤 2：修改 header.jsp 给添加 id，并注释查询所有的遍历内容

```
<ul class="nav navbar-nav" id="menu">
  <!--激活<li class='active'>
  <c:forEach items="${allCategory}" var="category">
    <li value="${category.cid}"><a href="#">${category.cname}</a></li>
  </c:forEach>
  <!--%>
</ul>
```

- 步骤 3：修改 header.jsp 添加 js 函数，页面加载发送 ajax 查询所有分类

```
<script type="text/javascript">
<!--
    ${function(){
      var url = "${pageContext.request.contextPath}/CategoryServlet";
      $.post(url,{"method":"findAll"},function(data){
        $.each(data,function(i,n){
          $("#menu").append("<li value='><a href='#>"+n.cname+"</a></li>");
        });
      });
    });
```



```
    });  
    });  
    //-->  
</script>
```

- 步骤 4：编写 CategoryServlet，提供 findAll()方法

```
public class CategoryServlet extends BaseServlet {  
    private static final long serialVersionUID = 1L;  
  
    //ajax 查询所有  
    public String findAll(HttpServletRequest request, HttpServletResponse response) throws Exception {  
        // 1.1 查询所有的分类  
        CategoryService categoryService = new CategoryServiceImpl();  
        List<Category> allCategory = categoryService.findAll();  
        // 1.2 将查询结果转换成 json  
        String jsonStr = JSONArray.fromObject(allCategory).toString();  
        // 1.3 将结果响应给浏览器  
        response.setContentType("application/json;charset=UTF-8");  
        response.getWriter().print(jsonStr);  
  
        return null;  
    }  
}
```

- 步骤 5：web.xml servlet 的配置

```
<servlet>  
    <servlet-name>CategoryServlet</servlet-name>  
    <servlet-class>cn.itcast.store.web.servlet.CategoryServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>CategoryServlet</servlet-name>  
    <url-pattern>/CategoryServlet</url-pattern>  
</servlet-mapping>
```

1.5增强：缓存技术


1.5.1 分析

当我们在不同模块之间切换时，发现菜单栏显示的分类数据都是一样的。浏览器每发送一次请求，服务器端都会查询一次数据库，从而对数据库服务器造成不必要的访问。实际开发中，我们采

用缓存技术来解决此类问题。

1.5.2 相关技术

- 缓存 (Cache): 通常指的就是内存中的一块空间, 介于应用程序和永久性数据存储源(如硬盘上的文件或者数据库)之间, 其作用是降低应用程序直接读写永久性数据存储源的频率, 从而提高应用的运行性能。
- 常见的缓存技术
 - redis 是一个 key-value 存储系统。和 Memcached 类似, 它支持存储的 value 类型相对更多。
 - Memcached 是一个高性能的分布式内存对象缓存系统, 用于动态 Web 应用以减轻数据库负载。
 - EhCache 是一个纯 Java 的进程内缓存框架, 具有快速、精干等特点。
- 导入 jar 包:



commons-pool2-2.3.jar
jedis-2.7.0.jar

1.5.3 代码实现

- 如果缓存中已经有, 将直接从缓存获得, 如果没有将从数据库获取。修改 CategoryService 代码, 给当前查询所有添加业务缓存。

```
/**
 * 查询分类通过缓存服务器
 */
public String findAllByAjax() throws Exception {

    Jedis j =null;
    String value=null;
    try {
        //从 redis 获取分类信息
        //1.获取连接
        j = JedisUtils.getJedis();

        //2.获取数据 判断数据是否为空
        value = j.get("category_list");

        //2.1 若不为空,直接返回数据
        if (value!=null) {
            System.out.println("缓存中有数据库");
        }
    }
}
```



```
        return value;
    }

    //2.2 若为空,从mysql 数据库中获取 并放入 redis 中
    List<Category> clist = findAll();
    //将 clist 转成 json 返回且放入 redis 中即可
    value=JsonUtil.list2json(clist);

    //将 value 放入 redis 中
    j.set("category_list", value);
    return value;
} finally {
    //释放 jedis
    JedisUtils.close(j);
}
}
```

1.5.4 Redis 工具类

```
public class JedisUtils {
    //创建连接池
    private static JedisPoolConfig config;
    private static JedisPool pool;

    static{
        config=new JedisPoolConfig();
        config.setMaxTotal(30);
        config.setMaxIdle(2);

        pool=new JedisPool(config, "192.168.17.132", 6379);
    }

    //获取连接的方法
    public static Jedis getJedis(){
        return pool.getResource();
    }

    //释放资源
    public static void close(Jedis j) {
        if(j != null){
            j.close();
        }
    }
}
```



```
}  
}  
}
```

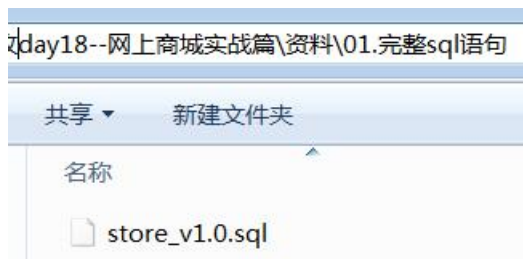
第2章 前台商品管理

2.1 准备工作

- 步骤 1：创建表并完善数据

```
-- 3.1 创建商品表  
CREATE TABLE `product` (  
  `pid` varchar(32) NOT NULL,  
  `pname` varchar(50) DEFAULT NULL,      #商品名称  
  `market_price` double DEFAULT NULL,    #商场价  
  `shop_price` double DEFAULT NULL,      #商城价  
  `pimage` varchar(200) DEFAULT NULL,    #商品图片路径  
  `pdate` date DEFAULT NULL,             #上架时间  
  `is_hot` int(11) DEFAULT NULL,         #是否热卖  
  `pdesc` varchar(255) DEFAULT NULL,     #商品描述  
  `pflag` int(11) DEFAULT 0,             #商品标记: 0=未下架(默认值), 1=已经下架  
  `cid` varchar(32) DEFAULT NULL,        #分类 id  
  PRIMARY KEY (`pid`),  
  KEY `product_fk_0001` (`cid`),  
  CONSTRAINT `product_fk_0001` FOREIGN KEY (`cid`) REFERENCES `category` (`cid`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
-- 3.2 初始化商品默认数据  
INSERT INTO `product` VALUES ('1','小米 4c 标准版',1399,1299,'products/1/c_0001.jpg','2015-11-02',1,'小米 4c 标准版 全网通 白色 移动联通电信 4G 手机 双卡双待','0','1'),('10','华为 Ascend Mate7',2699,2599,'products/1/c_0010.jpg','2015-11-02',1,'华为 Ascend Mate7 月光银 移动4G 手机 双卡双待双通 6 英寸高清大屏, 纤薄机身, 智能超八核, 按压式指纹识别!!选择下方“移动老用户 4G 飞享合约”, 无需换号, 还有话费每月返还!','0','1')
```

参考:





- 步骤 2：编写 JavaBean Product

```
public class Product {  
  
    private String pid;  
    private String pname;  
    private Double market_price;  
  
    private Double shop_price;  
    private String pimage;  
    private Date pdate;  
  
    private Integer is_hot; // 0 不是热门 1: 热门  
    private String pdesc;  
    private Integer pflag; // 0 未下架 1: 已经下架  
  
    // 分类，以面向对象的方式描述商品与分类之间的关系  
    // * 多个商品属于一个分类  
    private Category category;
```

- 步骤 3：编写 dao 接口，及实现类

```
/**  
 * 商品 dao 接口  
 */  
public interface ProductDao {  
  
}  
/**  
 * 商品 dao 实现类  
 */  
public class ProductDaoImpl implements ProductDao {  
  
}
```

- 步骤 4：编写 service 接口，及实现类

```
/**  
 * 商品 service 接口  
 */  
public interface ProductService {  
  
}  
/**  
 * 商品 service 实现类  
 */  
public class ProductServiceImpl implements ProductService {
```

```
}
```

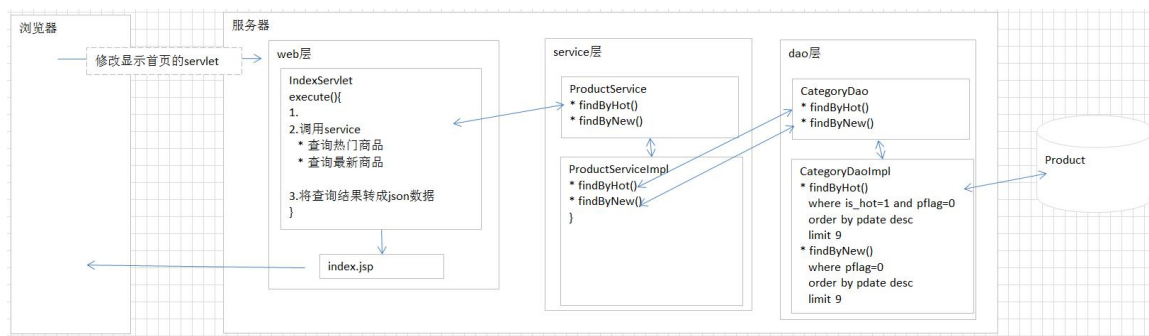
● 步骤 5: 编写 servlet

```
public class ProductService extends BaseServlet {  
    private static final long serialVersionUID = 1L;  
  
}
```

```
<servlet>  
    <servlet-name>ProductServlet</servlet-name>  
    <servlet-class>cn.itcast.store.web.servlet.ProductServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>ProductServlet</servlet-name>  
    <url-pattern>/ProductServlet</url-pattern>  
</servlet-mapping>
```

2.2 首页热门商品和最新商品的显示

2.2.1 分析



2.2.2 代码实现

● 步骤 1: 修改 IndexServlet, 添加查询热门和最新商品的查询

```
public class IndexServlet extends BaseServlet {  
    private static final long serialVersionUID = 1L;  
  
    public String execute (HttpServletRequest request, HttpServletResponse response)  
        throws Exception {
```



```
        ProductService productService = new ProductServiceImpl();

        // 1.1 查询热门商品:
        List<Product> hotList = productService.findByHot();
        // 1.2 查询最新商品:
        List<Product> newList = productService.findByNew();

        // 2 将查询结果存放
        request.setAttribute("hotList", hotList);
        request.setAttribute("newList", newList);

        return "/jsp/index.jsp";
    }

}
```

● 步骤 2: 修改 service, 提供 findByHot() 和 findByNew() 方法

```
// 接口
public interface ProductService {

    /**
     * 热门商品
     * @return
     */
    List<Product> findByHot() throws SQLException;

    /**
     * 最新商品
     * @return
     */
    List<Product> findByNew() throws SQLException;

}
```

```
// 实现类
public class ProductServiceImpl implements ProductService {

    private ProductDao productdao = new ProductDaoImpl();

    @Override
    public List<Product> findByHot() throws SQLException {
        return productdao.findByHot();
    }

}
```

```
@Override  
public List<Product> findByNew() throws SQLException{  
    return productdao.findByNew();  
}  
  
}
```

● 步骤 3: 修改 dao, 提供 findByHot()和 findByNew()方法

```
// 接口  
public interface ProductDao {  
    /**  
     * 热门商品  
     * @return  
     * @throws SQLException  
     */  
    List<Product> findByHot() throws SQLException;  
  
    /**  
     * 最新商品  
     * @return  
     * @throws SQLException  
     */  
    List<Product> findByNew() throws SQLException;  
}
```

```
//实现类  
public class ProductDaoImpl implements ProductDao {  
    @Override  
    public List<Product> findByHot() throws SQLException {  
        QueryRunner queryRunner = new QueryRunner(JDBCUtils.getDataSource());  
        String sql = "select * from product where is_hot=? and pflag = ? order by pdate desc limit ?";  
        List<Product> list = queryRunner.query(sql, new BeanListHandler<Product>(Product.class), 1,0,9);  
        return list;  
    }  
  
    @Override  
    public List<Product> findByNew() throws SQLException {  
        QueryRunner queryRunner = new QueryRunner(JDBCUtils.getDataSource());  
        String sql = "select * from product where pflag = ? order by pdate desc limit ?";  
        List<Product> list = queryRunner.query(sql, new BeanListHandler<Product>(Product.class),0,9);  
        return list;  
    }  
}
```

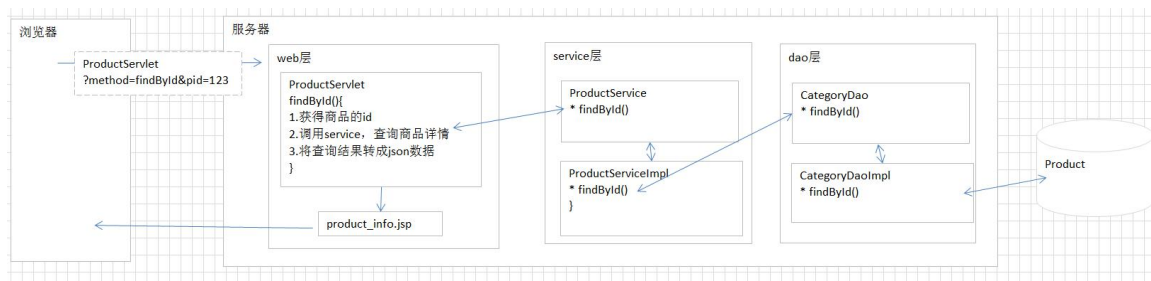
● 步骤 4：修改/jsp/index.jsp，页面显示

```
<%--热门商品列表 --%>
<c:forEach var="p" items="${ hotList }">
<div class="col-md-2" style="text-align:center;height:200px;padding:10px 0px;">
    <a href="#">
        
    </a>
    <p><a href="#" style='color:#666'>${ p.pname }</a></p>
    <p><font color="#E4393C" style="font-size:16px">&yen;${ p.shop_price }</font></p>
</div>
</c:forEach>
```

```
<%--最新商品列表 --%>
<c:forEach var="p" items="${ newList }">
<div class="col-md-2" style="text-align:center;height:200px;padding:10px 0px;">
    <a href="#">
        
    </a>
    <p><a href="#" style='color:#666'>${ p.pname }</a></p>
    <p><font color="#E4393C" style="font-size:16px">&yen;${ p.shop_price }</font></p>
</div>
</c:forEach>
```

2.3 商品详情

2.3.1 分析



2.3.2 代码实现

- 步骤 1: 修改/jsp/index.jsp，点击图片或标题可以查询商品详情

```
index.jsp
84 <!--热门商品列表-->
85 <c:forEach var="p" items="${ hotList }">
86 <div class="col-md-2" style="text-align:center;height:200px;padding:10px 0px;">
87 <a href="${ pageContext.request.contextPath }/ProductServlet?method=findById&pid=${ p.pid }">
88 
90 <p><a href="${ pageContext.request.contextPath }/ProductServlet?method=findById&pid=${ p.pid }" style='
91 <p><font color="#E4393C" style="font-size:16px">&yen;${ p.shop_price }</font></p>
92 </div>
93 </c:forEach>
```

- 步骤 2: 修改 ProductServlet，添加 findById 方法

```
public class ProductServlet extends BaseServlet {
    private static final long serialVersionUID = 1L;

    //通过 id 查询详情
    public String findById(HttpServletRequest request, HttpServletResponse response)
        throws Exception {

        // 1 接收参数:
        String pid = request.getParameter("pid");

        // 2 通知 service 进行查询
        ProductService productService = new ProductServiceImpl();
        Product product = productService.findById(pid);
        // 3.1 将查询结果存放作用域
        request.setAttribute("product",product);
        // 3.2 页面跳转
        return "/jsp/product_info.jsp";
    }
}
```

- 步骤 3: 修改 ProductService，添加 findById()方法

```
//接口
/**
 * 通过 id 查询详情
 * @param pid
 * @return
 */
Product findById(String pid) throws SQLException;
```

```
//实现类
@Override
public Product findById(String pid) throws SQLException {
```

```
return productDao.findById(pid);  
}
```

● 步骤 4：修改 ProductDao，添加 findById()方法

```
//接口  
/**  
 * 通过 id 查询详情  
 * @param pid  
 * @return  
 */  
Product findById(String pid) throws SQLException;
```

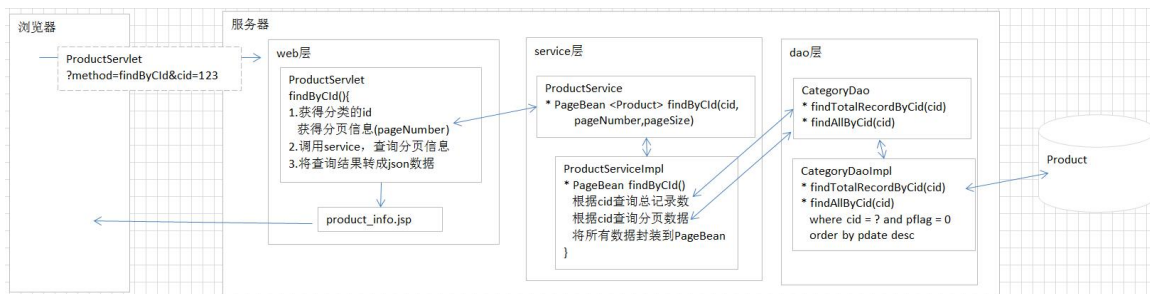
```
//实现类  
@Override  
public Product findById(String pid) throws SQLException {  
    QueryRunner queryRunner = new QueryRunner(JDBCUtils.getDataSource());  
    String sql = "select * from product where pid = ?";  
    Product product = queryRunner.query(sql, new BeanHandler<Product>(Product.class), pid);  
    return product;  
}
```

● 步骤 5：修改 product_info.jsp，显示信息

```
product_info.jsp  
43= <div style="margin:0 auto;width:950px;">  
44= <div class="col-md-6">  
45=   
46= </div>  
47= <div class="col-md-6">  
48= <div><strong>${ product.pname }</strong></div>  
49= <div style="border-bottom: 1px dotted #ddd; width: 350px; margin: 10px 0 10px 0;">  
50= <div>编号: ${ product.pid }</div>  
51= </div>  
52= <div style="margin: 10px 0 10px 0;">亿家价: <strong style="color: #ef0101;">${ product.shop_price }</strong> 参考价: <del>${ product.market_price }</del>  
53= </div>  
54= <div style="margin: 10px 0 10px 0;">促销: <a target="_blank" title="限时抢购 (2014-07-30 ~ 2015-01-01)" style="background-color: #f07373;">限时抢购</a> </div>  
55= <div style="padding: 10px; border: 1px solid #e7dbb1; width: 330px; margin: 15px 0 10px 0; background-color: #fffef6;">  
56= <div style="margin: 5px 0 10px 0;">白色</div>  
57= <div style="border-bottom: 1px solid #faeac7; margin-top: 20px; padding-left: 10px;">购买数量:  
58= <input id="quantity" name="quantity" value="1" maxlength="4" size="10" type="text"> </div>  
59= <div style="margin: 20px 0 10px 0; text-align: center;">  
60= <a href="${pageContext.request.contextPath}/jsp/cart.jsp">  
61= <input style="background: url('${pageContext.request.contextPath}/img/product.gif') no-repeat scroll 0 -600px rgba(0, 0, 0, 0); height: 36px; width: 1  
62= </a> &nbsp;收藏商品</div>  
63= </div>  
64= </div>  
65= <div class="clear"></div>  
66= <div style="width: 950px; margin: 0 auto;">  
67= <div style="background-color: #d3d3d3; width: 930px; padding: 10px 10px; margin: 10px 0 10px 0;">  
68= <strong>商品介绍</strong>  
69= </div>  
70= <div>  
71= <div>  
72= <div>  
73= <div style="width: 950px; margin: 0 auto;">  
74= <div style="background-color: #d3d3d3; width: 930px; padding: 10px 10px; margin: 10px 0 10px 0;">  
75= <strong>商品介绍</strong>  
76= </div>  
77= <div>  
78= <div>  
79= <div>  
80= <div>  
81= </div>
```


2.4 查询分类商品(含分页)

2.4.1 分析



2.4.2 代码实现

- 步骤 1: 修改/jsp/header.jsp 显示分类列表的 js 代码

```
//支持分页，如果没有传递默认值 1
$("#menu").append("<li value=''><a
href='${pageContext.request.contextPath}/ProductServlet?method=findByCid&cid="+n.cid+"'">"+n.cname+"</a></li>");
```

- 步骤 2: 修改 ProductServlet，添加 findByCid 方法

```
//通过分类 id 查询所有商品
public String findByCid(HttpServletRequest request, HttpServletResponse response)
    throws Exception {

    // 1 接收参数
    // 1.1 分类 id
    String cid = request.getParameter("cid");

    // 1.2 当前页
    int pageNumber = 1;
    try {
        //没有传递参数，或参数错误，使用默认值 1
        pageNumber = Integer.parseInt(request.getParameter("pageNumber"));
    } catch (Exception e) {
    }

    // 1.3 每页显示个数
    int pageSize = 12; //固定值，可以通过请求参数获得
```



```
// 2 调用业务层:
ProductService productService = new ProductServiceImpl();
PageBean<Product> pageBean =
productService.findByCid(cid, pageNumber, pageSize);

// 3 将数据存放 request 作用域, 并请求转发到指定的 jsp
request.setAttribute("pageBean", pageBean);
return "/jsp/product_list.jsp";
}
```

● 步骤 3: 编写 PageBean 对象

```
public class PageBean<T> {

    private int pageNumber; //当前页 (浏览器传递)
    private int pageSize;   //每页显示个数 (固定值, 也可以是浏览器传递)
    private int totalRecord; //总记录数 (数据库查询)
    private int totalPage;  //总分页数
    private int startIndex; //开始索引
    private List<T> data;   //分页数据 (数据库查询)

    public PageBean(int pageNumber, int pageSize, int totalRecord) {
        this.pageNumber = pageNumber;
        this.pageSize = pageSize;
        this.totalRecord = totalRecord;

        // 成员变量和局部变量重名, 此处统一使用成员变量, 及都需要添加 "this."
        // #1 总分页数
        this.totalPage = (this.totalRecord + this.pageNumber - 1) / this.pageSize;
        // #2 开始索引
        this.startIndex = (this.pageNumber-1) * pageSize;
    }
}
```

● 步骤 4: 修改 ProductService, 添加 findByCid()方法

```
//接口
/**
 * 分页查询所有商品
 * @param cid
 * @param pageNumber
 * @param pageSize
 * @return
```

```
*/  
  
PageBean<Product> findByCid(String cid, int pageNumber, int pageSize)  
    throws SQLException;
```

//实现类

```
public PageBean<Product> findByCid(String cid, int pageNumber, int pageSize) throws SQLException {  
    //1 获得总记录数  
    int totalRecord = productDao.findTotalRecordByCid(cid);  
    //2 封装数据  
    PageBean<Product> pageBean = new PageBean<>(pageNumber, pageSize, totalRecord);  
    //3 分页数据  
    List<Product> data = productDao.findAllByCid(cid, pageBean.getStartIndex(), pageBean.getPageSize());  
    pageBean.setData(data);  
  
    return pageBean;  
}
```

- 步骤 5：修改 ProductDao，提供 findTotalRecordByCid() 和 findAllByCid()

```
//接口  
/**  
 * 查询总记录（含分页）  
 * @param cid  
 * @return  
 * @throws SQLException  
 */  
int findTotalRecordByCid(String cid) throws SQLException;  
  
/**  
 * 查询指定分类的所有书籍（含分页）  
 * @param cid  
 * @param startIndex  
 * @param pageSize  
 * @return  
 * @throws SQLException  
 */  
List<Product> findAllByCid(String cid, int startIndex, int pageSize) throws  
SQLException;
```

//实现类

@Override

```
public int findTotalRecordByCid(String cid) throws SQLException {  
    QueryRunner queryRunner = new QueryRunner(JDBCUtils.getDataSource());  
    String sql = "select count(*) from product where cid = ?";
```

```
Long count = (Long) queryRunner.query(sql, new ScalarHandler(), cid);
return count.intValue();
}

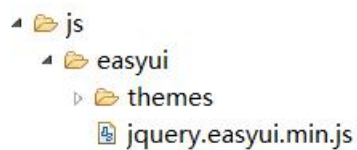
@Override
public List<Product> findAllByCid(String cid, int startIndex, int pageSize) throws SQLException {
    QueryRunner queryRunner = new QueryRunner(JDBCUtils.getDataSource());
    String sql = "select * from product where cid = ? and pflag = ? order by pdate desc limit ?,?";
    List<Product> list = queryRunner.query(sql, new BeanListHandler<Product>(Product.class),
cid,0,startIndex,pageSize);
    return list;
}
```

● 步骤 6：修改/jsp/product_list.jsp 页面，显示数据

```
<c:forEach items="${pageBean.data}" var="product">
<div class="col-md-2" style="height:240px;" >
    <a href="${pageContext.request.contextPath}/ProductServlet?method=findById&pid=${product.pid}">
        
    </a>
    <p><a href="${pageContext.request.contextPath}/ProductServlet?method=findById&pid=${product.pid}">
        <c:if test="{fn:length(product.pname) lt 27}"> ${product.pname}</c:if>
        <c:if test="{fn:length(product.pname) ge 27}"> ${fn:substring(product.pname,0,27)} ...</c:if>
    </a></p>
    <p><font color="#FF0000">商城价： &yen;${product.shop_price}</font></p>
</div>
</c:forEach>
```

● 步骤 7：分页条

1) 拷贝 easyui 资源



2) 导入 easyui css 和 js

```
<%--easyui --%>
<link rel="stylesheet" type="text/css"
    href="../../js/easyui/themes/icon.css" />
<link rel="stylesheet" type="text/css"
    href="../../js/easyui/themes/bootstrap/easyui.css" />
<script type="text/javascript"
    src="../../js/easyui/jquery.easyui.min.js" ></script>
```

3) 使用 \$.pagination 显示分页条，并设置回调函数

```
<div style="width:600px;margin:0 auto;margin-top:50px;">
    <div class="easyui-panel">
```



```
<div id="paginationId"
    style="font-size:14px;margin:0;display:block;"></div>
</div>

<script type="text/javascript">
    $(function() {
        $("#paginationId").pagination({
            total: '${pageBean.totalRecord}',          //总记录数
            pageSize: '${pageBean.pageSize}',          //每页显示个数
            pageNumber: '${param.pageNumber}',         //当前页
            layout: ['first', 'prev', 'sep', 'links',
                'sep', 'next', 'last', 'sep', 'manual'], //布局
            beforePageText: '当前第',                  //显示跳转文本框，enter 键触发
            afterPageText: '页，共{pages}页',
            displayMsg: '当前显示{from}到{to}条，共{total}条记录', //显示记录信息
            onSelectPage: function (pageNumber, pageSize) { //页面改变是触发
                location.href
                =
                "${pageContext.request.contextPath}/ProductServlet?method=findByCid&&cid=${param.ci
                d}&pageNumber=" + pageNumber;
            }
        });
    });
</script>
```