本文档为结合《网络开发工具与技术》课程知识

与课外自学内容独立完成的网站项目

MySpace 购物网站介绍

专业：档案学

年级：2014 级

姓名：梁镇涛

日期：2017 年 5 月 11 日

目录

一、项目概览

1. 基本简介

MySpace 购物商城是一个基于 Java 开发的网站，使用 JSP+Servlet+JavaBean 的 MVC 模式进行功能设计，数据库使用 MySQL。MySpace 能够实现除了实际支付外一个购物网站的基本功能，包括但不限于用户注册登录、信息管理、商品购买等。除了基本功能外，还加入了使用 echarts 实现的“热销榜”和“剁手榜”两个动态的统计图表功能，在服务器层面上也做了一些优化来使得网站性能得到提高。

在本文接下来的部分，将会以网站功能模块的方式进行网站的介绍，具体功能模块请参阅第 2 小节。

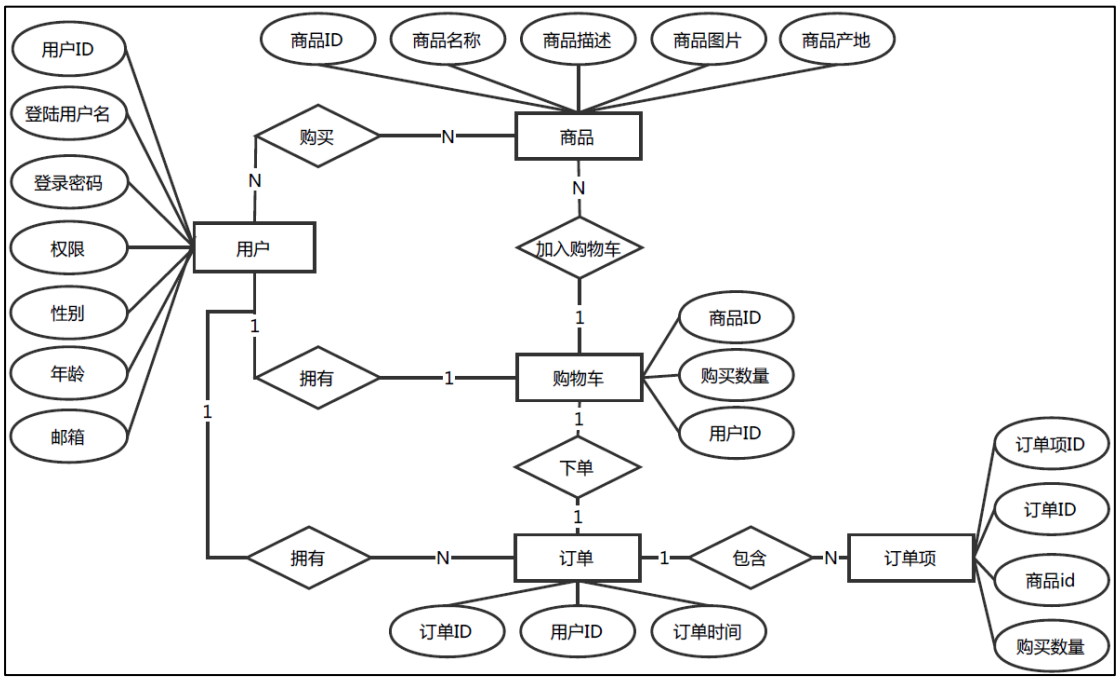
网站访问地址为：http://tomleung.cn （因域名未备案，若无法访问请输入 IP 进行访问：119.29.67.158，管理员用户名密码均为 super ）

网站源码托管地址：https://github.com/tomleung1996/JSP-Shop

2. 功能模块概览

首先要介绍的是网站数据库的结构，整个网站的数据库由五张表组成，分别是存放用户所有信息的 users 表、存放商品所有信息的 goods 表、存放所有用户购物车内容的 carts 表、存放订单基本信息（id、创建时间等）的 orders 表以及存放订单详情的 order_details 表。此外，还有一个基于 orders 表及 order_details 表连接建立的 orders_view 视图，便于程序查询利用。

从图 1 可以看出，在这五张表的基础上，网站最基本的功能可以分为针对两大对象的四大模块：



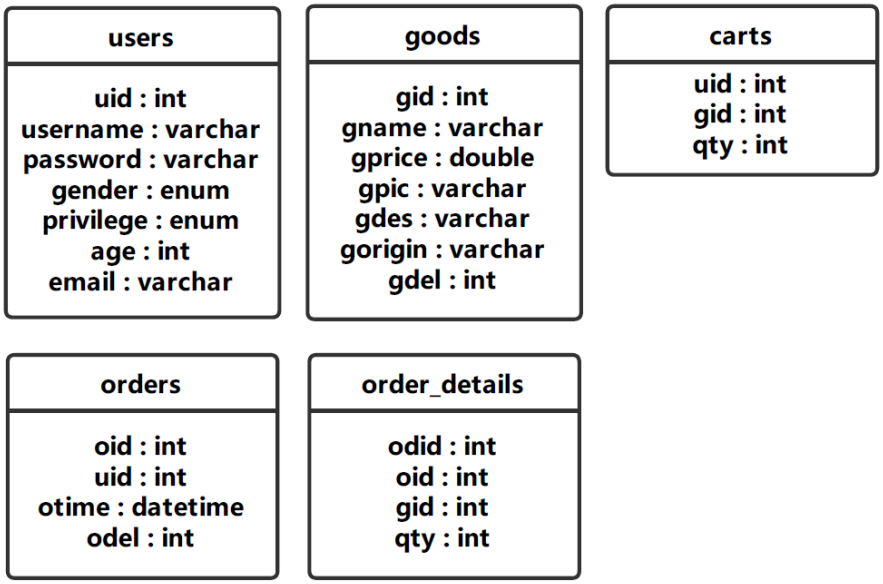


图 1 MySpace 数据库 ER 图及表结构

这四大模块分别是用户模块、商品模块、购物车模块和订单模块，其中这些模块大都是围绕用户和商品两个对象的联系进行的，用户和商品的互动也是这个网站的业务核心。

在每一个功能模块中，其功能实现的纵向层次结构如图 2 所示，从下到上最底层为数据库的表，然后是与数据库进行连接的 Connector 程序，接着是存放了

增删改查等业务操作的 DAO 程序，而这些 DAO 程序会被 Servlet 所调用以便为 Servlet 进行页面跳转提供决策依据，而最终 Servlet 会根据 DAO 提供的结果决定跳转到哪个 JSP，并将结果存放在 JavaBean 中以便 JSP 进行显示。同样地，JSP 也可以通过表单向 Servlet 提供信息，最终经过 DAO 和 Connector 到达并影响数据库（JSP、Servlet 和 DAO 类实际上并不是一种纵向关系，但是这里是为了模块功能的解释方便而将其阐述为一种纵向的双向流程）

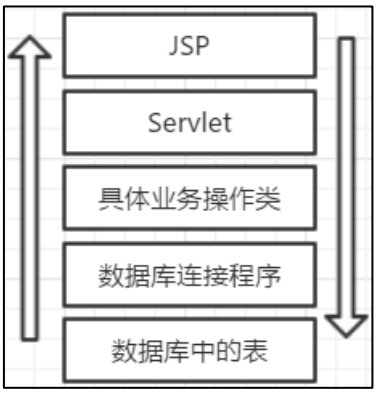


图 2 各功能模块实现层次图

接下来将按照功能模块对网站功能及其实现过程进行详细介绍。

二、用户功能模块

1. 功能概览

在本模块中，实现的功能主要有用户的注册、登录、信息的查看与修改、密码的修改。

2. 注册功能

注册功能使用 register.jsp 作为视图，User.java 作为数据 JavaBean，RegisterServlet.java 作为 Servlet 控制器，UserDAOImpl.java 和 DBConnect.java 进

行数据库操作。此外，使用了 NameValidateServlet.java 作为 ajax 验证用户名是否重复的后台支持。

首先，用户在 JSP 页面填写用户注册信息，表单如下图所示：

图 3 注册信息表单

表单的正确性首先会由页面上的 JavaScript 脚本进行检验，不正确则在输入框下方用红色字提示，点击提交后表单信息会被提交到 Servlet 进行核验，Servlet 在调用数据校验方法之后会决定注册信息有误还是注册成功（将信息写入数据库）。下图为注册 Servlet 的关键部分代码，其中 userDAO 是一个关于用户对象的数据库操作类实例，verify 是一个数据校验方法类的实例，user 是关于用户对象的 JavaBean 实例。Servlet 首先通过 request.getParameter 方法接收表单数据并存入 JavaBean，然后在经过校验后会将通过校验的 user 存放到数据库的 users 表中（调用 userDAO 的 insert 方法），如果通不过校验则会将请求转回到注册的 JSP 页面，并在请求作用域中加入错误信息属性，以供 JSP 页面报错使用。

```
try {
    if (userDAO.queryByName(user.getUsername()) != null) {
        request.setAttribute("fail", "用户名已被占用");
        request.getRequestDispatcher("register.jsp").forward(request, response);
        return;
    }
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
// 判断密码位数是否足够，否则返回register.jsp
if (!verify.PasswordLengthVerify(user)) {
    request.setAttribute("fail", "密码长度少于8位");
    request.getRequestDispatcher("register.jsp").forward(request, response);
    return;
}
// 判断两个密码是否相等，否则返回register.jsp
if (!verify.PasswordEqualVerify(user)) {
    request.setAttribute("fail", "两次密码输入不一致");
    request.getRequestDispatcher("register.jsp").forward(request, response);
    return;
}
// 判断年龄是否为数字，否则返回register.jsp
if (!verify.AgeNumericVerify(user)) {
    request.setAttribute("fail", "请输入正确的年龄");
    request.getRequestDispatcher("register.jsp").forward(request, response);
    return;
}
// 判断email是否正确，否则返回register.jsp
if (!verify.EmailVerify(user)) {
    request.setAttribute("fail", "邮箱格式不正确");
    request.getRequestDispatcher("register.jsp").forward(request, response);
    return;
}
```

```
// 通过全部检查后写入数据库

try {
    userDAO.insert(user);
    response.sendRedirect("login.jsp?success=1");
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

在这个 Servlet 中，所使用到的 User JavaBean 因结构比较简单，故不放出具体代码（就是包含了用户名、密码等几个域及其 getter 与 setter 方法的类）。而所用到的 UserDAOImpl 类由于不必要的代码太多，故在下面展示其实现的接口 UserDAO，具体实现方式不作展示（具体实现方式就是调用数据库连接类，

并通过 Preparedstatement 来执行 sql 并返回结果）

```java
package cn.tomleung.dao;

import java.util.List;

public interface UserDAO {
    public void insert(User user) throws Exception;
    public void updatePwd(User user) throws Exception;
    public void updateInfo(User user) throws Exception;
    public void delete(User user) throws Exception;
    public User queryByName(String username) throws Exception;
    public User queryByID(int uid) throws Exception;
    public List<User> queryAll() throws Exception;
    public int[] queryAllID() throws Exception;
    public int queryBuySumByID(int gid) throws Exception;
}
```

在用户注册功能中，主要用到 insert 方法和 queryByName 方法。

3. 登录功能

登录功能使用 login.jsp 作为视图，LoginServlet.java 作为控制器，数据操作类与 JavaBean 与注册功能所使用的相同，不再赘述。

首先用户通过登录 JSP 进行登录表单的填写并提交信息：

表单的信息会被提交到登录 Servlet 进行处理，该 Servlet 主要使用了 userDAO 的 queryByName 方法从数据库中拿到一个关于用户对象的 JavaBean（由数据库操作类负责产生该 JavaBean 并返回），然后将该 Bean 中的密码字段与用户所输入的进行比较。如果 Bean 为空或者密码不符则登录失败，否则登录成功，将该 Bean 存放至 session，跳转到 ShowAllServlet 显示首页商品。错误信息的返回方法与注册页面类似。下面是该 Servlet 的代码：

```java
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    // TODO Auto-generated method stub
    request.setCharacterEncoding("UTF-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    UserDAO userDAO = DAOFactory.getUserDAOInstance();
    HttpSession session = request.getSession(true);

    try {
        User user = userDAO.queryByName(username);
        if (user == null || (!MD5.Bit32(password).equalsIgnoreCase(user.getPassword()))) {
            request.setAttribute("fail", "用户名或密码有误，请重新核对");
            request.getRequestDispatcher("login.jsp").forward(request, response);
            return;
        } else {
            session.setAttribute("user", user);
            response.sendRedirect("ShowAllServlet");
            return;
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

UserDAOImpl 所实现的方法在上一小节已经介绍过，在此不赘述。

4. 信息查看与修改功能

信息查看功能实际上在登录功能中已经完成。如果登录成功，用户的 session 中会被存放一个关于用户的 JavaBean，该 Bean 中有用户的完整信息。用户只需要进入到 profile.jsp 即可查看自己的信息，如下图所示：

欢迎您，亲爱的：super 会员

权限：	超级管理员
性别：	男
年龄：	20
邮箱：	super@tomleung.cn

而信息修改功能涉及到的 Servlet 是 ModifyInfoServlet，用户通过填写 modifyinfo.jsp 的表单来提交想要修改的信息（如下图所示）

Servlet 部分会对这些信息进行正确性检验（与注册类似），通过检验则调用 userDAO 的 updateInfo 方法，否则返回错误信息。关键代码如下所示：

```
if (!verify.EmailVerify(user)) {
    request.setAttribute("fail", "邮箱格式不正确，请重新检查");
    request.getRequestDispatcher("modifyinfo.jsp").forward(request, response);
    return;
}
if (!verify.AgeNumericVerify(user)) {
    request.setAttribute("fail", "请输入正确的年龄");
    request.getRequestDispatcher("modifyinfo.jsp").forward(request, response);
    return;
}
try {
    userDAO.updateInfo(user);
    user = userDAO.queryByID(uid);
    session.setAttribute("user", user);
    response.sendRedirect("profile.jsp");
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

5. 密码修改功能

用户通过 modifypwd.jsp 来进行原密码和新密码、确认密码的填写，会有 JavaScript 对这些信息进行初步的验证，如下图所示：

ModifyPwdServlet 接收并处理这些信息，如果原密码正确且新密码两次输入一致则调用 userDAO 的 updatePwd 方法更新数据库中的密码，否则返回错误信息。具体实现与修改用户信息的过程类似，在此不表。

三、商品功能模块

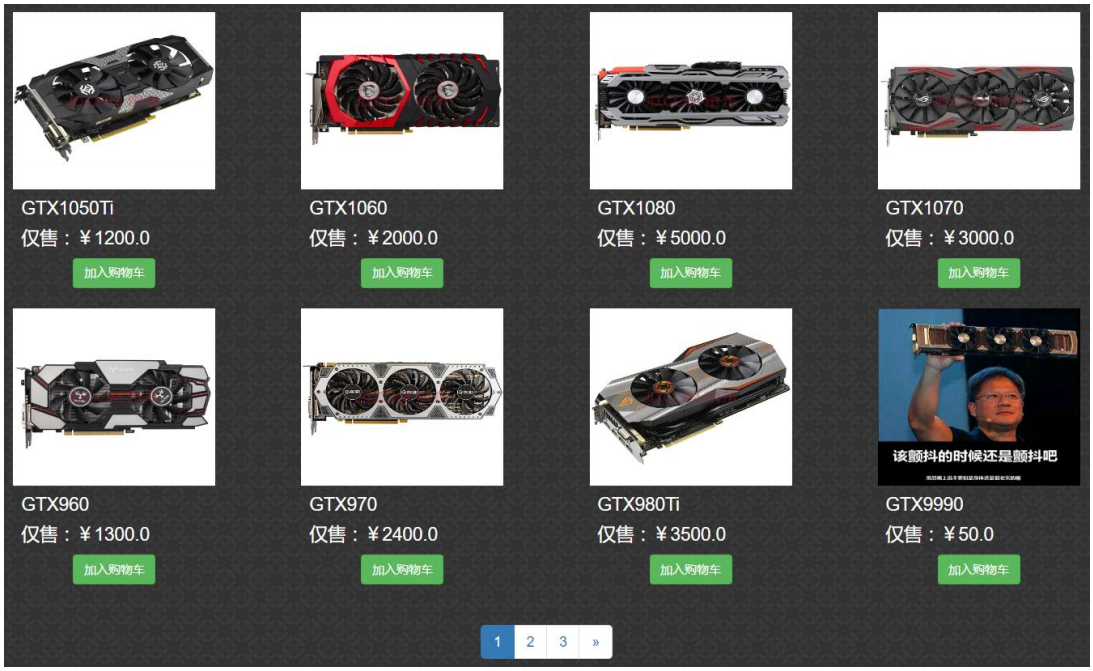
1. 功能概览

在本模块中，实现的功能主要有首页商品显示、商品详情显示、商品增加、搜索、删除和修改。

2. 首页商品显示功能

商品显示功能使用 index.jsp 作为视图，ShowAllServlet.java 作为控制器，Goods.java 作为 JavaBean 存放数据，使用 GoodDAO 的实现类进行数据库操作，数据库中的表为 goods。

首先，用户进入首页会看到一个商品的列表显示，带有翻页功能：



这个 JSP 页面使用 EL 表达式和 JSTL 标签来显示存放在 session 中的一个 ArrayList<Good>，每四个换一行。而该 ArrayList 是由 ShowAllServlet 存放进去的（在过滤器中设置了访问 index.jsp 的请求都会被转到 ShowAllServlet 中）下面是 ShowAllServlet 的关键部分代码：

```
String currentPageStr = request.getParameter("currentPage");
int itemPerPage = 8;
int currentPage = 1;
if (currentPageStr != null && !currentPageStr.equals("")) {
    currentPage = Integer.parseInt(currentPageStr.trim());
}

try {
    int totalGoods = goodDAO.queryAllCount();
    if (totalGoods < itemPerPage) {
        itemPerPage = totalGoods;
    }
    int totalPages = totalGoods / itemPerPage + ((totalGoods % itemPerPage) > 0 ? 1 : 0);
    ArrayList<Good> all = goodDAO.queryLimit((currentPage - 1) * itemPerPage, itemPerPage);
    request.setAttribute("totalPages", totalPages);
    request.setAttribute("currentPage", currentPage);
    session.setAttribute("all", all);
    request.getRequestDispatcher("index.jsp").forward(request, response);
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

该 Servlet 通过调用 goodDAO 的 queryAllCount 方法得到商品总数，根据每页所要显示的商品数来得出总页数。再调用 queryLimit 方法来取出每一页所要显示的商品，并将其放在一个 ArrayList 中添加到 session。该 Servlet 在每次翻页都会对数据库进行一次查询，和使用缓存结果集的翻页方法比较，该方法的实时性会更强。

Servlet 所使用到的 GoodDAO 接口含有的方法如下图所示，具体实现方式因过于冗杂不表，核心就是 sql 语句的写法，可以在本文第一部分的 github 网址中打开 GoodDAOImpl.java 文件查看具体的实现代码。

```java
package cn.tomleung.dao;

import java.util.ArrayList;

public interface GoodDAO {
    public void insert(Good good) throws Exception;
    public void update(Good good) throws Exception;
    public void delete(Good good) throws Exception;
    public ArrayList<Good> queryByName(String gname) throws Exception;
    public ArrayList<Good> queryByNameLimit(String gname,int from,int limit) throws Exception;
    public Good queryByID(int gid) throws Exception;
    public ArrayList<Good> queryAll() throws Exception;
    public ArrayList<Good> queryLimit(int from,int limit) throws Exception;
    public int queryAllCount() throws Exception;
    public int[] queryAllID() throws Exception;
    public int querySellSumByID(int gid) throws Exception;
}
```

3. 商品详情显示功能

用户通过点击首页中商品的图片或者购物车、历史订单中商品的名字可以进入到商品详情的显示页面，如下图所示：

商品详情的显示功能关键在于找到商品的 ID，而商品的 ID 在生成商品 URL

的时候已经写入到里面了。因为在这个网站中商品是以一个有血有肉的实体（JavaBean）存在的，在生成连接到商品详情页面的 URL 时，程序通过读取 Bean 中的 gid 变量并将其加在 URL 后面（?gid=xxx），而负责详情显示的 Servlet 则可以通过 request.getParameter 方法来得到这个值进行商品的查找与显示。

GoodDetailServlet 的代码很简单，如下图所示：

```
protected void service(HttpServletRequest request, HttpServletResponse response) t
    // TODO Auto-generated method stub
    int gid = Integer.parseInt(request.getParameter("gid"));
    Good good = null;
    GoodDAO goodDAO = DAOFactory.getGoodDAOInstance();
    try {
        good=goodDAO.queryByID(gid);
        request.setAttribute("good", good);
        request.getRequestDispatcher("gooddetail.jsp").forward(request, response);
        return;
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

使用到了 goodDAO 的 queryByID 方法，将查询到的商品存放到 request 作用域中供 JSP 页面使用，具体的显示方式在 JSP 中定义。

4. 商品搜索功能

商品搜索功能的入口为导航栏上的搜索框，搜索的显示使用到 search.jsp 作为视图，GoodSearchServlet 作为控制器，数据库操作类和 JavaBean 类与前述功能相同。用户输入的界面如下图所示：

点击搜索后，结果如下图所示：

搜索是对商品名称的搜索，搜索结果可以分页进行显示，下面展示该 Servlet 的关键部分代码：

```
try {
    int totalGoods = goodDAO.queryByName(gname).size();
    if (totalGoods == 0) {
        request.setAttribute("fail", "没有找到任何结果");
        session.setAttribute("all", null);
        request.getRequestDispatcher("search.jsp").forward(request, response);
        return;
    }
    if (totalGoods < itemPerPage) {
        itemPerPage = totalGoods;
    }
    int totalPages = totalGoods / itemPerPage + ((totalGoods % itemPerPage) > 0 ? 1 : 0);
    ArrayList<Good> all = goodDAO.queryByNameLimit(gname, (currentPage - 1) * itemPerPage, itemPerPage);
    request.setAttribute("totalPages", totalPages);
    request.setAttribute("currentPage", currentPage);
    session.setAttribute("all", all);
    request.getRequestDispatcher("search.jsp").forward(request, response);
    return;

} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

主要使用到了 goodDAO 的 queryByName 与 queryByNameLimit 方法

5. 增加商品功能

只有管理员可以通过 goodinsert.jsp 页面填写商品信息并进行增加操作，过滤器会保证这一点（过滤器的介绍在最后一部分）。负责这一功能的 Servlet 是 GoodInsertServlet，其他辅助类与上述功能相同。

管理员进行要增加的商品信息填写界面如下所示：

该表格使用了 multipart/form-data 的编码方式以实现图片上传功能，所以后台的 Servlet 不能像处理普通表单一样去接受他的信息，为保证服务器安全，还必须对上传图片的格式和大小进行检验。Servlet 的关键部分代码如下所示：

```
request.setCharacterEncoding("UTF-8");
response.setContentType("text/html; charset=UTF-8");
Good good = new Good();
GoodDAO goodDAO = DAOFactory.getGoodDAOInstance();
String savePath = "/gpic";
File saveDir = new File(savePath);
if(!saveDir.exists()){
    saveDir.mkdir();
}

DiskFileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload sfu = new ServletFileUpload(factory);
sfu.setHeaderEncoding("UTF-8");
sfu.setFileSizeMax(1024*1024*2);

try{
    List<FileItem> items = sfu.parseRequest(request);
    for(FileItem item:items){
        String fieldName=item.getFieldName();
        if(item.isFormField()){
            String value=item.getString();
            value = new String(value.getBytes("ISO-8859-1"),"UTF-8");
            switch(fieldName){
            case "gname":
                good.setGname(value);
                break;
            case "gprice":
                good.setGprice(Double.parseDouble(value));
                break;
            case "gorigin":
                good.setGorigin(value);;
                break;
            case "gdes":
                good.setGdes(value);;
                break;
```

```
}else{
    String fileName=item.getName();
    int pos = fileName.lastIndexOf("\\");
    fileName = fileName.substring(pos+1);
    if(!(fileName.toLowerCase().endsWith(".jpg")||fileName.endsWith(".png")||fileName.endsWith(".jpeg"))){
        request.setAttribute("fail", "文件格式不正确");
        request.getRequestDispatcher("goodinsert.jsp").forward(request, response);
        return;
    }
    int pos2 = fileName.lastIndexOf(".");
    fileName = good.getGname()+fileName.substring(pos2);
    File file = new File(savePath,fileName);
    item.write(file);
    good.setGpic(savePath+"/"+fileName);
    goodDAO.insert(good);
}
```

Servlet 会将信息一一填入关于商品对象的 JavaBean 中，在确认各项信息无误后，会调用 goodDAO 的 insert 方法插入商品。

6. 删除商品功能

只有管理员可以删除商品（过滤器同样保证这一点），商品的删除实现方式与商品详情的实现方式有点类似，即在构造删除按钮所指向的 URL 时拼接了商品的 id，Servlet 拿到该 ID 后可以进行准确的删除。商品的删除可以在商品详情页或者是商品管理页面进行，所涉及的 Servlet 是 GoodDeleteServlet。商品删除界面如下图所示：

Servlet 关键部分的代码如下图所示：

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    // TODO Auto-generated method stub
    int gid = Integer.parseInt(request.getParameter("gid"));
    Good good = new Good();
    good.setGid(gid);
    GoodDAO goodDAO = DAOFactory.getGoodDAOInstance();
    try {
        goodDAO.delete(good);
        response.sendRedirect("ShowAllServlet?flag=1");
    } catch (Exception e) {
        // TODO Auto-generated catch block
        request.setAttribute("fail", "删除失败");
        request.getRequestDispatcher("index.jsp").forward(request, response);
        return;
    }
}
```

主要调用了 goodDAO 的 delete 方法，值得一提的是，该方法不是真正从数据库中删除该商品的信息，而是把该商品的 odel 列设置为 1，表示删除，在网页上就不会显示出来，但在日后的数据统计中还能发挥作用。

7. 修改商品功能

修改商品功能本质上和添加商品功能的逻辑是一致的，只不过 sql 语句上有差别，该功能所涉及的 Servlet 是 GoodUpdateServlet，用到了 goodDAO 的 update 方法，具体实现代码和添加功能几乎一样，在此不表。同样，也只有管理员可以修改商品信息，修改入口和删除入口的位置几乎完全一样。

四、购物车功能模块

1. 功能概览

在本模块中，实现的功能主要有添加商品至购物车、购物车内商品的增减、清空购物车和结算。

2. 添加至购物车功能

这个功能的入口是“加入购物车”按钮，该按钮出现的位置在首页和商品详情页面中。实现该功能的 Servlet 是 CartInsertServlet，数据库操作类是 CartDAOImpl，存放数据的 JavaBean 是 Cart，所使用到的数据库表是 carts。

首先，用户可以通过点击加入购物车按钮将心仪的商品加入到购物车中，按钮如下图所示：

点击按钮之后，会将按钮所对应的商品 ID 传到负责的 Servlet 当中，Servlet

先从 session 中取出用户的 ID（在登录时存放的），连同商品 ID 一起查询数据库中是否存在这样一条记录，如果存在，则调用 cartDAO 的 update 方法让该购物车中该用户购买该商品的数量加一，否则调用 insert 方法插入这样一条数据。具体代码如下所示：

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcep
    // TODO Auto-generated method stub
    HttpSession session = request.getSession(true);
    int gid = Integer.parseInt(request.getParameter("gid"));
    int uid = ((User)session.getAttribute("user")).getUid();
    int currentPage = Integer.parseInt(request.getParameter("currentPage"));
    Cart cart = new Cart();
    cart.setUid(uid);
    cart.setGid(gid);
    cart.setQty(1);
    CartDAO cartDAO = DAOFactory.getCartDAOInstance();
    try {
        Cart tmpCart=cartDAO.queryByUGID(uid, gid);
        if(tmpCart==null){
            cartDAO.insert(cart);
            request.setAttribute("success", "加入购物车成功！");
            request.getRequestDispatcher("ShowAllServlet?currentPage="+currentPage).forward(request, response);
            return;
        }else{
            cart.setQty(tmpCart.getQty()+1);
            cartDAO.update(cart);
            request.setAttribute("success", "加入购物车成功！");
            request.getRequestDispatcher("ShowAllServlet?currentPage="+currentPage).forward(request, response);
            return;
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        request.setAttribute("fail", "加入购物车失败！");
        request.getRequestDispatcher("index.jsp").forward(request, response);
        return;
    }
```

下面展示 CartDAO 这个接口所具有的方法，不展示具体实现的原因和前面的原因一样，完整的具体实现代码可以在 github 上面查看。

```
package cn.tomleung.dao;

import java.util.ArrayList;

public interface CartDAO {
    public void insert(Cart cart) throws Exception;
    public void update(Cart cart) throws Exception;
    public void delete(Cart cart) throws Exception;
    public void truncate(int uid) throws Exception;
    public ArrayList<Cart> queryByID(int uid) throws Exception;
    public Cart queryByUGID(int uid,int gid) throws Exception;
    public ArrayList<Cart> queryAll() throws Exception;
}
```

3. 购物车内商品增减功能

这里包含了增加和减少两个功能，这两个功能的入口是在购物车界面的绿色加号和红色减号按钮，如下图所示：

其背后分别由 CartAddServlet 和 CartDeleteServlet 两个 Servlet 完成，依赖的辅助类和前述功能一样。CartAddServlet 被调用时会得到一个商品 ID 和用户 ID，通过这两个 ID 查询到这条购物车记录原本的购买数量，对其进行加一操作后存放回数据库。同理，CartDeleteServlet 只是进行了减法操作（有一点不同，如果购买数量为 1 的情况下继续减少购买量，该记录会消失）。

购买量增加的 Servlet 代码如下所示：

```
HttpSession session = request.getSession(true);
int gid = Integer.parseInt(request.getParameter("gid"));
int uid = ((User)session.getAttribute("user")).getUid();
int currentPage = Integer.parseInt(request.getParameter("currentPage"));
Cart cart = new Cart();
cart.setUid(uid);
cart.setGid(gid);
cart.setQty(1);
CartDAO cartDAO = DAOFactory.getCartDAOInstance();
try {
    Cart tmpCart=cartDAO.queryByUGID(uid, gid);
    if(tmpCart==null){
        cartDAO.insert(cart);
        request.setAttribute("success", "加入购物车成功！");
        request.getRequestDispatcher("ShowAllServlet?currentPage="+currentPage).forward(request, response);
        return;
    }else{
        cart.setQty(tmpCart.getQty()+1);
        cartDAO.update(cart);
        request.setAttribute("success", "加入购物车成功！");
        request.getRequestDispatcher("ShowAllServlet?currentPage="+currentPage).forward(request, response);
        return;
    }
} catch (Exception e) {
    // TODO Auto-generated catch block
    request.setAttribute("fail", "加入购物车失败！");
    request.getRequestDispatcher("index.jsp").forward(request, response);
    return;
}
```

主要用到了 cartDAO 的 insert 方法和 update 方法，而购买量减少的 Servlet 代码和这个基本一致，主要用到了 delete 方法和 update 方法，在此不做展示。

4. 清空购物车功能

清空购物车的入口按钮也在购物车页面中，如下图所示：

点击该按钮会提示确认信息，点击确定清空后会调用 CartTruncateServlet，该 Servlet 在 session 中简单获取用户的 ID，调用 CartDAO 的 truncate 方法将该用户的购物车内容全部删除。删除后会跳转到购物车显示页面，具体代码很简短，如下所示：

```
HttpSession session = request.getSession(true);
int uid = ((User)session.getAttribute("user")).getUid();
CartDAO cartDAO = DAOFactory.getCartDAOInstance();
try {
    cartDAO.truncate(uid);
    session.setAttribute("carts", null);
    response.sendRedirect("CartShowServlet");
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    request.setAttribute("fail", "清空购物车失败");
    request.getRequestDispatcher("CartShowServlet").forward(request, response);
    return;
}
```

5. 购物车显示功能

几乎忘了提到这个最基本的购物车显示功能。主要使用到了 cart.jsp 作为显示视图，CartShowServlet 作为控制器，所依赖的辅助类和前述功能相同。购物车显示的界面如下图所示：

JSP 页面通过 EL 表达式和 JSTL 标签从 session 中存放的 ArrayList<Cart>中取值并显示，而该数组是由 Servlet 存放至 session 中的。负责的 Servlet 主要使用了用户的 ID 作为 cartDAO 的 queryByID 方法的参数，查询到该用户的购物车记录并将其存放在 session 中。代码如下所示：

```
HttpSession session = request.getSession(true);
int uid = ((User)session.getAttribute("user")).getUid();
ArrayList<Cart> carts = new ArrayList<Cart>();
CartDAO cartDAO = DAOFactory.getCartDAOInstance();
GoodDAO goodDAO = DAOFactory.getGoodDAOInstance();
try {
    ArrayList<Cart> cs = cartDAO.queryByID(uid);
    for(Cart cart:cs){
        Cart singleCart = new Cart();
        Good tmpGood = goodDAO.queryByID(cart.getGid());
        singleCart.setUid(cart.getUid());
        singleCart.setGid(tmpGood.getGid());
        singleCart.setGname(tmpGood.getGname());
        singleCart.setGprice(tmpGood.getGprice());
        singleCart.setQty(cart.getQty());
        singleCart.setSubsum(singleCart.getGprice()*singleCart.getQty());
        carts.add(singleCart);
    }
    session.setAttribute("carts", carts);
    response.sendRedirect("cart.jsp");
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

6. 结算功能

结算功能是购物车与订单模块的一个交汇点，主要通过购物车页面中的入口调用 OrderCreateServlet，在数据库的 orders 与 order_details 表中创建相应的记录，在成功创建后清空购物车，达到实际上的结算效果。

OrderCreateServlet 主要调用了 orderDAO 的 insert 方法和 cartDAO 的 truncate 方法，orderDAO 接口所具有的全部方法稍后在订单模块会进行展示，下面展示 OrderCreateServlet 的具体代码：

```
HttpSession session = request.getSession(true);
int uid = ((User)session.getAttribute("user")).getUid();
CartDAO cartDAO = DAOFactory.getCartDAOInstance();
@SuppressWarnings("unchecked")
ArrayList<Cart> carts = (ArrayList<Cart>)session.getAttribute("carts");
OrderDAO orderDAO = DAOFactory.getOrderDAOInstance();
try {
    orderDAO.insert(carts);
    cartDAO.truncate(uid);
    session.setAttribute("carts", null);
    response.sendRedirect("cash.jsp");
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

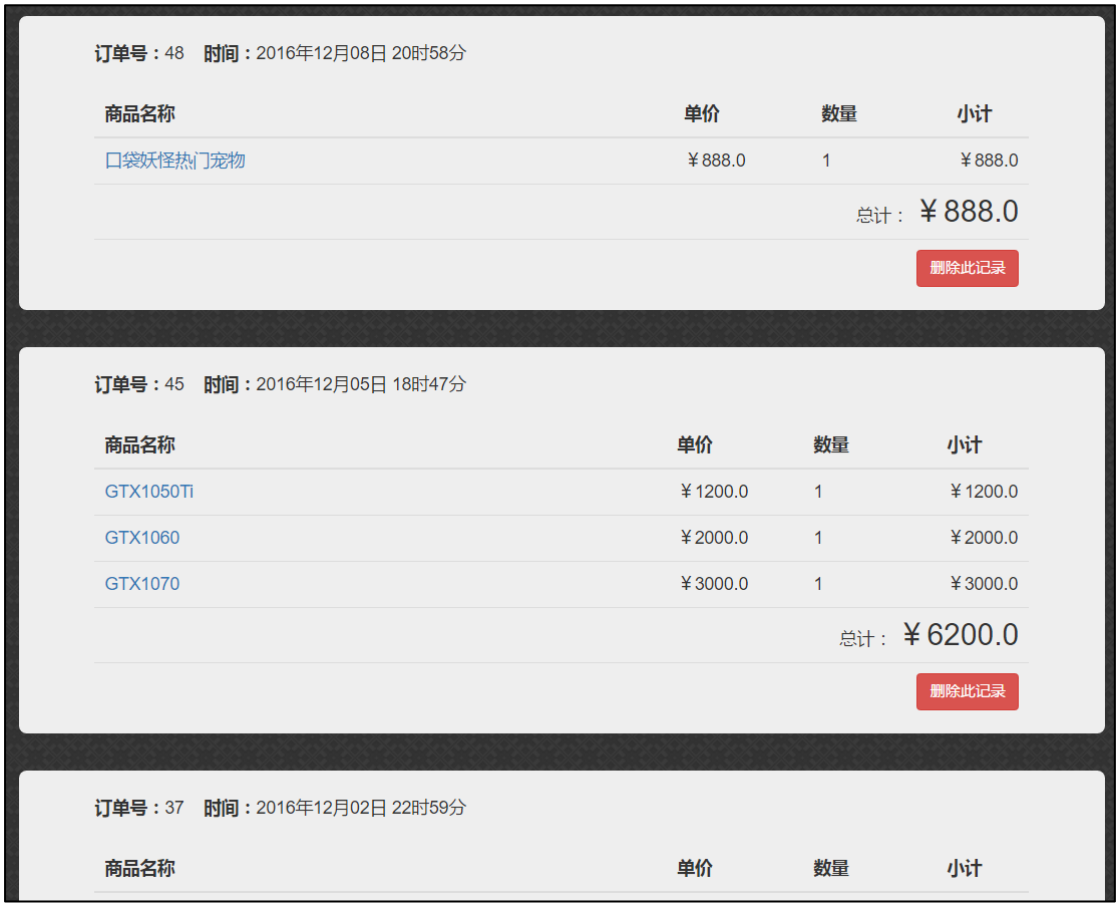
五、订单功能模块

1. 功能概览

在本模块中，实现的功能主要包括订单显示、订单创建（结算）和订单删除功能。这个模块在项目中属于比较难实现的部分，订单创建功能在购物车模块已经讲过，在此不再赘述。

2. 订单显示功能

订单显示功能由 order.jsp 作为视图显示，OrderShowServlet 作为控制器，OrderDAOImpl 作为数据库操作类，Order 和 OrderHead 作为存放数据的 JavaBean。订单页面的显示效果如下图所示;



每一个订单会用一个灰色圆角矩形区分开，各订单会包含（且仅包含一个）订单号和时间，以及具体的商品内容。在这个页面主要由 JSP 通过 EL 表达式和 JSTL 标签按照特定的方式读取 session 中的 ArrayList<OrderHead>生成。其中，一个 OrderHead 表示一个订单（圆角矩形）的内容，它持有一个时间、订单号以及一个 ArrayList<Order>对象，Order 对象中存放了订单中具体的每一条商品条目。

Servlet 的具体代码如下图所示，看起来代码比较简单，实际的业务代码都被

封装在了 OrderDAOImpl 这个实现类中，但限于篇幅不能进行展示，可以在网上查看源码：

```
HttpSession session = request.getSession(true);

int uid = ((User)session.getAttribute("user")).getUid();
OrderDAO orderDAO = DAOFactory.getOrderDAOInstance();
GoodDAO goodDAO = DAOFactory.getGoodDAOInstance();
try {
    ArrayList<OrderHead> orders = orderDAO.queryByUID(uid);
    for(OrderHead o:orders){
        double sum=0;
        for(Order o2:o.getOrder()){
            o2.setGood(goodDAO.queryByID(o2.getGid()));
            o2.setSubsum(o2.getGood().getGprice()*o2.getQty());
            sum+=o2.getSubsum();
        }
        o.setSum(sum);
    }
    Collections.sort(orders);
    session.setAttribute("orders", orders);
    response.sendRedirect("order.jsp");
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

该 Servlet 主要用到了 OrderDAO 的 queryByUID 方法，值得一提的是 OrderHead 实现了 Comparable 接口，通过 sort 方法实现了按照日期顺序显示订单。OrderDAO 接口所具有的方法如下图所示：

```
package cn.tomleung.dao;

import java.util.ArrayList;

public interface OrderDAO {
    public void insert(ArrayList<Cart> carts) throws Exception;
    public void delete(int oid) throws Exception;
    public ArrayList<OrderHead> queryByUID(int uid) throws Exception;
    public int queryOrderNumberByUID(int uid) throws Exception;
}
```

3. 订单删除功能

订单删除功能的入口在订单查看界面每个订单右下角的删除按钮，点击按钮会进行确认，确认删除后会调用 OrderDeleteServlet 方法，所使用到的辅助类和前述功能一致。功能入口如下图所示：

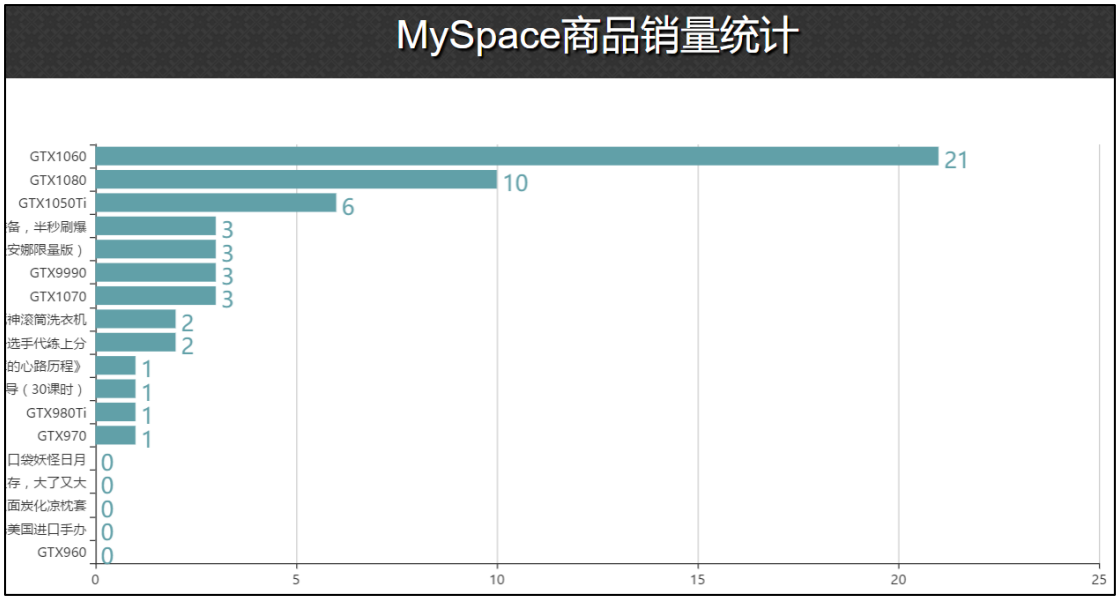
Servlet 具体的代码非常简单，调用了 OrderDAO 的 delete 方法，oid 是在生成删除订单按钮的链接时拼接进去的。

```
int oid = Integer.parseInt(request.getParameter("oid"));
OrderDAO orderDAO = DAOFactory.getOrderDAOInstance();
try {
    orderDAO.delete(oid);
    response.sendRedirect("OrderShowServlet");
    return;
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

六、其他功能模块

1. 商品销量统计功能

此功能利用了 echarts 来实现，使用 ajax 来获取由 ShowSellSumServlet 提供的 JSON 格式的销量数据，并在 sellsum.jsp 页面显示出来，显示效果如下图所示：



Servlet 的主要作用就是通过调用 goodDAO 的相关方法，形成两个 JSON 数组，并将这两个数组封装到一个 JSON 对象中通过输出流传给 JavaScript 需要接受的位置。这两个 JSON 数组分别是商品的名字及其销量，构造完成的结果如下图所示：

```
{
    "data":
        [0,0,0,0,0,1,1,1,1,2,2,3,3,3,3,6,10,21],
    "categories":
        ["GTX960","源氏美国进口手办","竹藤双面炭化凉枕套",
        "极致显存，大了又大","口袋妖怪日月","GTX970","GTX980Ti",
        "美国数学辅导专家盖本牛威尔 一对一极品辅导（30课时）",
        "华莱士国际讲座入场券\u2014\u2014《论自我修养与成为跑步冠军的心路历程》",
        "守望先锋国服第一选手代练上分","手残福利，旋转陀螺，死神滚筒洗衣机",
        "GTX1070","GTX9990","充气娃娃（年轻安娜限量版）","极品装备，半秒刷爆",
        "GTX1050Ti","GTX1080","GTX1060"]
}
```

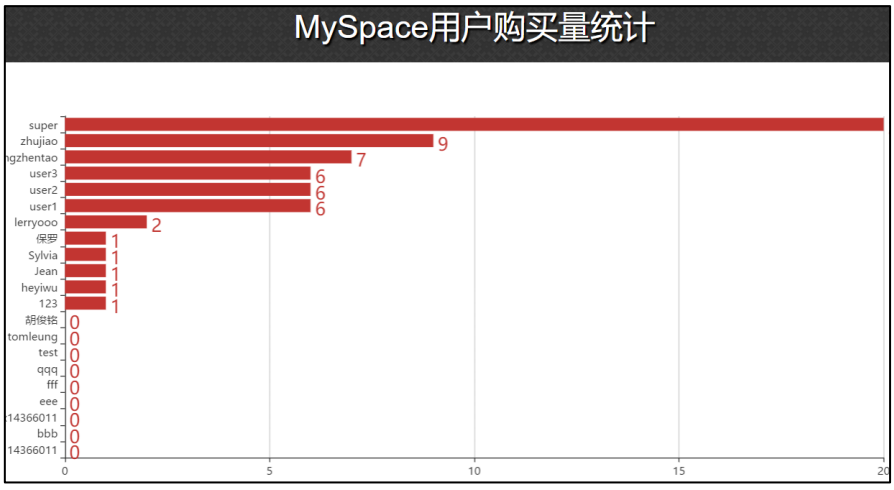
Servlet 的具体代码如下图所示，其中 sortByValue 方法的作用是将 Map 内元素按照其值升序排列。主要使用到了 goodDAO 的 queryAllID、queryByID 和 querySelllSumByID 三个方法：

```java
int[] goodIDs = null;
ArrayList<Good> goodList = new ArrayList<Good>();
Map<String, Integer> sellSum = new LinkedHashMap<String, Integer>();
GoodDAO goodDAO = DAOFactory.getGoodDAOInstance();
try {
    goodIDs = goodDAO.queryAllID();
    int j = 0;
    for (int i : goodIDs) {
        goodList.add(goodDAO.queryByID(i));
        int singleSum = goodDAO.querySellSumByID(i);
        sellSum.put(goodList.get(j++).getGname(), singleSum);
    }
    sellSum=sortByValue(sellSum);
    JSONArray gname = new JSONArray(sellSum.keySet());
    JSONArray sell = new JSONArray(sellSum.values());
    Map<String, JSONArray> a = new LinkedHashMap<String, JSONArray>();
    a.put("categories", gname);
    a.put("data", sell);
    JSONObject json = new JSONObject(a);
    response.setContentType("text/html; charset=utf-8");
    response.getWriter().write(json.toString());
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

而 JSP 页面的 echarts 脚本只需要根据百度提供的模板替换数据即可，有需要的话也可以自己根据官网的教程设定喜欢的外观样式，因为代码很长在此不表。

2. 用户购买量统计功能

和商品销量统计功能很类似，只是数据搜集来源不同，用户购买量网页的运行效果如下图所示：



负责提供数据的 Servlet 是 ShowBuySumServlet，其具体代码如下图所示：

```
int[] userIDs = null;
ArrayList<User> userList = new ArrayList<User>();
Map<String, Integer> buySum = new LinkedHashMap<String, Integer>();
UserDAO userDAO = DAOFactory.getUserDAOInstance();
try {
    userIDs = userDAO.queryAllID();
    int j = 0;
    for (int i : userIDs) {
        userList.add(userDAO.queryByID(i));
        int singleSum = userDAO.queryBuySumByID(i);
        buySum.put(userList.get(j++).getUsername(), singleSum);
    }
    buySum=sortByValue(buySum);
    JSONArray uname = new JSONArray(buySum.keySet());
    JSONArray buy = new JSONArray(buySum.values());
    Map<String, JSONArray> a = new LinkedHashMap<String, JSONArray>();
    a.put("categories", uname);
    a.put("data", buy);
    JSONObject json = new JSONObject(a);
    response.setContentType("text/html; charset=utf-8");
    response.getWriter().write(json.toString());
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

主要使用到了 userDAO 的 queryAllID、queryByID 和 queryBuySumByID 三个方法，和商品销量很类似，构造完成的 JSON 对象输出内容如下图所示：


```
{
    "data":
        [0,0,0,0,0,0,0,0,0,1,1,1,1,1,2,6,6,6,7,9,20],
    "categories":
        ["14366011","bbb","cx14366011","eee","fff",
        "qqq","test","tomleung","胡俊铭","123",
        "heyiwu","Jean","Sylvia","保罗","lerryooo",
        "user1","user2","user3","fuckuliangzhentao",
        "zhujiao","super"]
}
```


3. 过滤器保护功能

在项目初期，我通过在每一个需要被保护的 JSP 页面中加入判断语句来判断用户身份，后来我发现这种方式工作量大而且不可取，通过网上搜索和学习，转

而使用了过滤器（Filter）的方法来统一对网站中的资源进行保护。当前项目启用了两个过滤器，一个是用来压缩和设置静态内容缓存的（这个只需要在 tomcat 配置文件中启用即可，具体代码无须自己编写，故在此不详细说），而另一个则是我自己编写的，用来防止未登录用户进入网站内部页面和防止普通用户访问特权页面的，名字为 LoginFilter.java。其关键部分代码如下所示：

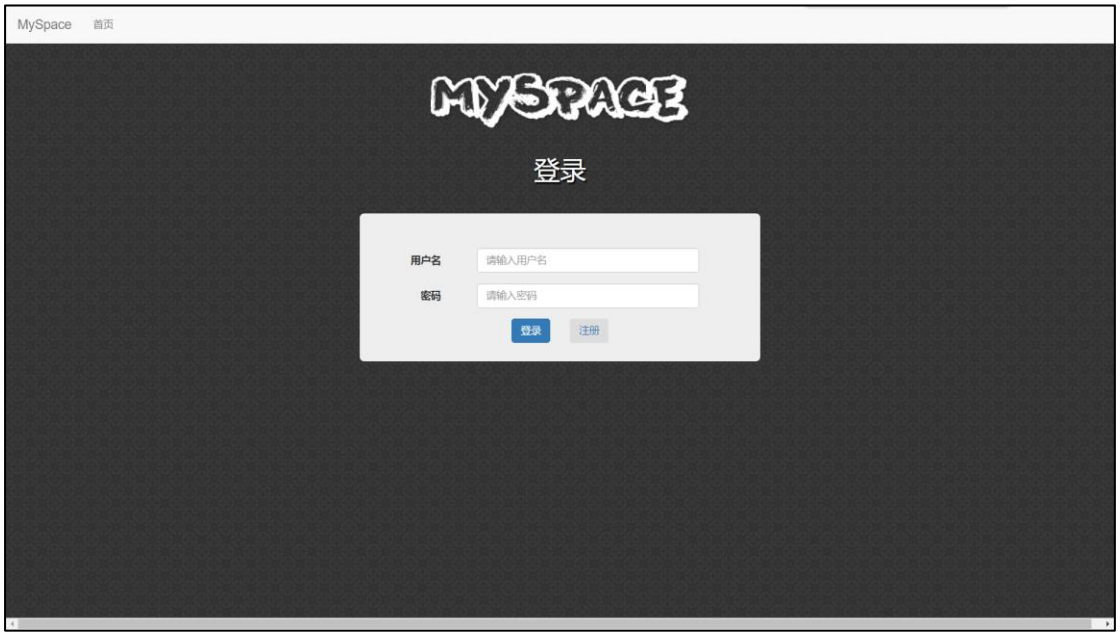
```
HttpServletRequest request = (HttpServletRequest) srequest;
HttpServletResponse response = (HttpServletResponse) sresponse;
HttpSession session = request.getSession(true);
String target = request.getRequestURI();
String regex1 = "login\\.jsp|register\\.jsp|LoginServlet|RegisterServlet"; // 过滤未登录用户
String regex2 = "goodmanage\\.jsp|goodupdate\\.jsp|goodinsert\\.jsp"; // 过滤非超级管理员用户
Pattern p1 = Pattern.compile(regex1);
Matcher m1 = p1.matcher(target);
Pattern p2 = Pattern.compile(regex2);
Matcher m2 = p2.matcher(target);

if (!(m1.find())) {
    if (session.getAttribute("user") == null || ((User) session.getAttribute("user")).getUsername() == null) {
        response.sendRedirect("login.jsp");
        return;
    } else if (m2.find() && !((User) session.getAttribute("user")).getPrivilege().equals("超级管理员")) {
        request.setAttribute("fail", "对不起，该功能只允许超级管理员使用");
        request.getRequestDispatcher("index.jsp").forward(request, response);
        return;
    } else if (target.indexOf("search") > 0) {
        response.sendRedirect("ShowAllServlet?search=1");
    } else if (target.indexOf("index.jsp") > 0||target.endsWith("cn")||target.endsWith("myshop/")) {
        response.sendRedirect("ShowAllServlet");
    } else if (target.indexOf("goodmanage.jsp") > 0) {
        response.sendRedirect("ShowAllServlet?flag=1");
    } else {
        chain.doFilter(request, response);
        return;
    }
} else {
    chain.doFilter(request, response);
    return;
}
```

这个过滤器使用了正则表达式来匹配 URL 中的信息，如果未登录用户访问的不是登录或注册页面则会被重定向至登录页面。如果普通用户试图访问特权页面，会被转到首页并提示警告信息。除此之外，该过滤器还保证了用户访问首页时会先经过 Servlet 产生数据，但是我觉得这种做法似乎不是很好。

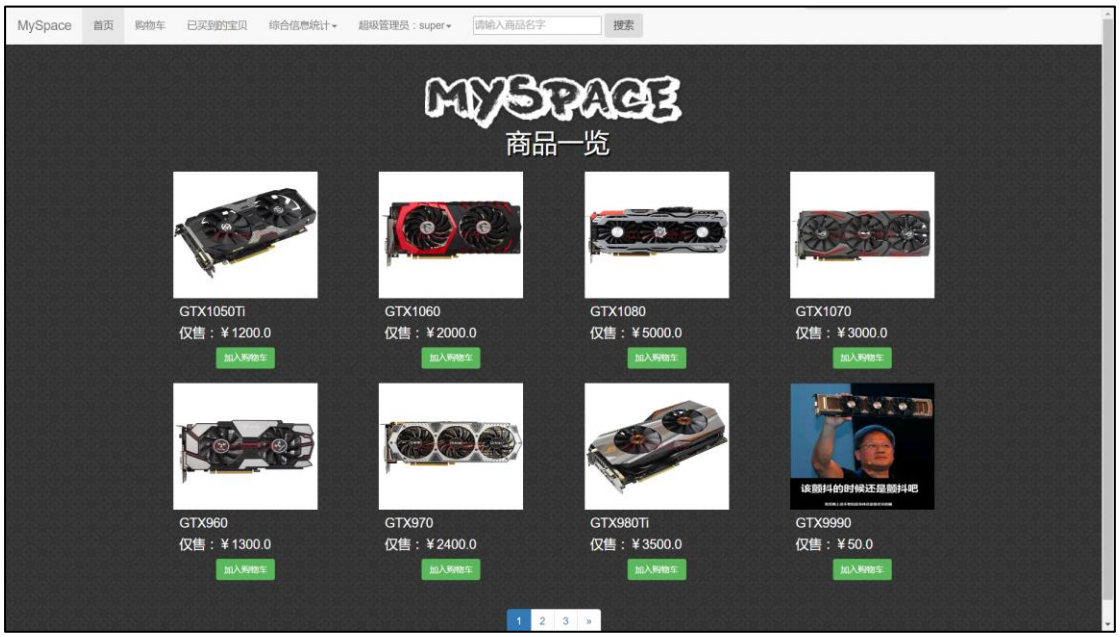
七、网站运行效果概览

受篇幅和载体形式所限，在此无法完全展示网站的真实运行效果，仅截取部分关键功能进行展示。真实运行效果可以前往网站观看。

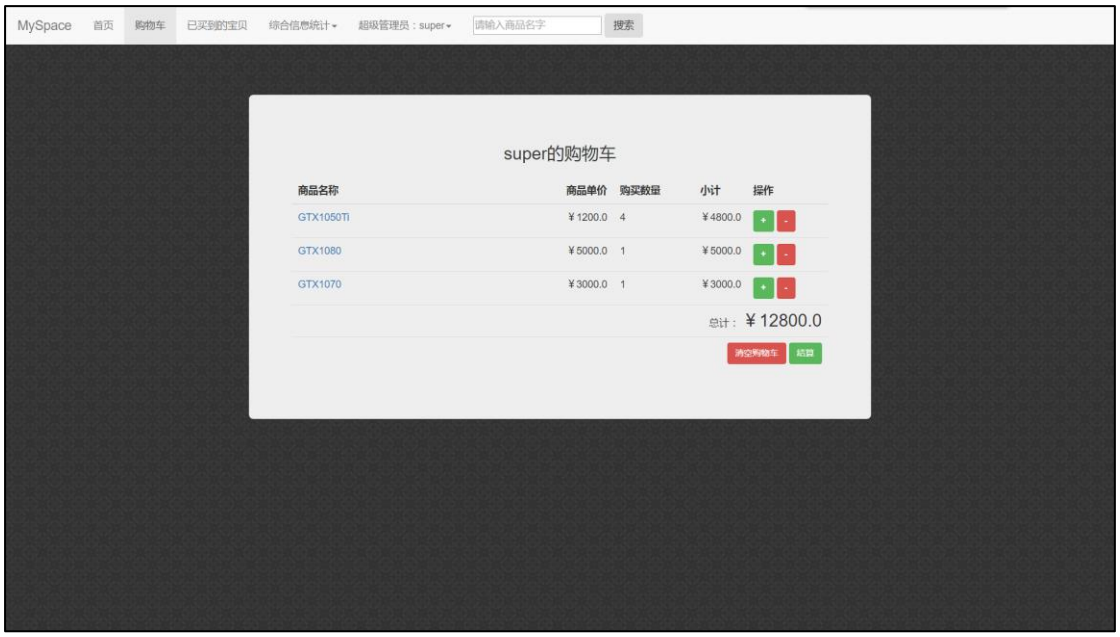
网站登录前页面：



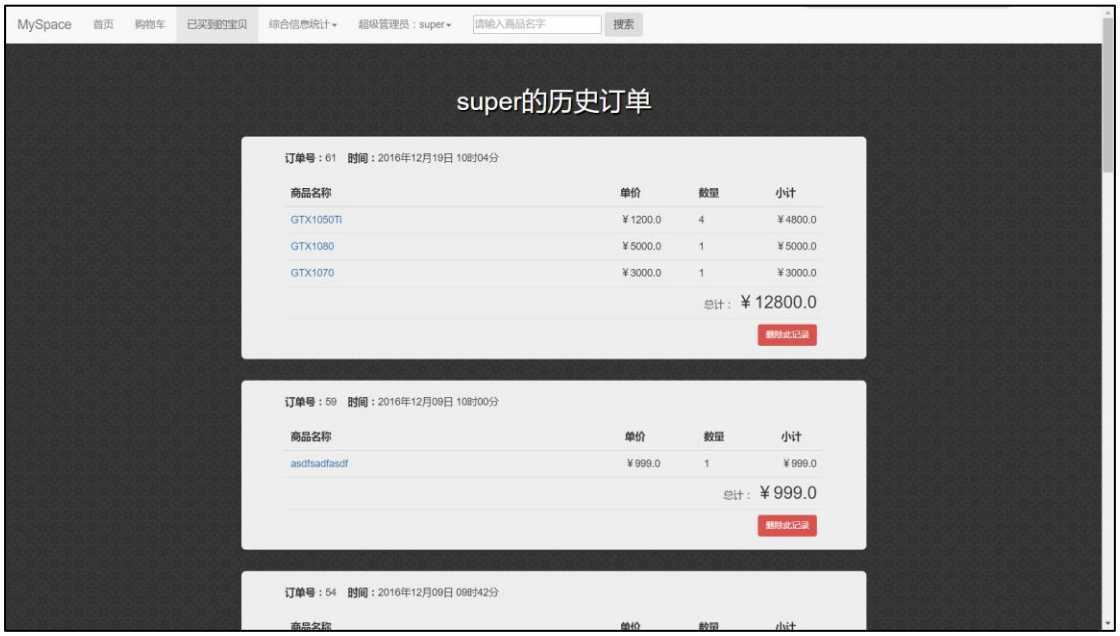
登陆后的首页：



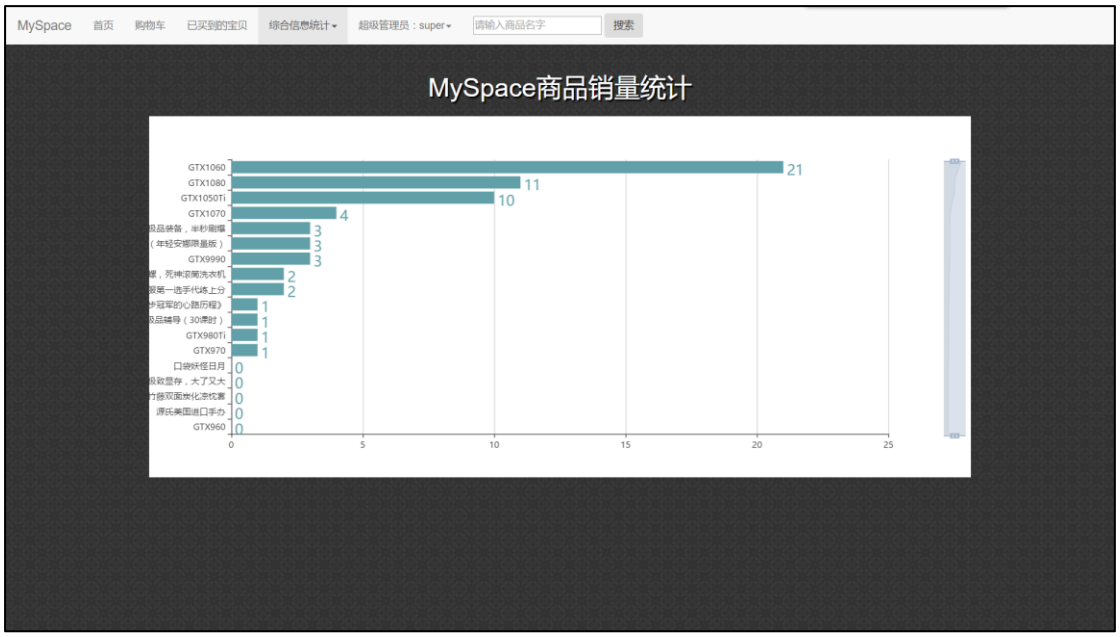
购物车页面：



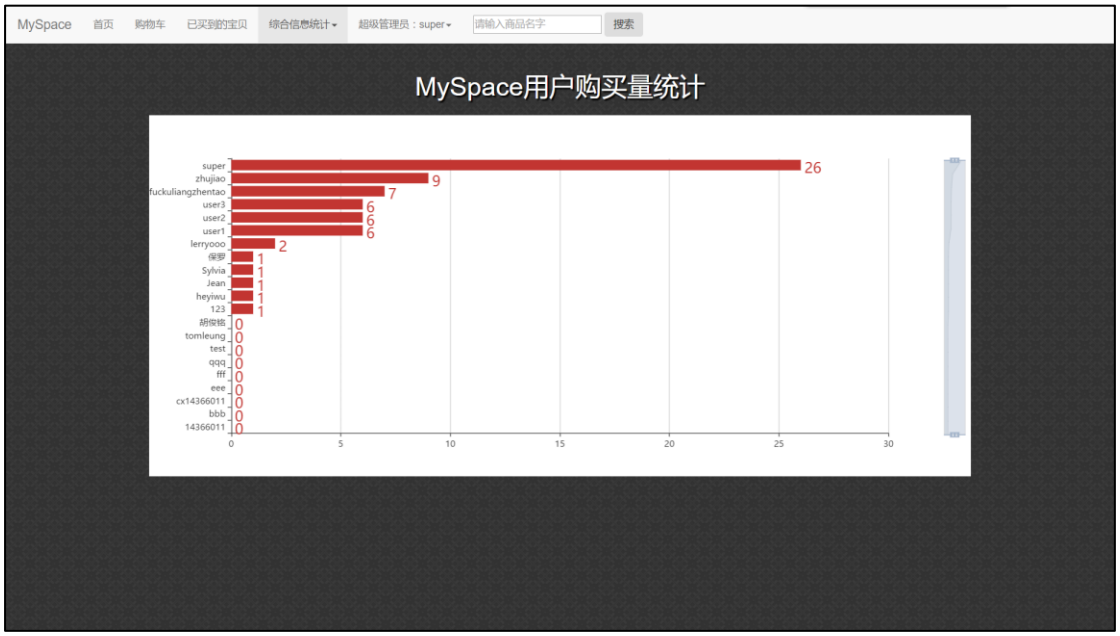
历史订单界面：



热销榜界面：

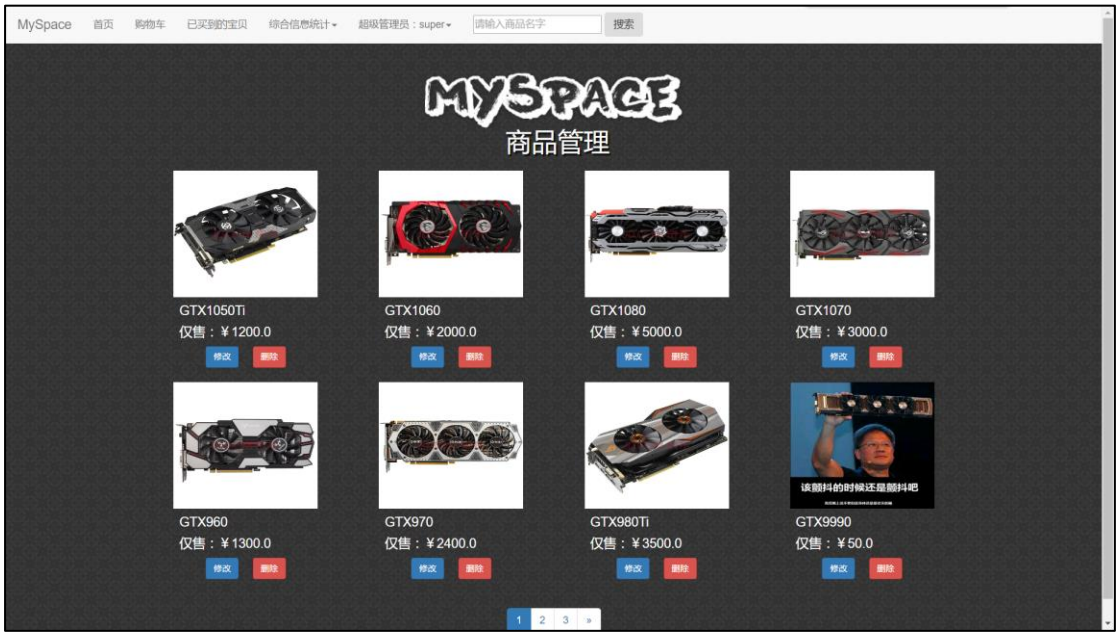

MySpace
首页
购物车
已买到的宝贝
综合信息统计
超级管理员：super
请输入商品名字
搜索
MySpace商品销量统计
GTX1060
GTX1080
GTX1050Ti
GTX1070
GTX9990
GTX980Ti
GTX970
GTX960
21
11
10
4
3
3
3
2
2
1
1
1
1
0
0
0
0
0
0
5
10
15
20
25

剁手榜界面：

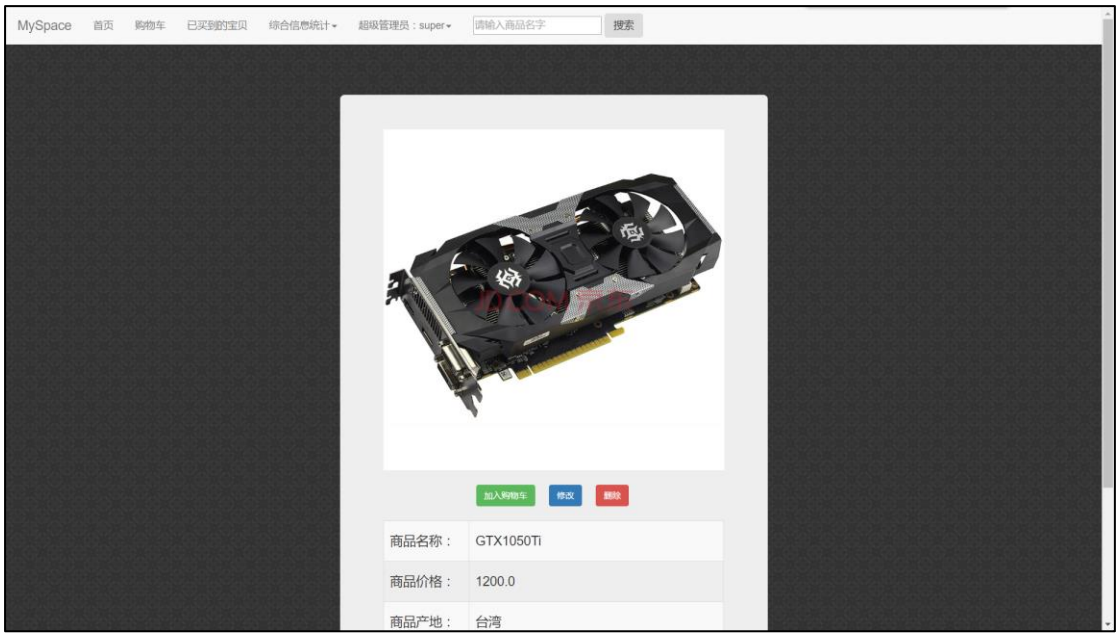

MySpace
首页
购物车
已买到的宝贝
综合信息统计
超级管理员：super
请输入商品名字
搜索
MySpace用户购买量统计
super
zhujiao
fuckuliangzhentao
user3
user2
user1
lerryooo
Sylvia
Jean
heyiwu
123
tomleung
test
qqq
fff
eee
cx14366011
bbb
14366011
26
9
7
6
6
6
2
1
1
1
1
1
0
0
0
0
0
0
0
0
5
10
15
20
25
30

个人信息查看页面：

商品管理页面：



商品详情页面：



商品添加界面：