第四讲 JavaWeb 高级应用

一、EL 表达式

1. EL 表达式简介

① 什么是 EL

el 表达式全称表达式语言，Expression Language

② EL 的功能

作用就是替代 JSP 页面中数据访问时的复杂编码
在 jsp 页面上获取域对象中的数据不仅要取出数据，还要判断取出的数据是否为 null
还要做数据类型转换

③ EL 的特点

自动转换类型
使用简单

2. EL 表达式语法

${ EL expression }

① 从不同域中取出数据

举例：

新建一个 web 工程 ELAndJSTL

添加一个 Servlet：DataServlet

分别向 request 对象，session 对象、application 对象放入字符串

```
HttpSession session = request.getSession();
ServletContext application = this.getServletContext();

request.setAttribute("reqKey", "reqVal");
session.setAttribute("sessionKey", "sessionVal");
application.setAttribute("appKey", "appVal");

request.getRequestDispatcher("a.jsp").forward(request, response);
```

添加 a.jsp 页面

获取这些域对象中的属性

${requestScope.reqKey }

${sessionScope.sessionKey }

${applicationScope.appKey }

- 指定作用域

属性范围	EL 中的名称
page	pageScope，例如${pageScope.username}表示在 page 作用域查找变量 username，找不到返回 null
request	requestScope
session	sessionScope
application	applicationScope

- 不指定作用域

${变量名}

按照 page → request → session → application 的作用域顺序依次查找，找到即返回，最终找不到返回 null

比如：

${appKey}

从 application 中取出值　　取出属性 attribute 的值

② 获取对象的属性值

点操作符

${sessionScope.product.id }

[ ]操作符

${product["id"]}

举例：

复制一个 product 类到当前工程

实例化一个 Product 对象，同时对其中的 id，name 属性赋值，其它属性暂时不管

然后把 product 对象放入到 session 中

```
Product product = new Product();
product.setId(1);
product.setName("aaa");
session.setAttribute("product", product);
```

回到 a.jsp 页面在页面中取出 product 对象中的值

${sessionScope.product.id } --- ${sessionScope.product.name }

注意：EL 对 null 处理

el 表达式不会抛异常，但也不会输出任何内容

③ 执行表达式——关系操作符

关系操作符	示例	结果
==(或 eq)	${23==5}或${23 eq 5} ${"a" =="a"}或${"a" eq "a"}	false true
!=(或 ne)	${23!=5}或${23 ne 5}	true
<(或 lt)	${23<5}或${23 lt 5}	false
>(或 gt)	${23>5}或${23 gt 5}	true

<=(或 le)	${23<=5}或${23 le 5}	false
>=(或 ge)	${23>=5}或${23 ge 5}	true

举例：试着判断下 session 中的 productid 是否等于 1

在 a.jsp 页面中写上如下代码

${sessionScope.product.id == 1 }

④ 执行表达式——逻辑操作符

关系操作符	示例	结果
&&(或 and)	如果 A 为 true，B 为 false，则${ A && B } (或${ A and B })	false
||(或 or)	如果 A 为 true，B 为 false，则${ A || B} (或 ${ A or B })	true
！(或 not)	如果 A 为 true，则${ ！A } (或${ not A })	false

举例：${sessionScope.product.id == 1 && sessionScope.product.name == "aaa"}

⑤ 执行表达式——empty 操作符

若变量 a 为 null，或长度为零的 String，或 size 为零的集合

则${ empty a }返回的结果为 true

${ not empty a }或${ ! empty a }返回的结果为 false

举例：${empty sessionScope.nullkey }

二、JSTL 标签库

1. 什么是 JSTL

JSP 标准标签库（JSP Standard Tag Library）

2. 使用 JSTL 的步骤

- 引入 JSTL 的 jar 文件和标签库描述符文件
- 在 JSP 页面添加 taglib 指令
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
- 使用 JSTL 标签

① out 标签

在 ElAndJstl 工程中，复制 jstl 的 jar 包到 lib 文件夹，然后引入

接下来新建一个 jsp 页面，b.jsp，在页面上添加 taglib 指令
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

使用 c:out 输出标签输出表达式内容<c:out value="${sessionScope.product.id }"/>

运行之前我们可以修改 DataServlet 跳转路径为 b.jsp

如果输出的内容是 null，就在页面上显示，该值不存在，这虽然可以用简单的分支结构搞定，但是有更简单的方法，使用 out 的默认值属性就可以做到
<c:out value="${sessionScope.nullkey }" default="该值不存在"/>

② 条件标签——if

if：实现 Java 语言中 if 语句的功能

```
<c:if test="codition">
    …
</c:if>
```

举例：

```
<c:if test="${sessionScope.product.name == 'aaa' }">
    true
</c:if>
```

③ 条件标签——choose

choose：实现 if - else if - else 语句的功能

```
<c:choose>
    <c:when test="condition1">
        内容 1
    </c:when>
    <c:when test="condition2">
        内容 2
    </c:when>
    ……
    <c:otherwise>
        内容 n
    </c:otherwise>
</c:choose >
```

举例：

如果 id==1 显示 id==1 如果 id==2 显示 id==2

如果 id>=3 显示 id>=3，否则显示 id<=0

先将 session 中的 productid 值取出来，放到一个临时变量去，这个赋值语句，可以使用 set 标签完成

```
<c:set var="id" value="${sessionScope.product.id }"/>
```

然后在 test 表达式中可以使用这个临时变量了

```
<c:choose>
    <c:when test="${id==1 }">
        id==1
    </c:when>
    <c:when test="${id==2 }">
        id==2
    </c:when>
    <c:when test="${id>=3 }">
        id==3
    </c:when>
    <c:otherwise>
        id<=0
    </c:otherwise>
```

</c:choose >

④ 迭代标签——forEach

<c:forEach items="collection" var="varName”begin="start" end="end" step="stepSize" varStatus="status" >

…循环体代码…

</c:forEach>

items 指定要遍历的集合对象

var 指定当前成员的引用

begin 指定从集合的第几位开始

end 指定迭代到集合的第几位结束

step 指定循环的步长

varStatus 属性用于存放 var 引用的成员的相关信息，如索引等

举例 1：遍历集合

先到 servlet 中，制作一个 Product 对象的 List 集合，放入到 session 的 attribute 属性里

```
Product product2 = new Product();
product.setId(2);
product.setName("ccc");
Product product3 = new Product();
product.setId(3);
product.setName("fff");
List<Product> productList = new ArrayList<Product>();
productList.add(product);
productList.add(product2);
productList.add(product3);
session.setAttribute("productList", productList);
```

回到 b.jsp 页面

```
<c:forEach items="${sessionScope.productList }" var="product" varStatus="status">
    ${status.index }    ${product.id }    ${product.name }
    <br/>
</c:forEach>
```

给 items 赋值的就是集合对象 productList

var 就是循环期间存储从集合中取出的元素的临时变量

varstatus 用于存放循环的相关信息，这里取出的就是索引下标

举例 2：循环指定次数

```
<c:forEach  begin="1"  end="5"  step= "2">
    <c:out value="*"></c:out>
</c:forEach>
```

三、案例：优化购物车前端输出

首先把 jar 包以及标签库的引入代码写好

```
<%@  taglib  uri="http://java.sun.com/jsp/jstl/core"  prefix="c"  %>
```

处理 map 集合

```
<c:forEach items="${sessionScope.car }" var="item">
```

将所有的<%=输出语句，改成 el 表达式

```
            <tr>
              <td>
                  <div  class="c_s_img"><img  src="images/c_2.jpg"  width="73"  height="73"
/></div>
                  ${item.key.name }
              </td>
              <td align="center">颜色：灰色</td>
              <td align="center">
                  <div class="c_num">
                      <input  type="button"  value=""  onclick="jianUpdate1(jq(this));"
class="car_btn_1" />
                      <input type="text" value="${item.value }" name="" class="car_ipt" />
                      <input  type="button"  value=""  onclick="addUpdate1(jq(this));"
class="car_btn_2" />
                  </div>
              </td>
              <td align="center" style="color:#ff4e00;">￥ ${item.key.price}</td>
              <td align="center">26R</td>
              <td align="center"><a href="#">删除</a>   <a href="#">加入收藏
</a></td>
            </tr>
            </c:forEach>
```

隔行变色

```
<c:forEach items="${sessionScope.car }" var="item" varStatus="status">
    <c:choose>
        <c:when test="${status.index%2 != 0 }">
            <tr class="car_tr">
        </c:when>
        <c:otherwise>
            <tr>
        </c:otherwise>
</c:choose>
```

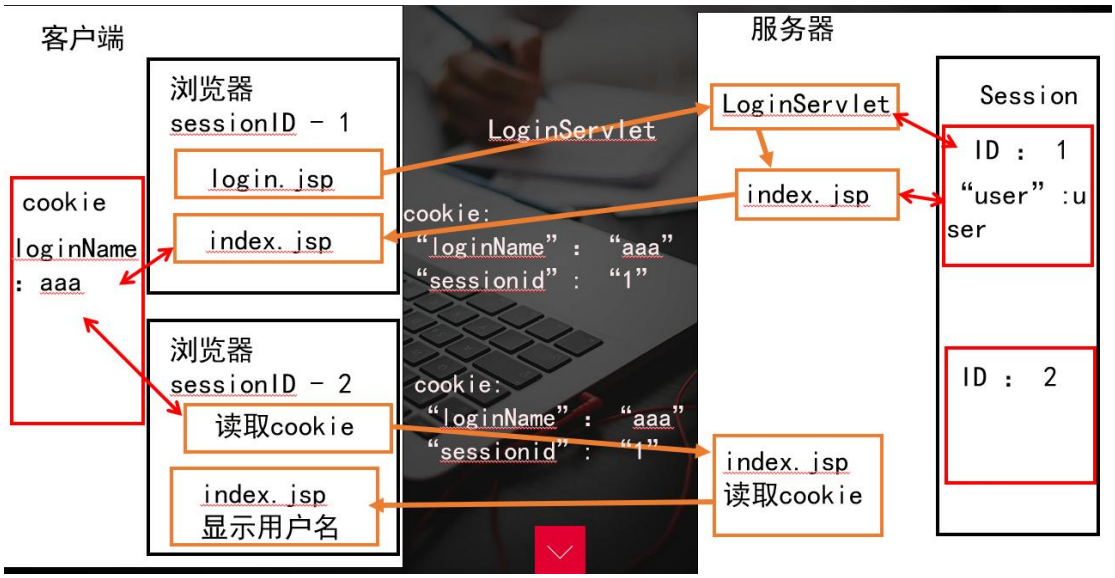
四、cookie 的使用

1. 什么是 cookie

cookie 是 Web 服务器保存在客户端的一系列文本信息

一般作用在什么场合呢？比如我们登录成功后，关闭浏览器，再重新打开浏览器进入网站，发现用户名已经显示在首页上了，但我们并没有做登录操作，系统怎么知道现在打开网页的是之前登录的用户呢

2. cookie 的流程



首先进入登录页面，提交用户名，密码到服务器

服务器验证通过，返回首页

注意，这次返回的信息，除了 index 页面的内容以外，还有 cookie 的数据，一个键值对

浏览器接收到 cookie 后，会写入到 cookie 文件中

关闭浏览器 1，打开浏览器 2

向服务器端请求 index 页面，这时候发送的请求信息里会多出 cookie 的数据，到了服务器端后，index 页面解析 cookie 数据，就能获取到我的登录用户名

接下来再回到浏览器时，index 页面上就能显示上次登录登录的用户名了

所以要完成这个业务场景，只需要搞定两个事情就可以了，

第一从服务器端把 cookie 保存到客户端

第二，在服务器端把客户端传来的 cookie 给读取到

3. 使用 cookie

- 创建 cookie 对象

  Cookie newCookie = new Cookie(String key, String value);
- 写入 cookie

  response.addCookie(newCookie);
- 读取 cookie

  Cookie[] cookies = request.getCookies();

4. cookie 对象的常用方法

方法名称	说明
void setMaxAge(int expiry)	设置 cookie 的有效期，以秒为单位
void setValue(String value)	在 cookie 创建后，对 cookie 进行赋值
String getName()	获取 cookie 的名称
String getValue()	获取 cookie 的值
int getMaxAge()	获取 cookie 的有效时间，以秒为单位

案例：
建一个工程测试一下，cookieprj

添加一个 cookieServlet，在 doGet 方法中，我们创建一个 cookie 对象，并且把这个对象写入到客户端
Cookie cookie=new Cookie("key", "value");
response.addCookie(cookie);

运行这个 servlet
cookie 是写入到客户端的，所以我们应该在本地找到这个 cookie 的数据

怎么找呢，对于 chrome 浏览器可以这样操作

打开 Chrome
进入 Chrome 设置
找到高级 -- 【隐私设置】
进入【内容设置】
在 Cookie 选项中查看所有保存的 Cookie

从列表上可以看出，浏览器是通过域名来区分不同网站的 cookie 的，所以我们点击 localhost，就可以看到有个 key

点进去，内容就是 value，正是我们刚才 Servlet 中设置的 cookie
另外我们从 cookie 中还看到了 sessionid，所以我们的 sessionid 也是存储在 cookie 中的

接下来我们完成读取 cookie 的操作
新建一个 jsp，cookie.jsp
在 cookie.jsp 中写上读取 cookie

```
<%
    Cookie[] cookies = request.getCookies();
    for(Cookie cookie : cookies){
        if(cookie.getName().equals("key")){
            out.println(cookie.getValue());
        }
    }
%>
```

在浏览器上访问 cookie.jsp
很显然能显示 cookie 中的值

接下来我们思考一个问题
如果把浏览器全部关闭，重新访问 cookie.jsp 还能读取到 cookie 中的值吗

实验结果是不能，页面上直接输出空指针异常，为啥呢
因为默认情况下，cookie 是保存在客户端内存中，只要浏览器窗口关闭，cookie 对象也就没了

为了 cookie 能够持久化保存在客户端，能够在多个浏览器窗口间共享
我们需要设置 cookie 的生命周期
所以到服务器端加上一句话，在 Response 之前
假设 cookie 的生命周期我设置为 24 小时
那么 setMaxAge 的参数值就可以设为
24*3600

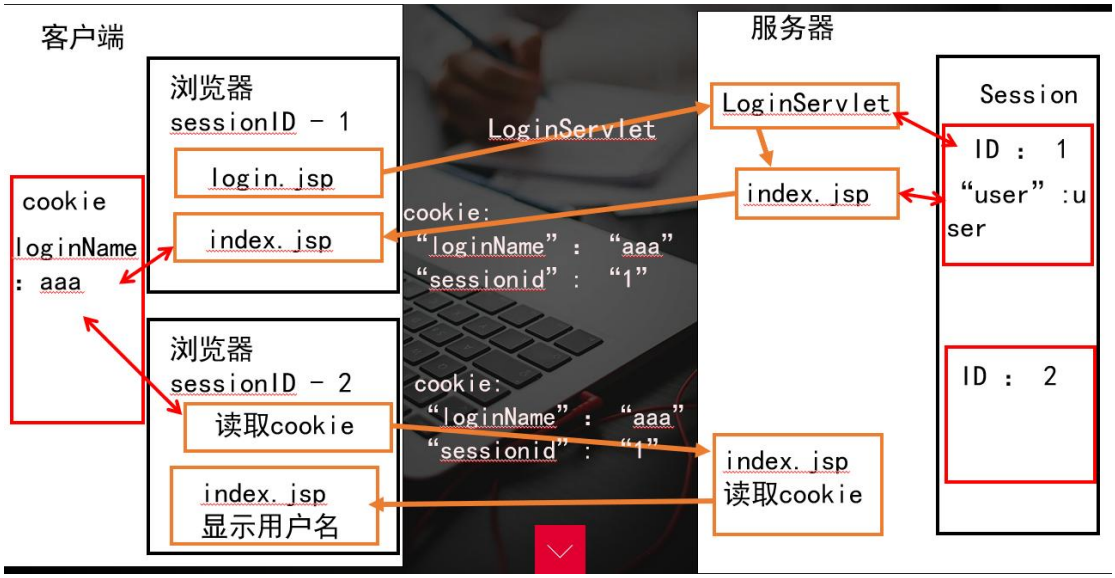
重新运行 cookie，在 chrome 打开 cookie，
可以看到创建时间和到期时间，相差一天

现在把浏览器关闭，重启浏览器，直接访问 cookie.jsp
此时在 cookie.jsp 上显示的就是我们 cookie 中的内容了

最后要补充的是，如果你想删除某个 cookie，只需要设置 cookie 的生命周期为-1 就可以了

大家可以试着把 cookiServlt 中的 cookie 的生命周期设为-1

5. 案例：用户自动登录功能



首先要修改的是 LoginServlet

我们需要把登录的用户信息保存到 session 中

以备后续需要

接下来就是把登录的用户名保存到客户端中

所以打开 LoginServlet，我们添加如下代码

```
HttpSession session = request.getSession();          session.setAttribute("user", user);
Cookie cookie = new Cookie("loginName",loginName);
cookie.setMaxAge(24*3600);
response.addCookie(cookie);
```

接下来回到 index 页面，首先从 session 中取用户信息，如果取不到再试着从 cookie 中取，如果还没有，那就是说在这台机器上，还没有用户登录过这个网站

所以我们先把 session 的流程走完，在来看 cookie 的流程

先替换登录所在的 span，然后加上对 session 中 user 对象的判断

```
<!--End 所在收货地区 End-->
            <span class="fr">
<c:if test="${sessionScope.user==null}">
      <span class="fl">你好，请<a href="login.jsp"    style="color:#ff4e00;">登录</a> <a href="Regist.html" style="color:#ff4e00;">免费注册</a>  </span>
</c:if>
  <c:if test="${sessionScope.user!=null}">
      <span      class="fl"><a      href="#">${sessionScope.user.loginName}</a> | <a href="#">我的订单</a> </span>
      <span class="fl">| <a href="#" >注销</a></span>
</c:if>
<span class="ss">
```

先进入 index 页面，显示需要登录，进入登录页面登录后，再重新打开窗口进入 index 页面，显示用户名

接下来在 index 页面中加入对 cookie 的操作

如果 session 中取出的 user 是 null，那么可以查看 cookie 中是否有登录用户名

```
<c:if test="${sessionScope.user==null}">
            <%
                String loginName = null;
                Cookie[] cookies = request.getCookies();
                if(cookies!=null){
                    for(Cookie cookie : cookies){
                        if(cookie.getName().equals("loginName"))
                        {
                            loginName = cookie.getValue();
                            break;
                        }
                    }
                }
                session.setAttribute("loginName",loginName);
            %>

              <c:if test="${empty sessionScope.loginName }">
                <span  class="fl"> 你 好 ，  请 <a  href="login.jsp"     style="color:#ff4e00;"> 登 录 </a> <a href="Regist.html" style="color:#ff4e00;">免费注册</a>  </span>
```

```
            </c:if>
              <c:if test="${! empty sessionScope.loginName }">
                 <span  class="fl"><a  href="#">${loginName}</a> | <a  href="#"> 我的订单</a> </span>
                 <span class="fl">| <a href="#" >注销</a></span>
              </c:if>
        </c:if>
```

至于为什么把 loginName 放入到 session 中，因为在其它页面中也许会需要用到 loginName

五、案例：完成确认订单页面

创建 Servlet --- confirmOrderServlet

修改 BuyCar.jsp 中的确认结算，超链接地址

```
<a href="ConfirmOrderServlet"><img src="images/buy2.gif" /></a>
```

页面中还需要一些 js 效果，比如修改购买数量，计算总价等功能，这些不是本次视频的重点，有兴趣的同学自己完成，我这里就直接提交到 confirmOrderServlet

在 confirmOrderServlet 中主要去完成两件事

一是确认用户是否已经登录，找出用户的详细信息，比如地址等信息

我这里 只做用户是否登录的判断，其余信息同学自己补充完成

二是计算商品总价

```
// 1、确认用户是否登录
HttpSession session = request.getSession();
if (session.getAttribute("user") == null) {
                request.getRequestDispatcher("login.jsp").forward(request, response);
} else {
// 2、计算总价
Map<Product, Integer> car = (Map<Product, Integer>) session.getAttribute("car");
double totalPrice = 0;
for (Product product : car.keySet()) {
    totalPrice += product.getPrice() * car.get(product);
}
```

```
session.setAttribute("totalPrice", totalPrice);
                request.getRequestDispatcher("BuyCar2.jsp").forward(request, response);
}
```

新建 BuyCar2.jsp,将 BuyCar.html 中的内容复制过去

同时引入 jstl 标签库

在 BuyCar2 的页面上，修改购物车内容，同时修改商品总价

```
    <c:forEach items="${sessionScope.car }" var="item" varStatus="status">
                <c:choose>
                    <c:when test="${status.index%2 != 0 }">
                        <tr class="car_tr">
                    </c:when>
                    <c:otherwise>
                        <tr>
                    </c:otherwise>
                </c:choose>

                    <td>
                        <div class="c_s_img"><img src="images/c_1.jpg" width="73"
height="73" /></div>
                        ${item.key.name }
                    </td>
                    <td align="center">颜色：灰色</td>
                    <td align="center">${item.value }</td>
                    <td align="center" style="color:#ff4e00;">￥${item.key.price }</td>
                    <td align="center">26R</td>
                  </tr>
                  </c:forEach>
                  <tr>
                    <td colspan="5" align="right" style="font-family:'Microsoft YaHei';">
                        商品总价：￥${requestScope.totalPrice } ；  返还积分 56R
                    </td>
                  </tr>
                </table>
```

六、案例：订单提交

添加一个 Servlet ---- CreateOrderServlet ，用作订单生成功能的实现
在 BuyCar2.jsp 页面的下方有个确认订单的超链接，修改此超链接的地址

接下来回到 CreateOrderServlet，回想一下在这个 Servlet 中需要做哪些事情

1、取出下订单的用户信息
2、向订单表中插入订单信息
3、向订单详情表中插入订单详情
4、清除购物车数据

先完成第一步

```
//            1、取出下订单的用户信息
HttpSession session = request.getSession();
if (session.getAttribute("user") == null) {
                request.getRequestDispatcher("login.jsp").forward(request, response);
} else {
      User user = (User) session.getAttribute("user");
}
```

现在考虑第二步和第三步
都需要有 dao 的支持，所以我们向 dao 层插入两个 dao 接口和类
一个是订单详情的，一个是订单表的
复制订单实体类
复制订单详情实体类
看一眼实体类，在订单类中有一个集合关于当前订单对象所对应的详情信息
复制 OrderDao 接口和 OrderDetailDao
复制 OrderDAO 的实现类和 OrderDetailDao 的实现类

接下来是订单生成的业务，这一块的代码逻辑比较复杂
我们就不复制了，带着大家一起写一下吧
先添加 Service 接口

```
public interface OrderService {
```

```
    // 结算订单（返回类型：Order 对象，参数：购物车、总价格、User 对象、收货地址）。
    public Order payShoppingCart(Map<Product,Integer> car,double totalPrice, User user,String address);
}
```

再添加实现类

重点考虑订单生成的步骤：

主要有两步，其余细节暂时不管

1.生成订单

2.生成订单明细

另外还得有事务控制，关于事务控制，我们后面再说

```
public class OrderServiceImpl    implements OrderService {

    /**
     * 结算购物车物品包含以下步骤：
     * 1.生成订单
     * 2.生成订单明细
     * 注意加入事物的控制
     */

    @Override
    public Order payShoppingCart(Map<Product, Integer> car, double totalPrice, User user, String address) {
        // TODO Auto-generated method stub
        Connection connection = null;
        Order order = new Order();
        try {
            connection = DataSourceUtil.openConnection();
            OrderDao orderDao = new OrderDaoImpl(connection);
            OrderDetailDao orderDetailDao = new OrderDetailDaoImpl(connection);
            //增加订单
            order.setUserId(user.getId());
            order.setLoginName(user.getLoginName());
            order.setUserAddress(address);
            order.setCreateTime(new Date());
```

```
                order.setCost(totalPrice);
                order.setSerialNumber(StringUtils.randomUUID());
                orderDao.add(order);
                //增加订单对应的明细信息
                for (Product product : car.keySet()) {
                    OrderDetail orderDetail = new OrderDetail();
                    orderDetail.setOrderId(order.getId());
                    orderDetail.setCost(product.getPrice());
                    orderDetail.setProduct(product);
                    orderDetail.setQuantity(car.get(product));
                    orderDetailDao.add(orderDetail);
                    connection.commit();
                }
            } catch (Exception e) {
                // TODO: handle exception
                e.printStackTrace();
                try {
                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
                order = null;
            } finally {
                try {
                    DataSourceUtil.closeConnection(connection);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            return order;
        }
}
```

为了生成一段随机的订单号，我们复制一个 StringUtils 工具类进来，具体操作照着写就可以了，不用去管具体的实现算法

接下来进入 CreateOrderServlet，调用 OrderService

```
else {
User user = (User) session.getAttribute("user");

//                2、向订单详情表中插入订单详情
//                3、向订单表中插入订单信息
String address = "用户地址";
double totalPrice = Double.parseDouble(session.getAttribute("totalPrice").toString());
Map<Product, Integer> car = (Map<Product, Integer>) session.getAttribute("car");
OrderService orderService = new OrderServiceImpl();
Order order = orderService.payShoppingCart(car, totalPrice, user, address);
request.setAttribute("order", order);
//                4、清除购物车数据
session.removeAttribute("car");
session.removeAttribute("totalPrice");
                request.getRequestDispatcher("BuyCar3.jsp").forward(request, response);
}
```

新建 BuyCar3.jsp，复制 BuyCar3.html 内容进去

用 el 表达式替换订单号，和付款金额

```
请记住您的订单号: <font color="#ff4e00">${requestScope.order.serialNumber}</font>
您的应付款金额为: <font color="#ff4e00">￥${requestScope.order.cost }</font>
```

启动服务器从头开始测试

七、事务处理

以前的 service 方法中操作 dao，只涉及到一条 sql 语句

在 payShoppingCart 这个方法中涉及到至少 2 条 sql

一条是向 order 表中插入订单数据

另一条是向 order_detail 表中插入数据

大家想有没有可能发生这样的问题

在插入 order 表数据之后，操作 order_detail

表的时候，发生了异常

那么这个时候 order 表中是有最新的订单数据
但是它所关联的具体商品的购买信息，没有插入到 order_detail 表中，那么这个订单还是有效订单吗
答案当然是无效的
那么代码上怎么解决这个问题呢

这就得用到事务了，也就是说我们要把操作 order 表和 order_detail 表的 sql 语句看做一个整体，要么所有的 sql 语句全部执行成功，要么就一条不执行

所以事务的一个基本功能作用，就是把一组 sql 当做一个整体来操作

所以这个时候我们就要在原来的代码上添加事务管理功能
相关操作只要照着模板去套好了

第一步首先把 jdbc 中的自动提交功能给关闭了
connection = DataSourceUtil.openConnection();
connection.setAutoCommit(false);
第二步到 try 语句块的最后，如果到此处还没有异常发生，就说明所有的 sql 语句都正常执行了
我们把事务提交给服务器，确认所有的 sql 语句执行有效 connection.commit()
第三步如果发生异常怎么办
如果有异常发生，程序会进入到 catch 语句块
所以我们在 catch 中把已经执行的 sql 语句撤销
这个动作，我们称之为事务回滚
代码为 connection.rollback();

八、案例：会员后台

1. 点击用户名进入用户后台管理页面

在购物街原型中有一个用户管理后台界面 member.html
要进入这个页面，可以点击首页上的用户名进入
现在把它改成 jsp 页面, 完成后, 修改 header.jsp 页面中的用户名链接, 让它跳转到 member.jsp

页面

思考一个问题

要进入 member.jsp 页面，用户必须登录后才能进入
否则是不允许直接直接打开这个 jsp 页面的，要完成这个功能怎么办

最直接的方法是，就是在 member.jsp 页面上添加判断代码
比如：
注意这里判断用户是否登录是通过 session 中有没有 user 对象，cookie 是不管用的，因为 cookie 是存储在客户端，保密性比较差

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:if test="${sessionScope.user==null}">
        <c:redirect url="login.jsp" />
</c:if>
```

Redirect 标签，很显然就是做重定向用的

2. 点击我的订单查看用户的所有订单信息

首页中我的订单链接，应该链接到哪里？
应该是一个 Servlet 对吧，在这个 servlet 中查询当前用户的订单情况
所以我们创建一个 Servlet --- UserOrderServlet

在这个 servlet 中写一些什么代码？
先得知道是哪个用户
然后再去查这个用户的订单情况，对吧

那如果这个用户没有登录，咋办，还是得去跳转到登录页面
所以这里的代码应该写成什么样？

```
HttpSession session = request.getSession();
Object obj = session.getAttribute("user");
if(obj == null){
```

```
                request.getRequestDispatcher("login.jsp").forward(request, response);
}else{
    User user = (User) obj;
//  查询 user 的订单情况,将查询结果放入到 session 中
                request.getRequestDispatcher("Member_Order.jsp").forward(request, response);
}
```

新建一个用户订单页 Member_Order.jsp

修改我的订单 连接，到 UserOrderServlet 中去

九、过滤器

过滤器就是在请求到达目标资源之前，先执行的一段程序代码

新建一个 javaweb 工程 FilterProject

添加一个 Filter -- TestFilter1

过滤器的注解配置不好用，使用 xml 形式

所以删除注解，添加 web.xml

里面写上 TestFilter 的配置，和 Servlet 的配置很相似

```
  <filter>
    <filter-name>TestFilter1</filter-name>
    <filter-class>TestFilter1</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>TestFilter1</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

/*的意思是所有到达服务器的请求都要先经过 TestFilter1 的处理

为了测试我们再添加一个 Servlet -- MyServlet

在 MyServlet 中输出一句话

```
System.out.println("MyServlet");
request.getRequestDispatcher("a.jsp").forward(request, response);
```

再跳转到 a.jsp 页面，所以我们还需要添加一个 a.jsp

```
<h1>a.jsp</h1>
```

修改 TestFilter1 的代码

```
System.out.println("Filter1 Begin");
chain.doFilter(request, response);
System.out.println("Filter1 End");
```

运行 MyServlet
查看运行的输出结果

请求的资源是 MyServlet，但是因为我们的过滤器，是针对所有的请求都要过滤，所以程序的流程先走到了 TestFilter1，执行 Filter 方法，当执行到 chain.doFilter 这个方法的时候，请求将再次转发，转发到我本来要请求的资源 MyServlet 上，当 MyServlet 执行完毕后，程序流程又回到了 TestFilter1，将继续执行 chain.doFilter 方法调用后面的代码

最终生成响应把页面返回给浏览器

这就是过滤器的执行流程

我之前已经说过/*是拦截所有资源的请求，所以如果我直接访问 a.jsp，也会被过滤器拦截的
我们可以试试看
直接输入 a.jsp 的路径，看到页面后，我们在输出结果中可以看到，过滤器又一次的执行了

那有没有办法只过滤访问 jsp 的请求，而不去管访问 servlet 的请求呢

可以的，我们只需要去修改一下拦截的路径就可以了，切换到 web.xml 修改路径

```
  <filter-mapping>
    <filter-name>TestFilter1</filter-name>
    <url-pattern>*.jsp</url-pattern>
  </filter-mapping>
```

这样拦截的就是后缀为 jsp 的请求，servlet 没有后缀，所以不拦截
重启 tomcat，我们测试一下

接下来我们再思考一个问题，过滤器可以有多个吗，答案是可以的

我们再编写一个 TestFilter2

System.out.println("TestFilter2 Begin");

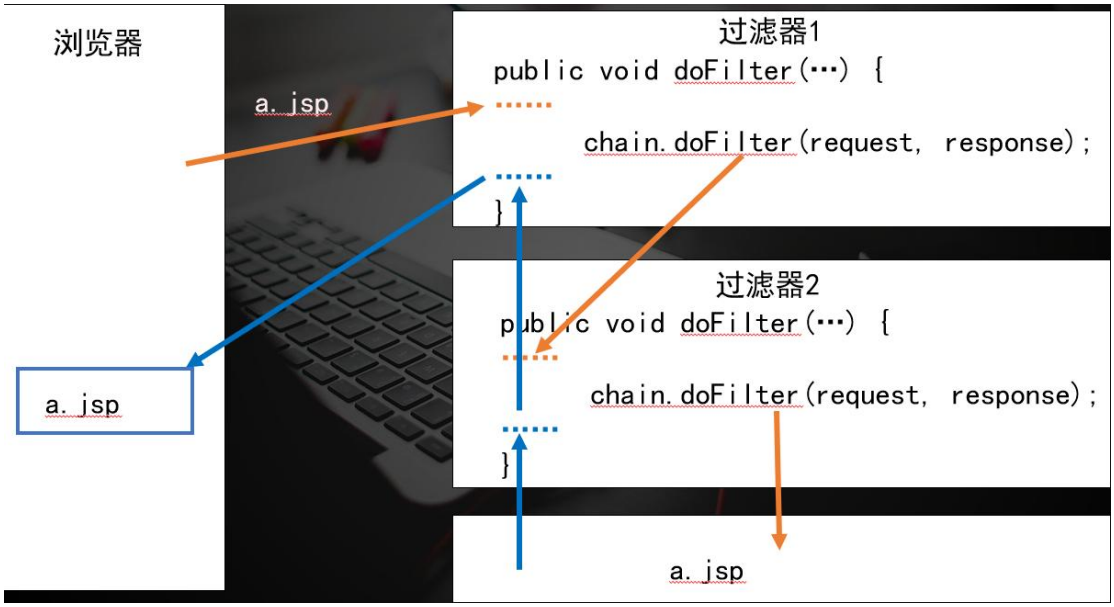
chain.doFilter(request, response);

System.out.println("TestFilter2 End");

在 web.xml 中再配置一下

接下来我们重启 tomcat，请求 a.jsp 看下流程

可以看到两个过滤器都执行了



先执行 Filter1 中 doFilter 方法的前面代码，到了 doFilter，请求转发到 Filter2，再执行 Filter2 中 DoFilter 上面的代码

接下来就去执行目标资源，执行完成后，返回到 Filter2，执行 Filter2 中 dofilter 后面的代码，再执行的就是 Filter1 中 doFilter 后面的代码

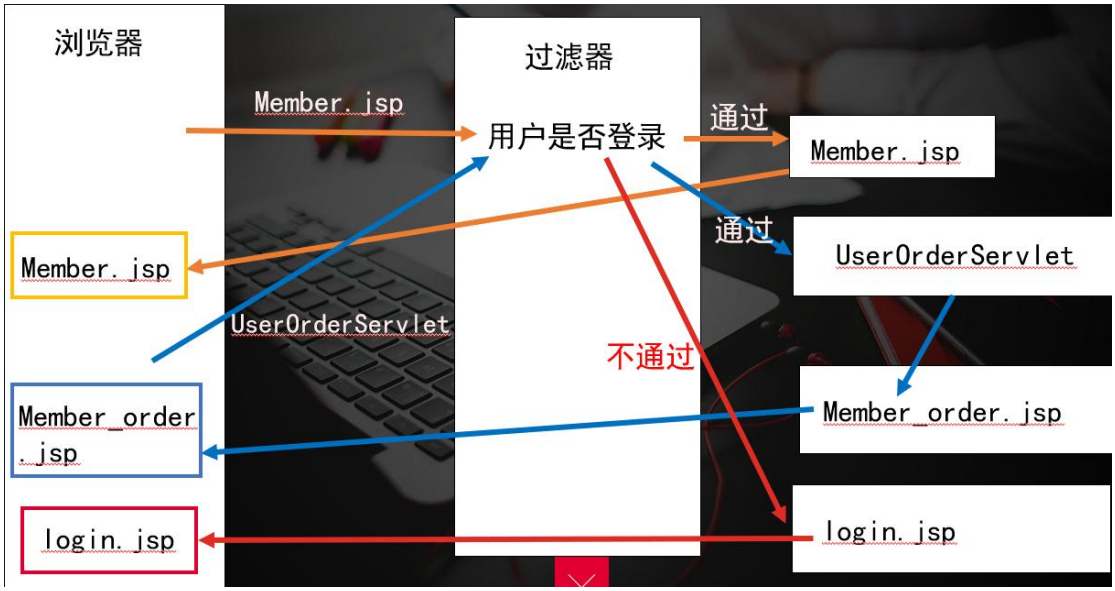
最后将响应返回给浏览器

大家有没有想过为啥先执行的是 Filter1 再执行的是 Filter2

因为我们在 web.xml 中是先配置的 filter1，如果调换二者的位置，重新启动服务器

会发现先执行的就是 Filter2 了

十、实例：统一权限验证



创建一个 LoginValidateFilter

删除注解，添加代码

```
HttpServletRequest httpReq = (HttpServletRequest)request;
HttpSession session = httpReq.getSession();
if(session.getAttribute("user")!=null){
    chain.doFilter(request, response);
}
else{
            request.getRequestDispatcher("login.jsp").forward(request, response);
}
```

这一段代码相信对大家都没什么难度

接下来去配置这个 Filter

这个时候问题出现了，url-pattern 这个属性怎么配置

要知道现在的 member.jsp 等页面资源都设置在根目录下，这个很难和其它页面区分开来，所以我们有必要单独为用户后台的页面，设置一个父目录，和其它页面区分开来

我们新建一个 user 目录将 member.jsp 和 member_order.jsp 页面复制进去，同时打开 UserOrderServlet，我们也要讲这个 servlet 的访问路径和其它 Servlet 区分开来，所以也设置一个父目录 user

这样我们的 url-pattern 就可以写成

```
<filter>
    <filter-name>LoginValidateFilter</filter-name>
    <filter-class>com.shoppingstreet.filter.LoginValidateFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginValidateFilter</filter-name>
    <url-pattern>/user/*</url-pattern>
</filter-mapping>
```

修改 header 中的相关路径

重启服务器，和浏览器，先进入首页
先登录，走正常流程，试一下

然后重新打开浏览器，不登录再走一遍流程

流程上是没有问题了，但是我们的 jsp 页面不能正常显示了，css 样式，等资源路径发生错误了

打开 jsp 页面，所有的 css 文件和 js 文件都使用的是相对路径
现在 member.jsp 的路径是在 User 目录下，很现任这些 css 路径是相对于根目录的，怎么处理呢
一种方式，用我们之前学过的再前面加上../它就能回到根目录下去了，还有一种写法，就是让 jsp 页面上的所有相对路径的起始路径都是从项目的根目录开始，这样就不用每次改目录的时候还需要改所有的相对路径了

怎么操作呢，打开 member.jsp 页面添加这几行代码

```
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>
```

此时 basePath 的值就是 localhost:8080/shoppingstreet
也就是工程的根目录，而且可以看到它是动态获取的，如果以后有变动，比如域名不是 Localhost 了，也不会有什么问题

然后再<head>标签下加上 base 标签

<base href="<%=basePath%>">

这样这个页面上的所有相对路径都是相对于我们项目的根目录了，我们一起测试下好了

测试过程中发现了问题 login 页面的路径又不对了，这是因为过滤器那边也出现了问题，那么我们使用重定向好了

并且配上绝对路径

这样就不会有问题了

```
String path = httpReq.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
HttpServletResponse httpResp = (HttpServletResponse) response;
httpResp.sendRedirect(basePath+"login.jsp");
```

之后也一样，如果你觉得相对路径不好解决的时候，可以考虑使用绝对路径