中南大学

Web应用开发技术

实验报告

|  |  |
| --- | --- |
| 姓 名： | 江名威,林温健 |
| 学 号： | 8209190325, 8209190306 |
| 班 级： | 软件工程1903 |
| 指导老师： | 宋 铁 |
| 完成时间： | 2021/1/27 |
| 分 数： |  |
| 批阅日期： |  |

**实验一 基于MVC用JSP/Servlet实现JpetStore**

1. **实验内容——基本任务。**
2. 商品展示业务模块，包括大类Category、小类Product和具体商品Item的展示和搜索功能。

**1.1大类Category**



大类Category先实现Category.jsp进行大类的具体展示，利用jsp中c:foreach语法对每个大类进行展示，具体类的ID利用a标签进行跳转到小类进行查看，接下来实现domain层的Category.java，完成大类的基础设计，接下来实现persistence中的数据库连接层，先写DAO定义所有接口，再写实现的Implement方法，对数据库进行操作；再写一个Category的Service类，可以在Servlet层中调用该类的相关方法进行进一步操作，最后完成ViewCategoryServlet对jsp页面与业务层进行连接跳转。

完成Control层ViewCategoryServlet:

**public** **class** **ViewCategoryServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEW\_CATEGORY = "/WEB-INF/jsp/catalog/Category.jsp";

**private** **String** categoryId;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

categoryId = request.getParameter("categoryId");

**CatalogService** service = **new** **CatalogService**();

**Category** category = service.getCategory(categoryId);

**List**<**Product**> productList = service.getProductListByCategory(categoryId);

//保存数据

**HttpSession** session = request.getSession();

session.setAttribute("category", category);

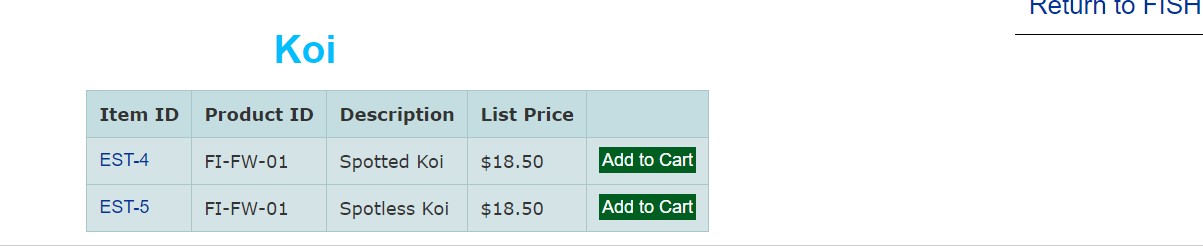
session.setAttribute("productList", productList);

//跳转页面

request.getRequestDispatcher(VIEW\_CATEGORY).forward(request, response);

}

**1.2小类Product**



小类Product先完善Product.jsp，主要是通过productId对产品类进行信息展示，之后再实现Product.java，接下进行ProductDAOImpl数据库的连接进行编写，最后完成Servlet的编写。

控制层Servlet代码

**public** **class** **ViewProductServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEW\_PRODUCT = "/WEB-INF/jsp/catalog/Product.jsp";

**private** **String** productId;

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

productId = request.getParameter("productId");

**CatalogService** service = **new** **CatalogService**();

**Product** product = service.getProduct(productId);

**List**<**Item**> itemList = service.getItemListByProduct(productId);

**HttpSession** session = request.getSession();

session.setAttribute("product", product);

session.setAttribute("itemList", itemList);

request.getRequestDispatcher(VIEW\_PRODUCT).forward(request, response);

}

}

**1.3具体商品Item**



用户点击产品后进入查看具体商品，完善Item.jsp，Copy代码模版中的Item.java，实现ItemDAOImpl的具体数据库连接操作，完成最后的Servlet连接。

控制层的Servlet连接:ViewItemServlet:

**public** **class** **ViewItemServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEW\_ITEM = "/WEB-INF/jsp/catalog/Item.jsp";

**private** **String** itemId;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

itemId = request.getParameter("itemId");

**CatalogService** service = **new** **CatalogService**();

**Item** item = service.getItem(itemId);

**HttpSession** session = request.getSession();

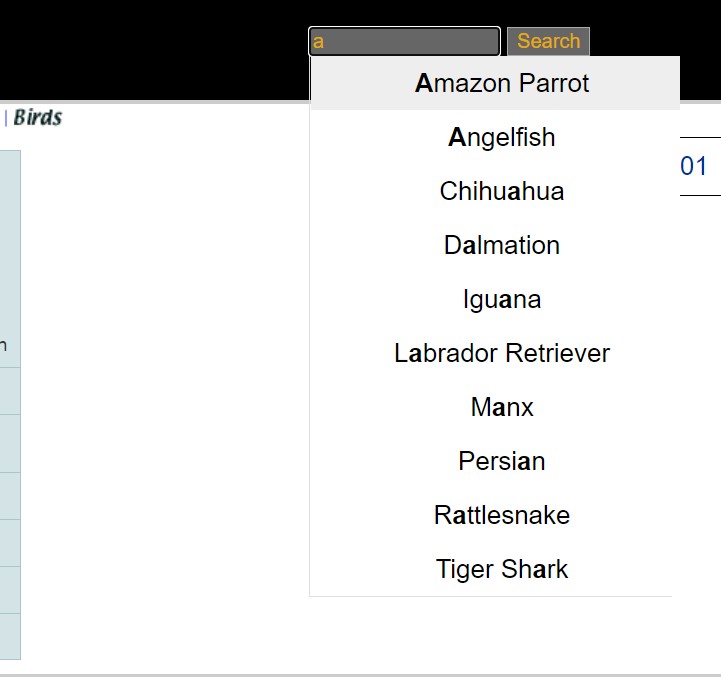
session.setAttribute("item", item);

request.getRequestDispatcher(VIEW\_ITEM).forward(request, response);

}

}

**1.4完成搜索功能**



搜索功能先完善显示的SearchProducts.jsp，接下直接进行SearchProductServlet编写和配置，通过获得输入的关键字再通过CatalogService的方法进行查找并显示。

SearchProductServlet

**public** **class** **SearchProductServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** SEARCH\_PRODUCTS = "/WEB-INF/jsp/catalog/SearchProducts.jsp";

**private** **String** keyword;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

keyword = request.getParameter("keyword");

//request.setAttribute("keyword", keyword);

**CatalogService** service = **new** **CatalogService**();

**List**<**Product**> productList = service.searchProductList(keyword);

//保存数据

**HttpSession** session = request.getSession();

session.setAttribute("keyword", keyword);

session.setAttribute("productList", productList);

//跳转页面

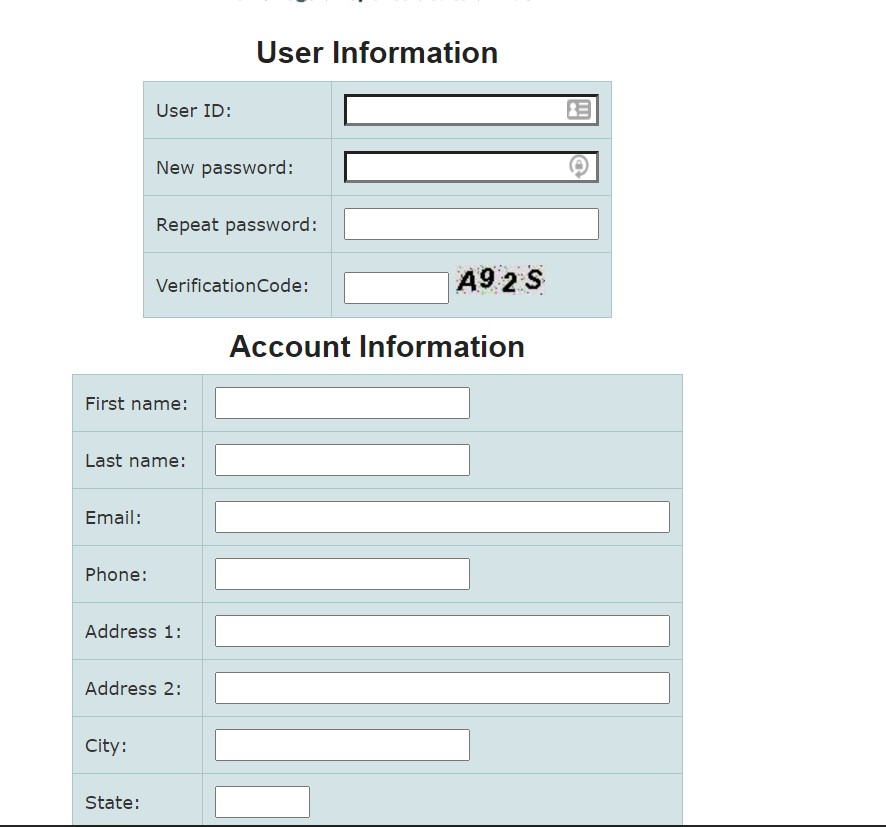
request.getRequestDispatcher(SEARCH\_PRODUCTS).forward(request, response);

}

}

1. 用户管理业务模块，包括用户注册、登录、修改用户信息、查询用户相关订单等业务功能。

**2.1用户注册**



用户注册时，需要一个全新的包含用户所有信息的输入界面即NewAccountForm.jsp，之后需要先进行设计NewAccountFormServlet，该Servlet的功能仅为实现用户点击注册后跳转用户输入新账号信息的表格，接下实现Account.java，完成Account的基础功能，之后进行AccountDAO接口定义以及完成AccountDAOImpl连接数据库的实现，接下来完成AccountService的编写，之后进行NewAccountServlet配置，获取网页界面的信息，赋值给一个新账号，再将该账号插入到数据库中。

AccountService:

**public** **class** **AccountService** {

**private** **AccountDAO** accountDAO;

**public** **AccountService**(){

accountDAO = **new** **AccountDAOImpl**();

}

**public** **Account** **getAccount**(**String** username) {

**return** accountDAO.getAccountByUsername(username);

}

**public** **Account** **getAccount**(**String** username, **String** password) {

**Account** account = **new** **Account**();

account.setUsername(username);

account.setPassword(password);

**return** accountDAO.getAccountByUsernameAndPassword(account);

}

**public** **void** **insertAccount**(**Account** account) {

accountDAO.insertAccount(account);

accountDAO.insertProfile(account);

accountDAO.insertSignon(account);

}

**public** **void** **updateAccount**(**Account** account) {

accountDAO.updateAccount(account);

accountDAO.updateProfile(account);

**if** (account.getPassword() != **null** && account.getPassword().length() > 0) {

accountDAO.updateSignon(account);

}

}

}

NewAccountServlet:

**public** **class** **NewAccountServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** MAIN = "/WEB-INF/jsp/catalog/Main.jsp";

**private** **static** **final** **String** NEWACCOUNTFORM = "/WEB-INF/jsp/account/NewAccountForm.jsp";

**private** **Account** account;

**private** **Account** account1;

**private** **AccountService** accountService;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**HttpSession** session = request.getSession();

account = (**Account**) session.getAttribute("account");

account = **null**;

session.setAttribute("account", account);

//获得输入的验证码值

**String** value1=request.getParameter("vCode");

/\*获取图片的值\*/

**String** value2=(**String**)session.getAttribute("checkcode");

**Boolean** isSame = **false**;

/\*对比两个值（字母不区分大小写）\*/

**if**(value2.equalsIgnoreCase(value1)){

isSame = **true**;

}

**String** username = request.getParameter("username");

**String** password = request.getParameter("password");

**String** firstName = request.getParameter("firstName");

**String** lastName = request.getParameter("lastName");

**String** email = request.getParameter("email");

**String** phone = request.getParameter("phone");

**String** address1 = request.getParameter("address1");

**String** address2 = request.getParameter("address2");

**String** city = request.getParameter("city");

**String** state = request.getParameter("state");

**String** zip = request.getParameter("zip");

**String** country = request.getParameter("country");

**String** languagePreference = request.getParameter("languagePreference");

**String** favouriteCategoryId = request.getParameter("favouriteCategoryId");

**String** listOption = request.getParameter("listOption");

**String** bannerOption = request.getParameter("bannerOption");

account1 = **new** **Account**();

account1.setUsername(username);

account1.setPassword(password);

account1.setFirstName(firstName);

account1.setLastName(lastName);

account1.setEmail(email);

account1.setPhone(phone);

account1.setAddress1(address1);

account1.setAddress2(address2);

account1.setCity(city);

account1.setState(state);

account1.setZip(zip);

account1.setCountry(country);

account1.setLanguagePreference(languagePreference);

account1.setFavouriteCategoryId(favouriteCategoryId);

account1.setListOption(**Boolean**.parseBoolean(listOption));

account1.setBannerOption(**Boolean**.parseBoolean(bannerOption));

**if**(isSame){

accountService = **new** **AccountService**();

accountService.insertAccount(account1);

**if**(account1 != **null**){

request.getRequestDispatcher(MAIN).forward(request, response);

}**else**{

session.setAttribute("messageAccount", "Invalid Verification Code.");

**if**(account1 != **null**){

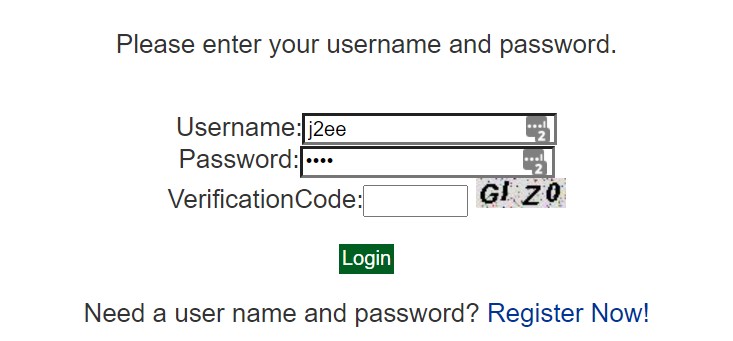
request.getRequestDispatcher(NEWACCOUNTFORM).forward(request, response);

}

}

}

**2.2用户登录**



用户登录先完成SignonForm.jsp，能够使用户输入登入的账号与密码，接下来先用一个Servlet实现SignOnFormServlet实现点击登录时网页跳转到用户输入账号密码的界面，接下来使用SignOnServlet实现获取用户的账号密码对数据库进行核查，如果存在则登录，反之则给予一定的错误信息提示。

SignOnFormServlet:

**public** **class** **SignOnFormServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** SIGNONFORM = "/WEB-INF/jsp/account/SignonForm.jsp";

**private** **Account** account;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**HttpSession** session = request.getSession();

account = (**Account**)session.getAttribute("account");

request.getRequestDispatcher(SIGNONFORM).forward(request, response);

}

}

SignOnServlet:

**public** **class** **SignOnServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** MAIN = "/WEB-INF/jsp/catalog/Main.jsp";

**private** **static** **final** **String** SIGNONFORM = "/WEB-INF/jsp/account/SignonForm.jsp";

**private** **Account** account;

**private** **AccountService** accountService;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**String** username = request.getParameter("username");

**String** password = request.getParameter("password");

accountService = **new** **AccountService**();

account = accountService.getAccount(username, password);

**HttpSession** session = request.getSession();

session.setAttribute("account", account);

//获得输入的验证码值

**String** value1=request.getParameter("vCode");

/\*获取图片的值\*/

**String** value2=(**String**)session.getAttribute("checkcode");

**Boolean** isSame = **false**;

/\*对比两个值（字母不区分大小写）\*/

**if**(value2.equalsIgnoreCase(value1)){

isSame = **true**;

}

**if** (account == **null** || !isSame){

**if**(!isSame){

session.setAttribute("messageSignOn", "Invalid Verification Code. Signon failed.");

}**else**{

session.setAttribute("messageSignOn", "Invalid username or password. Signon failed.");

}

session.setAttribute("account", **null**);

request.getRequestDispatcher(SIGNONFORM).forward(request, response);

}**else**{

account.setPassword(**null**);

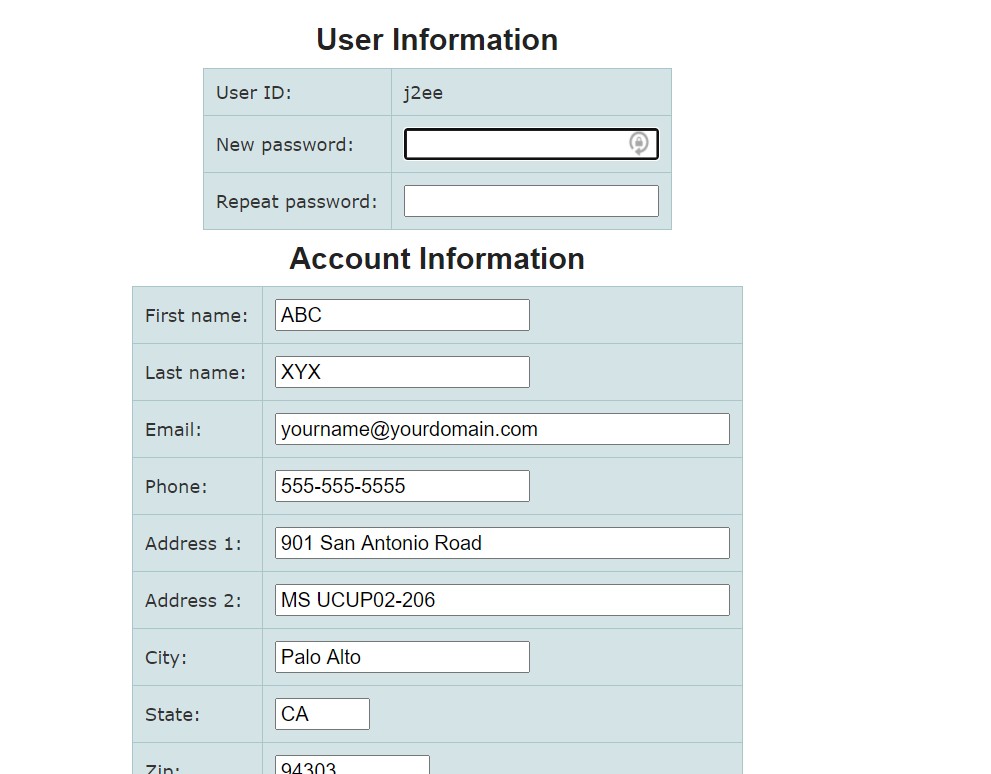
request.getRequestDispatcher(MAIN).forward(request, response);

}

}

}

**2.3用户修改信息**



修改个人账号信息也就是进行个人信息查看，同时可以进行修改，修改后点击Save Account Information即可获取新的用户信息，之后进行数据库信息更新。

EditAccountServlet:

**public** **class** **EditAccountServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** EDITACOUNT = "/WEB-INF/jsp/account/EditAccountForm.jsp";

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

request.getRequestDispatcher(EDITACOUNT).forward(request, response);

}

}

SaveAccountServlet:

**public** **class** **SaveAccountServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** EDITACOUNT = "/WEB-INF/jsp/account/EditAccountForm.jsp";

**private** **Account** account;

**private** **AccountService** accountService;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**HttpSession** session = request.getSession();

account = (**Account**) session.getAttribute("account");

**String** username = account.getUsername();

**String** password = request.getParameter("password");

**String** firstName = request.getParameter("firstName");

**String** lastName = request.getParameter("lastName");

**String** email = request.getParameter("email");

**String** phone = request.getParameter("phone");

**String** address1 = request.getParameter("address1");

**String** address2 = request.getParameter("address2");

**String** city = request.getParameter("city");

**String** state = request.getParameter("state");

**String** zip = request.getParameter("zip");

**String** country = request.getParameter("country");

**String** languagePreference = request.getParameter("languagePreference");

**String** favouriteCategoryId = request.getParameter("favouriteCategoryId");

**String** listOption = request.getParameter("listOption");

**String** bannerOption = request.getParameter("bannerOption");

account.setUsername(username);

account.setPassword(password);

account.setFirstName(firstName);

account.setLastName(lastName);

account.setEmail(email);

account.setPhone(phone);

account.setAddress1(address1);

account.setAddress2(address2);

account.setCity(city);

account.setState(state);

account.setZip(zip);

account.setCountry(country);

account.setLanguagePreference(languagePreference);

account.setFavouriteCategoryId(favouriteCategoryId);

account.setListOption(**Boolean**.parseBoolean(listOption));

account.setBannerOption(**Boolean**.parseBoolean(bannerOption));

accountService = **new** **AccountService**();

accountService.updateAccount(account);

session.setAttribute("account", account);

request.getRequestDispatcher(EDITACOUNT).forward(request, response);

}

}

**2.4查询订单**

账户订单查看，先完成ListOrders.jsp的实现，接下来实现Order.java的基础模块，创建OrderDAO接口，完成数据库的具体操作OrderDAOImpl，接下来实现OrderService，成功完成业务层调用，最后完成ViewListOrderServlet实现用户可以从数据库中查看到自己的订单信息。

OrderService

**public** **class** **OrderService** {

**private** **ItemDAO** itemDAO;

**private** **OrderDAO** orderDAO;

**private** **SequenceDAO** sequenceDAO;

**private** **LineItemDAO** lineItemDAO;

**public** **OrderService**(){

itemDAO = **new** **ItemDAOImpl**();

orderDAO = **new** **OrderDAOImpl**();

sequenceDAO = **new** **SequenceDAOImpl**();

lineItemDAO = **new** **LineItemDAOImpl**();

}

**public** **void** **insertOrder**(**Order** order) {

order.setOrderId(getNextId("ordernum"));

**for** (**int** i = 0; i < order.getLineItems().size(); i++) {

**LineItem** lineItem = (**LineItem**) order.getLineItems().get(i);

**String** itemId = lineItem.getItemId();

**Integer** increment = **new** **Integer**(lineItem.getQuantity());

**Map**<**String**, **Object**> param = **new** **HashMap**<**String**, **Object**>(2);

param.put("itemId", itemId);

param.put("increment", increment);

itemDAO.updateInventoryQuantity(param);

}

orderDAO.insertOrder(order);

orderDAO.insertOrderStatus(order);

**for** (**int** i = 0; i < order.getLineItems().size(); i++) {

**LineItem** lineItem = (**LineItem**) order.getLineItems().get(i);

lineItem.setOrderId(order.getOrderId());

lineItemDAO.insertLineItem(lineItem);

}

}

**public** **Order** **getOrder**(**int** orderId) {

**Order** order = orderDAO.getOrder(orderId);

order.setLineItems(lineItemDAO.getLineItemsByOrderId(orderId));

**for** (**int** i = 0; i < order.getLineItems().size(); i++) {

**LineItem** lineItem = (**LineItem**) order.getLineItems().get(i);

**Item** item = itemDAO.getItem(lineItem.getItemId());

item.setQuantity(itemDAO.getInventoryQuantity(lineItem.getItemId()));

lineItem.setItem(item);

}

**return** order;

}

**public** **List**<**Order**> **getOrdersByUsername**(**String** username) {

**return** orderDAO.getOrdersByUsername(username);

}

**public** **int** **getNextId**(**String** name) {

**Sequence** sequence = **new** **Sequence**(name, -1);

sequence = sequenceDAO.getSequence(sequence);

**if** (sequence == **null**) {

**throw** **new** **RuntimeException**("Error: A null sequence was returned from the database (could not get next " + name

+ " sequence).");

}

**Sequence** parameterObject = **new** **Sequence**(name, sequence.getNextId() + 1);

sequenceDAO.updateSequence(parameterObject);

**return** sequence.getNextId();

}

}

ViewListServlet:

**public** **class** **ViewListOrderServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEWLISTORDER = "/WEB-INF/jsp/order/ListOrders.jsp";

**private** **String** username;

**private** **OrderService** orderService;

**private** **List**<**Order**> orderList = **new** **ArrayList**<**Order**>();

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

username = request.getParameter("username");

orderService = **new** **OrderService**();

orderList = orderService.getOrdersByUsername(username);

**HttpSession** session = request.getSession();

session.setAttribute("orderList", orderList);

//HttpSession session = request.getSession();

**Account** account = (**Account**)session.getAttribute("account");

request.getRequestDispatcher(VIEWLISTORDER).forward(request, response);

}

}

1. 订单管理模块，包括购物车管理、新增订单、地址信息等业务功能。

**3.1添加商品到购物车**

先实现Cart.jsp，能够在网页显示购物车栏目信息，接下来具体实现Cart.java使得用户可以创建Cart类，最后创建AddItemToCartServlet，当用户在浏览商品界面时，点击Add to Cart时可以将该商品加入到购物车中并显示相关信息。

AddItemToCartServlet

**public** **class** **AddItemToCartServlet** **extends** **HttpServlet** {

//Servlet的功能即负责中转

//1.处理完请求后的跳转页面

**private** **static** **final** **String** VIEW\_CART = "/WEB-INF/jsp/cart/Cart.jsp";

//2.定义处理该请求所需要的数据

**private** **String** workingItemId;

**private** **Cart** cart; //购物车

//3.是否需要调用业务逻辑层

**private** **CatalogService** catalogService;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

workingItemId = request.getParameter("workingItemId");

**Account** account;

//从对话中，获取购物车

**HttpSession** session = request.getSession();

cart = (**Cart**)session.getAttribute("cart");

account = (**Account**)session.getAttribute("account");

**if**(cart == **null**){

//第一次使用购物车

cart = **new** **Cart**();

}

**if**(cart.containsItemId(workingItemId)){

//已有该物品，数量加一

cart.incrementQuantityByItemId(workingItemId);

}**else**{

catalogService = **new** **CatalogService**();

**boolean** isInStock = catalogService.isItemInStock(workingItemId);

**Item** item = catalogService.getItem(workingItemId);

cart.addItem(item, isInStock);

session.setAttribute("cart", cart);

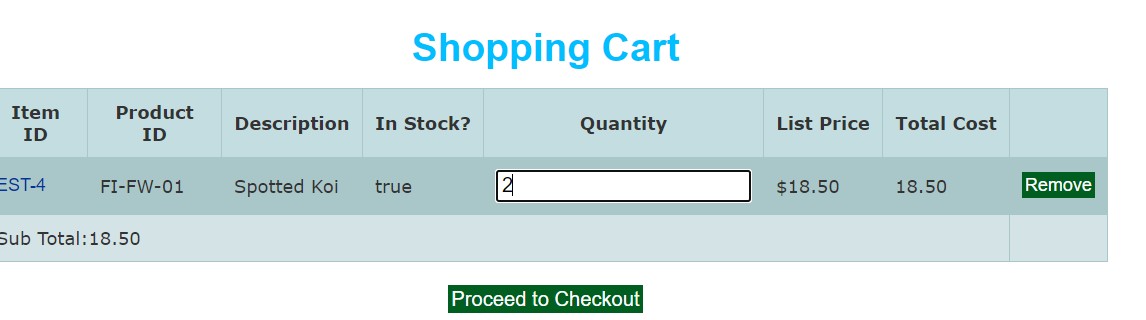
request.getRequestDispatcher(VIEW\_CART).forward(request, response);

}

}

}

**3.2更改购物车商品的数量信息**



在购物车的数量栏输入想要更改后的商品数量，点击Update Cart即可更新购物车中的商品数量、总价。点击之后利用UpdateCartQuantitiesServlet实现信息更新以及页面更新，传递商品的数量信息获取后进行相关操作。

UpdateCartQuantitiesServlet：

**public** **class** **UpdateCartQuantitiesServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEW\_CART = "/WEB-INF/jsp/cart/Cart.jsp";

**private** **String** workingItemId;

**private** **Cart** cart;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

workingItemId = request.getParameter("workingItemId");

**CatalogService** catalogService = **new** **CatalogService**();

//从对话中，获取购物车

**HttpSession** session = request.getSession();

cart = (**Cart**)session.getAttribute("cart");

**Iterator**<**CartItem**> cartItemIterator = cart.getAllCartItems();

**while** (cartItemIterator.hasNext()){

//产品数量增加

**CartItem** cartItem = (**CartItem**)cartItemIterator.next();

**String** itemId = cartItem.getItem().getItemId();

**try** {

**int** quantity = **Integer**.parseInt((**String**) request.getParameter(itemId));

cart.setQuantityByItemId(itemId, quantity);

**if** (quantity < 1) {

cartItemIterator.remove();

}

} **catch** (**Exception** e) {

//ignore parse exceptions on purpose

e.printStackTrace();

}

//CartItem cartItem = cartItemIterator.next();

//cartItem.incrementQuantity();

}

session.setAttribute("cart", cart);

request.getRequestDispatcher(VIEW\_CART).forward(request, response);

}

}

**3.3移除购物车中的商品**

点击购物车中Remove按钮，获取购物车中的信息，之后进入RemoveItemFromCartServlet进行删除操作，并更新该购物车中的信息，如果误操作，则提示该误操作信息。

RemoveItemFromCartServlet：

**public** **class** **RemoveItemFromCartServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEW\_CART = "/WEB-INF/jsp/cart/Cart.jsp";

**private** **static** **final** **String** ERROR = "/WEB-INF/jsp/common/Error.jsp";

**private** **String** workingItemId;

**private** **Cart** cart;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

workingItemId = request.getParameter("workingItemId");

**HttpSession** session = request.getSession();

cart = (**Cart**)session.getAttribute("cart");

**Item** item = cart.removeItemById(workingItemId);

**if**(item == **null**) {

session.setAttribute("message", "Attempted to remove null CartItem from Cart.");

request.getRequestDispatcher(ERROR).forward(request, response);

}**else**{

request.getRequestDispatcher(VIEW\_CART).forward(request, response);

}

}

}

**3.4新增订单——跳转到查看寄件人与收件人地址**

生成订单界面，包括修改地址信息时共有四个页面，第一个页面是购买方的支付与寄件地址信息确认，即NewOrderForm.jsp，接下来确认后是发货人地址与收货人地址信息确认（ConfirmOrder.jsp），或者点击修改地址即可通过ShippingFormServlet进行跳转到修改收件人地址界面，最后点击Comfirm之后即可生成最终订单（ViewOrder.jsp）。

第一个页面在购物车点击Proceed to Checkout之后利用NewOrderFormServlet进行跳转到地址信息显示界面。

NewOrderFormServlet:

**public** **class** **NewOrderFormServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** NEW\_ORDER = "/WEB-INF/jsp/order/NewOrderForm.jsp";

**private** **static** **final** **String** SIGNONFORM = "/WEB-INF/jsp/account/SignonForm.jsp";

**private** **static** **final** **String** ERROR = "/WEB-INF/jsp/common/Error.jsp";

**private** **Account** account;

**private** **Order** order;

**private** **Cart** cart;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**HttpSession** session = request.getSession();

account = (**Account**)session.getAttribute("account");

cart = (**Cart**)session.getAttribute("cart");

**if** (account == **null**){

session.setAttribute("message", "You must sign on before attempting to check out. Please sign on and try checking out again.");

request.getRequestDispatcher(SIGNONFORM).forward(request, response);

} **else** **if**(cart != **null**){

order = **new** **Order**();

order.initOrder(account, cart);

session.setAttribute("order", order);

request.getRequestDispatcher(NEW\_ORDER).forward(request, response);

}**else**{

session.setAttribute("message", "An order could not be created because a cart could not be found.");

request.getRequestDispatcher(ERROR).forward(request, response);

}

}

}

**3.5新增订单——地址信息即修改收件人地址**

在生成订单的第一个界面时有个更改地址的选择框，选择之后点击Continue即可利用ShippingFormServlet跳转到更改收件人地址的界面ShippingForm.jsp,接下里可以进行收货人地址信息更改，再利用ShippingAdressServlet进行数据更新存入数据库接下来使用。

ShippingFormServlet:

**public** **class** **ShippingFormServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** SHIPPINGFORM = "/WEB-INF/jsp/order/ShippingForm.jsp";

**private** **static** **final** **String** SIGNONFORM = "/WEB-INF/jsp/account/SignonForm.jsp";

**private** **Account** account;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**HttpSession** session = request.getSession();

account = (**Account**)session.getAttribute("account");

**if** (account == **null**){

request.getRequestDispatcher(SIGNONFORM).forward(request, response);

} **else**{

request.getRequestDispatcher(SHIPPINGFORM).forward(request, response);

}

}

}

ShippingAdressServlet

**public** **class** **ShippingAddressServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** CONFIRM\_ORDER\_FORM = "/WEB-INF/jsp/order/ConfirmOrder.jsp";

**private** **Order** order;

**private** **String** shipToFirstName;

**private** **String** shipToLastName;

**private** **String** shipAddress1;

**private** **String** shipAddress2;

**private** **String** shipCity;

**private** **String** shipState;

**private** **String** shipZip;

**private** **String** shipCountry;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

shipToFirstName = request.getParameter("shipToFirstName");

shipToLastName = request.getParameter("shipToLastName");

shipAddress1 = request.getParameter("shipAddress1");

shipAddress2 = request.getParameter("shipAddress2");

shipCity = request.getParameter("shipCity");

shipState = request.getParameter("shipState");

shipZip = request.getParameter("shipZip");

shipCountry = request.getParameter("shipCountry");

**HttpSession** session = request.getSession();

order = (**Order**)session.getAttribute("order");

order.setShipToFirstName(shipToFirstName);

order.setShipToLastName(shipToLastName);

order.setShipAddress1(shipAddress1);

order.setShipAddress2(shipAddress2);

order.setShipCity(shipCity);

order.setShipState(shipState);

order.setShipZip(shipZip);

order.setCourier(shipCountry);

session.setAttribute("order", order);

**Account** account = (**Account**)session.getAttribute("account");

request.getRequestDispatcher(CONFIRM\_ORDER\_FORM).forward(request, response);

}

}

**3.6新增订单——确定订单信息**

在第一个地址页面后点击Continue进入第二个地址查看界面（ConfirmOrderFormServlet），确认订单信息的正误。

**public** **class** **ConfirmOrderFormServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** CONFIRM\_ORDER\_FORM = "/WEB-INF/jsp/order/ConfirmOrder.jsp";

**private** **static** **final** **String** SHIPPINGFORM = "/WEB-INF/jsp/order/ShippingForm.jsp";

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

shippingAddressRequired = request.getParameter("shippingAddressRequired");

order = **new** **Order**();

**HttpSession** session = request.getSession();

order = (**Order**)session.getAttribute("order");

**Account** account = (**Account**)session.getAttribute("account");

**if** (shippingAddressRequired == **null**){

request.getRequestDispatcher(CONFIRM\_ORDER\_FORM).forward(request, response);

}

**else**{

shippingAddressRequired = **null**;

request.getRequestDispatcher(SHIPPINGFORM).forward(request, response);

}

}

}

**3.7新增订单——生成最终订单**

最终即可通过ViewOrderServlet生成最终的订单，包含地址信息、商品清单等信息.

ViewOrderServlet:

**public** **class** **ViewOrderServlet** **extends** **HttpServlet** {

**private** **static** **final** **String** VIEWORDER = "/WEB-INF/jsp/order/ViewOrder.jsp";

**private** **static** **final** **String** ERROR = "/WEB-INF/jsp/common/Error.jsp";

**private** **Order** order;

**private** **OrderService** orderService;

**private** **Cart** cart;

**protected** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

doGet(request, response);

}

**protected** **void** **doGet**(**HttpServletRequest** request, **HttpServletResponse** response) **throws** **ServletException**, **IOException** {

**HttpSession** session = request.getSession();

order = (**Order**) session.getAttribute("order");

cart = (**Cart**) session.getAttribute("cart");

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

orderService = **new** **OrderService**();

orderService.insertOrder(order);

session.setAttribute("order", order);

//清空购物车

cart = **null**;

session.setAttribute("cart", cart);

session.setAttribute("message", "Thank you, your order has been submitted.");

request.getRequestDispatcher(VIEWORDER).forward(request, response);

} **else** {

session.setAttribute("message", "An error occurred processing your order (order was null).");

request.getRequestDispatcher(ERROR).forward(request, response);

}

}

}

**//以上三个业务模块，用运行界面截图和核心代码展示功能完成情况。**

1. **实验内容——扩展任务。**
2. 添加验证码功能：在用户注册和用户登录模块中添加验证码功能。

**1.1原理**

先设置字符表即26个字母和各个数字，接下来利用字符表产生随机验证码，同时随机产生120个干扰点（点数适合即可，具体没有特别要求），产生图像，画背景，在不同高度输出验证码的不同字符，设置设置浏览器不要缓存此图片，然后创建内存图像并获得图形，将图像传到客户端，验证码功能即可实现随机显示。

**1.2核心代码**

VerificationCode.java:

/\*实现的核心代码\*/

**public** **void** **doPost**(**HttpServletRequest** request, **HttpServletResponse** response)

**throws** **ServletException**, **IOException** {

response.setContentType("image/jpeg");

**HttpSession** session=request.getSession();

**int** width=60;

**int** height=20;

//设置浏览器不要缓存此图片

response.setHeader("Pragma", "No-cache");

response.setHeader("Cache-Control", "no-cache");

response.setDateHeader("Expires", 0);

//创建内存图像并获得图形上下文

**BufferedImage** image=**new** **BufferedImage**(width, height,**BufferedImage**.TYPE\_INT\_RGB);

**Graphics** g=image.getGraphics();

/\*

\* 产生随机验证码

\* 定义验证码的字符表

\*/

**String** chars="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

**char**[] rands=**new** **char**[4];

**for**(**int** i=0;i<4;i++){

**int** rand=(**int**) (**Math**.random() \*36);

rands[i]=chars.charAt(rand);

}

/\*

\* 产生图像

\* 画背景

\*/

g.setColor(**new** **Color**(**0xDCDCDC**));

g.fillRect(0, 0, width, height);

/\*

\* 随机产生120个干扰点

\*/

**for**(**int** i=0;i<120;i++){

**int** x=(**int**)(**Math**.random()\*width);

**int** y=(**int**)(**Math**.random()\*height);

**int** red=(**int**)(**Math**.random()\*255);

**int** green=(**int**)(**Math**.random()\*255);

**int** blue=(**int**)(**Math**.random()\*255);

g.setColor(**new** **Color**(red,green,blue));

g.drawOval(x, y, 1, 0);

}

g.setColor(**Color**.BLACK);

g.setFont(**new** **Font**(**null**, **Font**.ITALIC|**Font**.BOLD,18));

//在不同高度输出验证码的不同字符

g.drawString(""+rands[0], 1, 17);

g.drawString(""+rands[1], 16, 15);

g.drawString(""+rands[2], 31, 18);

g.drawString(""+rands[3], 46, 16);

g.dispose();

//将图像传到客户端

**ServletOutputStream** sos=response.getOutputStream();

**ByteArrayOutputStream** baos=**new** **ByteArrayOutputStream**();

**ImageIO**.write(image, "JPEG", baos);

**byte**[] buffer=baos.toByteArray();

response.setContentLength(buffer.length);

sos.write(buffer);

baos.close();

sos.close();

session.setAttribute("checkcode", **new** **String**(rands));

}

**public** **void** **init**() **throws** **ServletException** {

// Put your code here

}

}

SigninForm.jsp:

VerificationCode:<input type="text" name="vCode" size="5" maxlength="4"/>

<a href="signOn"><img border="0" src="verificationCode" name="checkcode"></a>

NewAccountForm.jsp:

<tr>

<td>VerificationCode:</td>

<td>

<input type="text" name="vCode" size="5" maxlength="4"/>

<a href="newAccount"><img border="0" src="verificationCode" name="checkcode"></a>

</td>

</tr>

SignOnServlet、NewAccountServlet：

//获得输入的验证码值

**String** value1=request.getParameter("vCode");

/\*获取图片的值\*/

**String** value2=(**String**)session.getAttribute("checkcode");

**Boolean** isSame = **false**;

/\*对比两个值（字母不区分大小写）\*/

**if**(value2.equalsIgnoreCase(value1)){

isSame = **true**;

}

**if**(isSame){

……

//具体登录代码

……

request.getRequestDispatcher(MAIN).forward(request, response);

}**else**{

……

**LogService** logService = **new** **LogService**();

**String** logInfo = logService.logInfo(" ") + strBackUrl + " 注册账号，验证码错误";

logService.insertLogInfo(account1.getUsername(), logInfo);

}

//验证码输入错误后刷新界面，重新生成验证码

request.getRequestDispatcher(NEWACCOUNTFORM).forward(request, response);

}

1. 日志功能：数据库中添加日志信息表，给项目添加日志功能，用户登录后记录用户行为，比如浏览了哪些商品、将商品添加进购物车、生成订单等。

**2.1所要记录的内容**

用户账号、跳转时间、当前操作页面的网址、具体操作页面的名字、操作的商品信息、设计者自定义的具体操作说明。

**2.2记录实现方法**

利用Log.java和HTTPServletRequest的各种方法获取当前页面跳转时的用户账号、跳转时间、当前操作页面的网址、具体操作页面的名字、操作的商品信息，再加上自定义的具体操作说明，传输给LogService的相关方法即可将其插入到数据库中记录下来。

在每次页面跳转之前先获取当前对话中用户是否已登录，如果账号不为空，那么进行获取日志记录操作，并插入到数据库中，之后正常跳转。

如果用户未登录则不记录该操作

**2.3核心代码**

Log.java

**public** **class** **Log** {

**public** **final** **static** **int** OUT\_CONSOLE=1;

**public** **final** **static** **int** OUT\_FILE=2;

**public** **final** **static** **int** OUT\_BOTH=3;

**public** **final** **static** **int** DEBUG\_LEVEL=0;

**public** **final** **static** **int** LOG\_LEVEL=3;

**public** **final** **static** **int** INFO\_LEVEL=6;

**public** **final** **static** **int** WARN\_LEVEL=9;

**public** **final** **static** **int** ERROR\_LEVEL=12;

**public** **final** **static** **int** FATAL\_LEVEL=15;

**protected** **static** **int** level=DEBUG\_LEVEL;// 输出的级别开关，高于指定级别的输出

**protected** **static** **int** out=OUT\_CONSOLE; //输出模式，输出到控制台，文件或者都输出

**public** **static** **String** logdir="log"; // 保存日志文件的目录

**protected** **static** **BufferedWriter** bw =**null**;

**protected** **static** **String** currentDate = "";

**protected** **static** **Date** data=**null**;

**protected** **static** **StringBuilder** bb=**new** **StringBuilder**();

**protected** **static** **String** marsk=" [ERROE] ";

/\*\*

\* 日志函数，不受 level 限制，始终打印，最高级别

\* @param s

\*/

**synchronized** **public** **static** **String** **log**(**Object** ...s) {

bb.delete(0, bb.length());

bb.append(time());

bb.append(marsk);

// 获取调用的位置

//StackTraceElement[] stack = new Exception().getStackTrace();

//bb.append(stack[2].getClassName().replaceAll("\\$\n.$", ""));

bb.append(" ");

/\*

bb.append("::");

bb.append(stack[2].getMethodName());

bb.append("()第[");

bb.append(stack[2].getLineNumber());

bb.append("]行： ");

\*/

**for**(**Object** str : s) {

bb.append(str);

bb.append(" ");

}

//out(bb.toString());

**return** bb.toString();

}

**public** **static** **String** **logInfomation**(**Object** ...s){

marsk = " ";

**return** **log**(s);

}

/\*

public static void debug(Object ...s) {

if(level<=DEBUG\_LEVEL) {

marsk=" [DEBUG] ";

log(s);

}

}

public static void info(Object ...s) {

if(level<=INFO\_LEVEL){

marsk=" [INFO ] ";

log(s);

}

}

public static void warn(Object ...s) {

if(level<=WARN\_LEVEL){

marsk=" [WARN ] ";

log(s);

}

}

public static void error(Object ...s) {

if(level<=ERROR\_LEVEL){

marsk=" [ERROR] ";

log(s);

}

}

public static void fatal(Object ...s) {

if(level<=FATAL\_LEVEL){

//marsk=" [FATAL] ";

marsk = " ";

log(s);

}

}

protected static void out(String s) {

if(out==OUT\_BOTH) {

System.out.println(s);

tofile(s);

}else if(out==OUT\_FILE) {

tofile(s);

}else if(out==OUT\_CONSOLE) {

System.out.println(s);

}

}

protected static void tofile(String s) {

getWriter();

try {

bw.newLine();

bw.write(s);

bw.flush();

}catch(Exception e) {

}

}

\*/

**protected** **static** **BufferedWriter** **getWriter**() {

**if**(currentDate.equals(**new** **SimpleDateFormat**("dd").format(data))) {

**return** bw;

}

**File** fi=**new** **File**(logdir

+**File**.separator+**new** **SimpleDateFormat**("yyyy年").format(data)

+**File**.separator+**new** **SimpleDateFormat**("MM月").format(data)

+**File**.separator+**new** **SimpleDateFormat**("dd").format(data)+"日.log");

**try** {

**if**(!fi.exists()) {

**if**(!fi.getParentFile().exists())

fi.getParentFile().mkdirs();

fi.createNewFile();

}

**FileOutputStream** fiou = **new** **FileOutputStream**(fi,**true**);

**OutputStreamWriter** opw = **new** **OutputStreamWriter**(fiou, "UTF-8");

bw= **new** **BufferedWriter**(opw);

}**catch**(**Exception** e) {

}

**return** bw;

}

**protected** **static** **String** **time**() {

data = **new** **Date**();

**return** **new** **SimpleDateFormat**("yyyy-MM-dd HH:mm:ss").format(data);

}

/\*\*

\* 初始化

\* @param out\_put\_level 日志输出等级,只输出较高等级的 debug < info < warn < error < fatal

\* @param out2where 输出到哪里 OUT\_CONSOLE仅仅控制台输出,OUT\_FILE仅仅输出到文件,OUT\_BOTH均输出

\* @param log\_dir 日志保存到哪里

\*/

**public** **static** **void** **init**(**int** out\_put\_level,**int** out2where,**String** log\_dir) {

level = out\_put\_level;

out = out2where;

logdir = log\_dir;

}

**public** **static** **void** **main**(**String**[] args) {

//Log.debug("这是一个错误","可变长参数","无需拼接字符串");

}

}

LogService:

**public** **class** **LogService** {

**Log** log;

**private** **LogDAO** logDAO;

**public** **LogService**(){

log = **new** **Log**();

logDAO = **new** **LogDAOImpl**();

}

**public** **String** **logInfo**(**Object** ...s){

**return** log.logInfomation(s);

}

**public** **void** **insertLogInfo**(**String** username, **String** logInfo){

logDAO.insertLog(username, logInfo);

}

**//以上完成扩展任务1、3**

1. **小组成员分工及实验总结**
2. 小组分工：

林温健负责商品展示业务模块、用户管理业务模块以及添加验证码功能，江名威负责订单管理模块、日志功能以及撰写试验一实验报告

1. 实验总结

实验试验一在跟着老师完成category等模块后基本上了解了MVC的构架，然后我们分工完成了剩下的部分，第一个商品展示模块较为简单，其次是用户业务管理模块。用户注册时，需要一个全新的包含用户所有信息的输入界面，之后需要先进行设Servlet，其功能仅为实现用户点击注册后跳转用户输入新账号信息的表格，获取网页界面的信息，赋值给一个新账号，再将该账号插入到数据库中。问题主要出现在日志功能的撰写之上，一开始项目能够运行，然而每当登录或者加入购物车之类的动作发生都会报错，并且在Navicat中查看日志表，也会发现有部分内容并未记录在案，按照指示不断理清思路，发现是数据库执行语句sql少了一个变量，以至于总是报错变量超过限制。接下来我们想要完成购物车持久化功能，然后对于完善逻辑有一定的疑虑，是否需要一进入网页便将其购物车放入数据库中，还是一旦用户意图产生加购物车行为，便跳转到用户登录界面，在小组成员进行讨论过后决定采用后者。

通过这次试验，我们首次采用了一个固定的结构，即MVC架构，完成较为复杂的web应用，在相对熟悉这个结构的内容和内在逻辑过后，的确对简化web应用开发有了相当大的益处，同时，MVC架构将前后端分离了，后端的工作也通过多个不同的servlet进行了很精准的细化

**实验二 使用AJAX和jQuery改善用户体验**

1. **实验内容——基本任务。**
2. 账号管理模块。在新建账号和修改账号信息页面上用AJAX技术进行表单验证，如“判断用户名是否已存在”等功能。
3. 查询商品模块。在主页面右上角的查询商品功能中添加自动补全功能。

3. 购物车模块。在购物车模块中修改购买数量时，通过AJAX修改购物车信息。

**1. 账号管理模块。在新建账号信息页面上使用AJAX进行表单验证，实现“判断用户名是否已存在”功能。**



**1.1实现方法**

首先在jsp页面中使用传统的HTML绑定方法使得其能够运行js事务即检测用户名是否已存在，接下来在js代码中通过XML异步通信技术，将该用户名传到servlet中调用业务逻辑功能访问数据库判断该有户名是否已存在，从servlet中返回一个用户区别用户名是否已存在的变量到js中，再到js中根据该变量的值进行相应的输出，即该用户已存在或者可用。

**1.2核心源代码**

NewAccountForm.jsp:

<input type="text" name="username" id="username" onblur="usernameIsExist();"/>

<div id="usernameMsg"></div>

<script type="text/javascript" src="${pageContext.request.contextPath }/js/register.js"></script>

register.js

**function** createXMLHttpRequest(){

…………

}

**function** usernameIsExist() {

**var** username = document.getElementById('username').value;

sendRequest("usernameIsExistServlet?username=" + username);

}

**function** sendRequest(url) {

…………

}

**function** processResponse() {

**if** (xmlHttpRequest.readyState == 4) {

**if** (xmlHttpRequest.status == 200) {

**var** responseInfo = xmlHttpRequest.responseXML.getElementsByTagName("msg")[0].firstChild.data;

**var** div1 = document.getElementById('usernameMsg');

**if** (responseInfo == "Exist") {

div1.innerHTML = "<font color='red'>用户名已存在</font>";

} **else** {

div1.innerHTML = "<font color='green'>用户名可用</font>";

}

}

}

}

UsernameIsExistServlet.java

tring username = request.getParameter("username");

**Account** account = **new** **Account**();

account.setUsername(username);

**AccountService** accountService = **new** **AccountService**();

response.setContentType("text/xml");

**PrintWriter** out = response.getWriter();

**if**(accountService.getAccount(account.getUsername()) != **null**){

out.println("<msg>Exist</msg>");

}**else** {

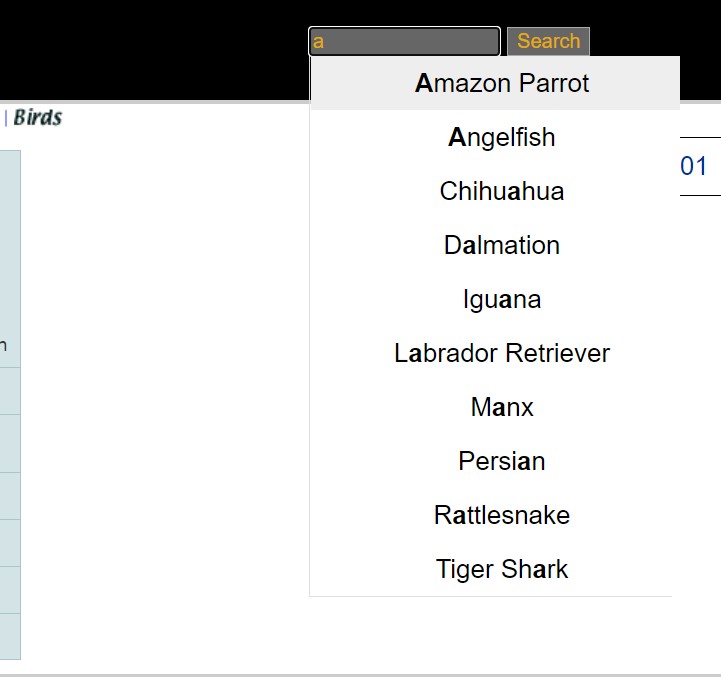
out.println("<msg>NotExist</msg>");

}

out.flush();

out.close();

**2.查询商品模块。在主页面右上角的查询商品功能中添加自动补全功能。**



**2.1实现方法**

查询商品的自动补全功能基本设计流程是一样，本功能的难点在于js中对搜索到的结果的样式和事务处理。

首先在jsp中引入该js文件，接下来js中首先进行获取搜索关键字，然后通过servlet层获取所有的结果，并将结果以字符串的形式返回到js中，最后，js则将结果转化为数组形式然后通过相对应的样式显示在搜索框下方。

js中还需要处理的是对于搜索到的结果进行样式匹配加以区分，当加入关键字或者删除关键字时，搜索到的结果要及时进行变更，搜索栏得到结果中包含的关键字需要进行加粗的样式处理从而能够区分开。

**2.2核心源代码**

searchProduct.css

div**.wrap**{

width: **240px**;

margin: **20px** auto **0** auto;

}

**.auto-inp**{

width: **240px**;

height: **36px**;

padding-left: **.5em**;

border-radius: **2px**;

border: **1px** solid **#dedede**;

color: **#666**;

outline: none;

}

**.auto**{

width: **240px**;

border: **1px** solid **#dedede**;

border-top: none;

position: absolute;

}

**.auto\_out**{

width: **238px**;

height: **36px**;

line-height: **36px**;

padding-left: **.5em**;

color: **#000**;

background: **#fff**;

overflow: hidden;

white-space: nowrap;

-o-text-overflow: ellipsis;

text-overflow: ellipsis;

}

**.auto\_out.on**{

background: **#eee**;

cursor: pointer;

}

**.hidden**{

display: none;

}

searchProduct.js

Array.prototype.unique = **function**(){

**this**.sort();

**var** res = [];

**var** json = {};

**for** (**var** i = 0; i < **this**.length; i++) {

**if**(!json[**this**[i]]){

res.push(**this**[i]);

json[**this**[i]] = 1;

}

}

**return** res;

}

// 对样式操作

**var** setClass = {

hasClass: **function**(elements,cName){ // 判断是否含有某个class

**if**(elements.className.match(**new** RegExp( "(\\s|^)" + cName + "(\\s|$)") ))

**return** **true**;

**else**

**return** **false**;

},

addClass: **function**(elements,cName){ // 添加class

**if**( !**this**.hasClass( elements,cName ) ){

elements.className += " " + cName;

};

},

removeClass: **function**(elements,cName){ // 移除某个class

**if**( **this**.hasClass( elements,cName ) ){

elements.className = elements.className.replace( **new** RegExp( "(\\s|^)" + cName + "(\\s|$)" )," " );

}

}

}

**var** Bind = **function**(This){

**return** **function**(){

This.init();

}

}

**function** AutoComplete(keyword,auto,arr) {

**this**.obj = document.getElementById(keyword);

**this**.autoObj = document.getElementById(auto);

**this**.search\_value = ""; //当前的搜索输入值

**this**.index = -1; //当前选中的DIV的索引

**this**.value\_arr = arr; //数据库中供检索的值 不包含重复值

}

AutoComplete.prototype = {

// 初始化

init: **function**(){

**var** This = **this**;

setClass.removeClass(This.autoObj,"hidden");

**this**.autoObj.style.left = **this**.obj.offsetLeft + "px";

**this**.autoObj.style.top = **this**.obj.offsetTop + **this**.obj.offsetHeight + "px";

},

//删除自动完成需要的所有DIV

deleteDIV: **function**(){

**while**(**this**.autoObj.hasChildNodes()){

**this**.autoObj.removeChild(**this**.autoObj.firstChild);

}

setClass.addClass(**this**.autoObj,"hidden");

},

autoOnmouseover: **function**(index){

**if**(index != **this**.index){

setClass.addClass(**this**.autoObj.children[index],"on");

setClass.removeClass(**this**.autoObj.children[**this**.index],"on");

**this**.index = index;

}

},

setValue: **function**(This){

**return** **function**(){

This.obj.value = **this**.seq;

setClass.addClass(This.autoObj,"hidden");

}

},

// 响应键盘

pressKey: **function**(event){

**var** code = event.keyCode;

**var** length = **this**.autoObj.children.length;

**if**(code == 38){ //↑

setClass.removeClass(**this**.autoObj.children[**this**.index],"on");

**this**.index--;

**if**(**this**.index < 0){

**this**.index = length - 1;

}

setClass.addClass(**this**.autoObj.children[**this**.index],"on");

**this**.obj.value = **this**.autoObj.children[**this**.index].seq;

}**else** **if**(code == 40){ //↓

setClass.removeClass(**this**.autoObj.children[**this**.index],"on");

**this**.index++;

**if**(**this**.index > length-1){

**this**.index = 0;

}

setClass.addClass(**this**.autoObj.children[**this**.index],"on");

**this**.obj.value = **this**.autoObj.children[**this**.index].seq;

}**else**{ //回车

**this**.obj.value = **this**.autoObj.children[**this**.index].seq;

setClass.addClass(**this**.autoObj,"hidden");

**this**.index = -1;

}

},

// 程序入口

start: **function**(event){

event = event || window.event;

**var** code = event.keyCode;

**var** This = **this**;

**if**(code != 13 && code != 38 && code != 40){

**this**.init();

**this**.deleteDIV();

**this**.search\_value = **this**.obj.value;

**var** valueArr = **this**.value\_arr.unique();

//去掉前后空格不能为空

**if**(**this**.obj.value.replace(/(^\s\*)|(\s\*$)/g,"") == ""){ **return**;}

//判断数组中是否含有输入的关键字

**try**{

**var** reg = **new** RegExp("("+ **this**.obj.value +")","i"); //输入"aaa" 则 reg = /(aaa)/i

}**catch**(e){

alert(e.message);

}

**var** div\_index = 0; //记录匹配索引个数

**for** (**var** i = 0; i < valueArr.length; i++) {

**if**(reg.test(valueArr[i])){

**var** div = document.createElement("div");

div.className = "auto\_out";

div.seq = valueArr[i];

div.index = div\_index;

div.innerHTML = valueArr[i].replace(reg,"<strong>$1</strong>");

**this**.autoObj.appendChild(div);

setClass.removeClass(**this**.autoObj,"hidden");

div\_index++;

**if**(div\_index == 1) {

setClass.addClass(**this**.autoObj.firstChild,"on");

**this**.index = 0;

}

div.onmouseover = **function**(){

This.autoOnmouseover(**this**.index);

}

div.onclick = **this**.setValue(This);

}

}

}**else**{

**this**.pressKey(event);

}

window.onresize = Bind(This);

}

}

**var** xmlHttpRequest;

**function** createXMLHttpRequest(){

……

}

**var** keyword = document.getElementById('keyword').value;

sendRequest("findResultServlet?keyword=" + keyword);

**function** sendRequest(url) {

……

}

**function** processResponse() {

**if** (xmlHttpRequest.readyState == 4) {

**if** (xmlHttpRequest.status == 200) {

**var** resp = xmlHttpRequest.responseText;

**var** array = resp.split(",");

**var** autoComplete = **new** AutoComplete("keyword","auto",array);

document.getElementById("keyword").onkeyup = **function**(event){

autoComplete.start(event);

}

}

}

}

searchProductsearvlet.java

//获取搜索框输入的内容

**String** keyword = request.getParameter("keyword");

//向server层调用相应的业务

**CatalogService** service = **new** **CatalogService**();

**List**<**Product**> productList = service.searchProductList(keyword);

response.setContentType("text/xml");

**PrintWriter** out = response.getWriter();

//返回结果

**String** res = "";

**for**(**int** i=0; i<productList.size(); i++){

**if**(i>0){

res += "," + productList.get(i);

}**else**{

res += productList.get(i);

}

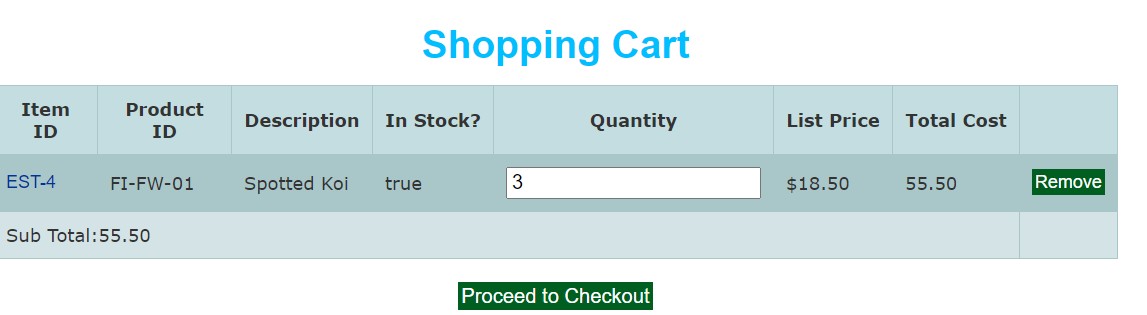
}

out.write(res);

out.flush();

out.close();

**3.购物车模块。在购物车模块中修改购买数量时，通过AJAX修改购物车信息，而不再要求用户点击更新按钮才能更新商品信息。**



**3.1实验方法**

购物车修改数量功能改善前是需要用户点击更新按钮，商品相关信息才会更新。通过AJAX功能改善后用户在数量框输入相应数量后，鼠标离开输入框，商品相对应的数量、总价等信息即会发生改变。

首先，在jsp页面中设置对应的id用户获取js操作的节点，通过HTML绑定调用js中的功能，接下来，在js页面中回去更改商品的数量值，调用servlet更改相对应的信息并将更改后的信息通过字符串传输回来，再在js中对字符串进行处理并显示到jsp页面上。

**3.2核心源代码**

Cart.jsp

<input type="text" id="quantity" onblur="updateCart();" name="${cartItem.item.itemId}" value="${cartItem.quantity}">

<script type="text/javascript" src="${pageContext.request.contextPath }/js/updateCart.js"></script>

cartChange.js & updateCart.js

**function** createXMLHttpRequest(){

……

}

**function** updateCart() {

**var** quantity = document.getElementById("quantity").value;

sendRequest("updateCartJSServlet?quantity="+ quantity);

}

**function** sendRequest(url) {

……

}

**function** processResponse() {

**if** (xmlHttpRequest.readyState == 4) {

**if** (xmlHttpRequest.status == 200) {

**var** resp = xmlHttpRequest.responseText;

**var** array = resp.split(",");

**var** quantity = document.getElementById("quantity");

**var** total = document.getElementById("total");

**var** subtotal = document.getElementById("subtotal");

quantity.innerText = array[0];

total.innerText = array[1];

subtotal.innerText = array[2];

}

}

}

CategoryShowJsServlet.java

//从对话中，获取购物车

**HttpSession** session = request.getSession();

cart = (**Cart**)session.getAttribute("cart");

**Iterator**<**CartItem**> cartItemIterator = cart.getAllCartItems();

**while** (cartItemIterator.hasNext()) {

//产品数量增加

**CartItem** cartItem = (**CartItem**) cartItemIterator.next();

**String** itemId = cartItem.getItem().getItemId();

**try** {

**int** quantity = **Integer**.parseInt((**String**) request.getParameter("quantity"));

cart.setQuantityByItemId(itemId, quantity);

**if** (quantity < 1) {

cartItemIterator.remove();

}

} **catch** (**Exception** e) {

e.printStackTrace();

}

}

session.setAttribute("cart", cart);

**Cart** cart2 = (**Cart**)session.getAttribute("cart");

**Iterator**<**CartItem**> cartItemIterator2 = cart2.getAllCartItems();

**String** quantityAll = "";

**while** (cartItemIterator2.hasNext()) {

//产品数量增加

**CartItem** cartItem2 = cartItemIterator2.next();

**int** quantity2 = cartItem2.getQuantity();

quantityAll += quantity2 + "," + cartItem2.getTotal() + "," + cart2.getSubTotal();

}

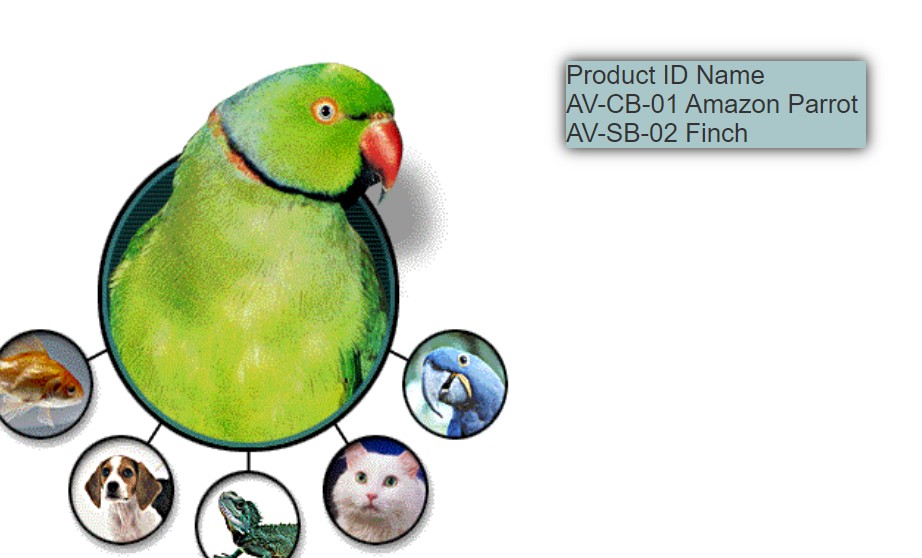
response.setContentType("text/xml");

**PrintWriter** out = response.getWriter();

out.write(quantityAll);

1. **实验内容——选做任务。**
2. 商品展示模块。在首页展示商品的图片上添加AJAX效果，即当鼠标移到某个商品图片上时，用悬浮窗展示该类型商品的信息。
3. 订单模块。通过选项卡或模版内容实现订单多个信息部分的录入，不需要页面跳转，在一个页面完成订单所有内容的录入。 无
4. 其他效果：可自行选择其他UI模版，或选择其他功能点进行用户体验的优化。

#### 1.商品展示模块：在首页展示商品的图片上添加AJAX效果，即当鼠标移到某个商品图片上时，用悬浮窗展示该类型商品的信息，鼠标移开时悬浮窗隐藏。



**1.1实现方法**

商品展示模块的悬浮窗效果就是对jsp中的一个元素当鼠标移动到上时进行显示操作，鼠标移开后进行隐藏操作，同时利用AJAX异步通信显示对应的信息。

首先，在jsp页面进行id、HTML绑定设置，接下来在js中先获取对应categoryId，再通过servlet获取该商品类别对应的产品ID和name作为对应的显示信息，将该信息作为字符串传输回到js中，js中就进行相应显示隐藏操作。未使用框架。

**1.2核心源代码**

IncludeTop.jsp

<div id="inform" style="display: none"></div>

<script type="text/javascript" src="${pageContext.request.contextPath }/js/mouseEvent.js"></script>

<area alt="BIRDS" coords="72,2,280,250" href="viewCategory?categoryId=BIRDS" shape="rect" onmouseover="showInform(alt);" onmouseout="hiddenInform(event)"/>

……

mouseEvent.js

**var** xmlHttpRequest;

**function** createXMLHttpRequest(){

……

}

**function** showInform(categoryId) {

console.log(categoryId);

sendRequest("categoryShowJsServlet?categoryId=" + categoryId);

}

**function** sendRequest(url) {

……

}

**function** processResponse() {

**if** (xmlHttpRequest.readyState == 4) {

**if** (xmlHttpRequest.status == 200) {

**var** resp = xmlHttpRequest.responseText;

//显示悬浮层

**var** inform = document.getElementById("inform");

inform.innerText = resp;

inform.style.display = "block";

}

}

}

//隐藏悬浮层

**function** hiddenInform(event){

**var** informDiv = document.getElementById('inform');

**var** x=event.clientX;

**var** y=event.clientY;

**var** divx1 = informDiv.offsetLeft;

**var** divy1 = informDiv.offsetTop;

**var** divx2 = informDiv.offsetLeft + informDiv.offsetWidth;

**var** divy2 = informDiv.offsetTop + informDiv.offsetHeight;

**if**( x < divx1 || x > divx2 || y < divy1 || y > divy2){

document.getElementById('inform').style.display='none';

}

}

CategoryShowJsServlet.java

**String** categoryId = request.getParameter("categoryId");

**System**.out.println(categoryId+" ddddddddd");

**CatalogService** catalogService = **new** **CatalogService**();

**List**<**Product**> productList = catalogService.getProductListByCategory(categoryId);

**String** resp = "Product ID Name\n";

**int** i = 0;

**while**(i < productList.size()){

**Product** product = productList.get(i);

resp += product.getProductId() + " " + product.getName() + "\n";

i++;

}

response.setContentType("text/xml");

**PrintWriter** out = response.getWriter();

out.write(resp);

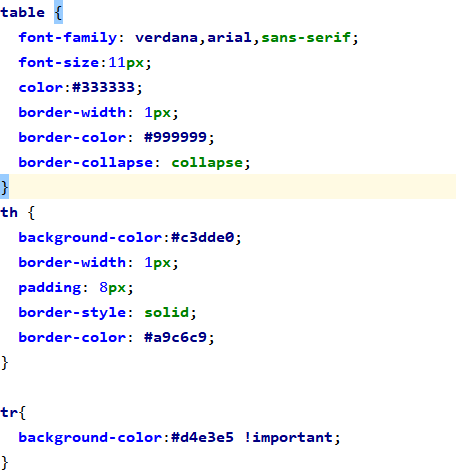
### 2.表格UI设计

**2.1实现方法：修改css样式**

**2.2效果：**

****

**2.3 核心代码：**

****

1. **小组成员分工及实验总结**
2. 小组分工：

江名威为组长负责商品展示模块，查询商品模块，林温健负责账号管理模块，购物车模块，UI设计。

1. 实验总结

实验二在实验一的基础上增加了AJAX技术来改善用户的体验，相对于实验一，难度有一定提升。在完成实验二的基本任务时，我先是根据老师上课讲授的PPT内容，按照步骤一步步向不同文件中添加相应的代码，实现基本任务一：账号检测是否存在。实验的过程并不是一帆风顺的，我添加完代码后开始了调试，发现总体功能并没有实现。经过许久的debug，在浏览器中提示得到的value为0，我反复检查代码，发现问题出在JSP页面的元素的id属性忘记添加，导致从页面中得到元素的值丢失。到头来问题出现在最基础的问题上，而不是AJAX的实现原理，所以做实验时一定要注意一些看起来不起眼的细节，往往是是这些细节会耗费大量时间去检查。在完成接下来的任务时，其基本原理与老师上课讲授基本相同，主要区别是服务器端servel处理的内容不同，返回JS的内容不同，由此导致JSP页面显示的内容区别，来实现相应的功能。在设计网页界面样式时，我先是修改了CSS文件里的样式，刷新后发现浏览器网页中并没有更新。我带着疑问查询了网上的资料，有的说是浏览器缓存的问题，我清除了缓存之后仍然不能解决。最后，我将浏览器改成了chrome，页面就及时更新了，问题原来出在浏览器上。在chrome浏览器上，我通过控制台调试css样式，快速方便地设计了比较好看的css样式，完成了页面修改的任务。

通过这次实现，我们小组的成员们对AJAX技术有了更近一步的认识。经过自己动手实现AJAX，我们巩固了AJAX技术的实现原理知识。总体来说，这次实验让我们对网页有了更深层次的了解。