第十二章：会话

第1节：简介

> 客户端服务器连接开始，请求和响应的过程中都属于一次会话。客户端和服务器交流的过程，这个过程是一个持续的过程，当关闭客户端或者服务器 该次会话结束。再次过程中都属于同一次会话，不管访问时哪个页面或者 哪个Servlet都属于该会话
>
> 一次会话指的是：就好比打电话，A给B打电话，接通之后，会话开始，直到挂断电话，该次会话就结束了，而浏览器访问服务器，就跟打电话一样，浏览器A给服务器发送请求，访问web程序，该次会话就已经接通，其中不管浏览器发送多少请求(就相当于接通电话后说话一样)，都视为一次会话，直到浏览器关闭，本次会话结束。其中注意，一个浏览器就相当于一部电话，如果使用火狐浏览器，访问服务器，就是一次会话了，然后打开google浏览器，访问服务器，这是另一个会话，虽然是在同一台电脑，同一个用户在访问，但是，这是两次不同的会话。

第2节：作用

> 因为HTTP协议是无状态的，所以很显然服务器不可能知道我们已经在上一次的HTTP请求中通过了验证。当然，最简单的解决方案就是所有的请求里面都带上用户名和密码，这样虽然可行，但大大加重了服务器的负担（对于每个request都需要到数据库验证），也大大降低了用户体验(每个页面都需要重新输入用户名密码，每个页面都带有登录表单)。既然直接在请求中带上用户名与密码不可行，那么就只有在服务器或客户端保存一些类似的可以代表身份的信息了，所以就有了cookie与session，cookie和session用来跟踪用户的整个会话。
>
> 在程序中，会话跟踪是很重要的事情。理论上，一个用户的所有请求操作都应该属于同一个会话，而另一个用户的所有请求操作则应该属于另一个会话，二者不能混淆。例如，用户A在超市购买的任何商品都应该放在A的购物车内，不论是用户A什么时间购买的，这都是属于同一个会话的，不能放入用户B或用户C的购物车内，这不属于同一个会话。
>
> 而Web应用程序是使用HTTP协议传输数据的。HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话。即用户A购买了一件商品放入购物车内，当再次购买商品时服务器已经无法判断该购买行为是属于用户A的会话还是用户B的会话了。要跟踪该会话，必须引入一种机制。Cookie和session就是这样的一种机制。它可以弥补HTTP协议无状态的不足。

第3节：使用方式

3.1 Cookie

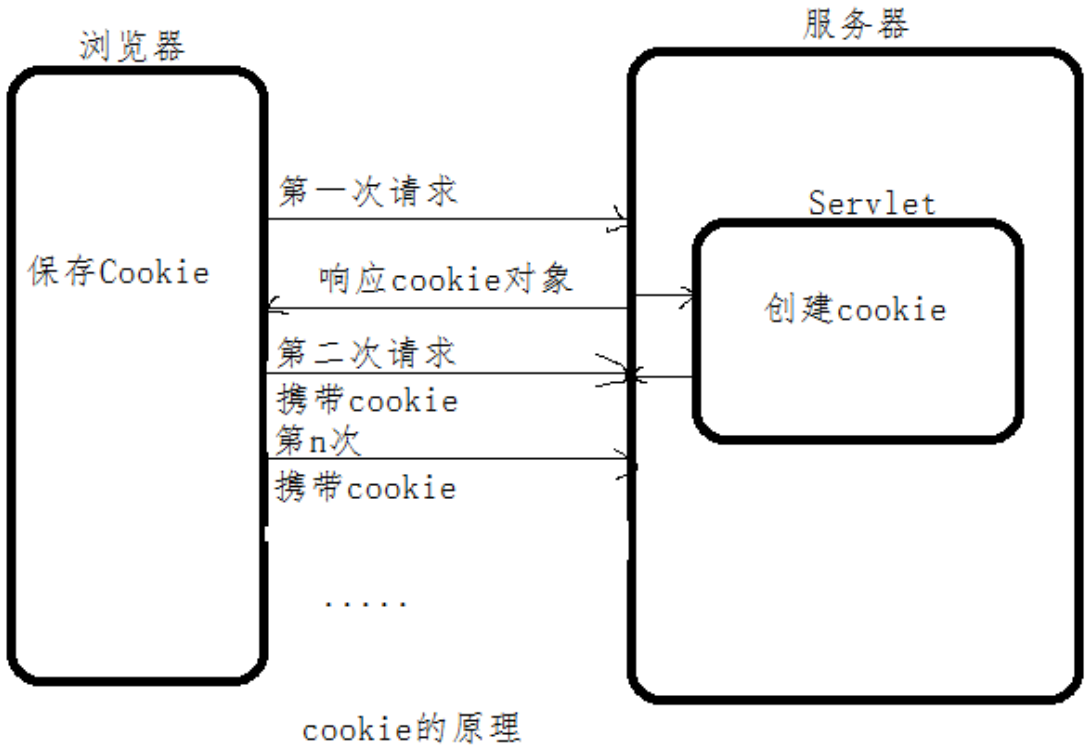
3.1.1 理解

> Cookie：本意是小点心 小甜点 小饼干。
> 创建一个cookie，即servlet发送到Web浏览器的少量信息，由浏览器保存，然后发送回服务器。cookie的值可以唯一标识客户端，因此cookie通常用于会话管理。

3.1.2 原理

> 那么如何跟踪会话呢，我们需要掌握该会话的原理就清楚了：
> 原理：
> Cookie是解决HTTP无状态性的有效手段，服务器可以设置或读取Cookie中所包含的信息。当用户登录后，服务器会发送包含登录凭据的Cookie到用户浏览器客户端，而浏览器对该Cookie进行某种形式的存储（内存或硬盘）。用户再次访问该网站时，浏览器会发送该Cookie（Cookie未到期时）到服务器，服务器对该凭据进行验证，合法时使用户不必输入用户名和密码就可以直接登录。
> 本质上讲，Cookie是一段文本信息。客户端请求服务器时，如果服务器需要记录用户状态，就在响应用户请求时发送一段Cookie信息。客户端浏览器保存该Cookie信息，当用户再次访问该网站时，浏览器会把Cookie做为请求信息的一部分提交给服务器。服务器检查Cookie内容，以此来判断用户状态，服务器还会对Cookie信息进行维护，必要时会对Cookie内容进行修改。

原理图如下：



cookie的原理

> 简而言之我们可以这么理解：
> 1. 当从浏览器发送请求到服务器后，servlet创建cookie，保存少量数据，发送浏览器。
> 2. 浏览器获得服务器发送的cookie数据，将自动的保存到浏览器端。
> 3. 下次访问时，浏览器将自动携带cookie数据发送给服务器。

浏览器查看Cookie的方式：

3.1.3 使用

```
1.Cookie创建
Cookie cookie = new Cookie(String key,String value);
key:表示cookie的名字
value:表示cookie中保存的数据

2.设置有效时间  不受浏览器关闭的影响
// 为何要设置有效时间呢？因为浏览器关闭后该会话就结束了，因此要设置有效时间来保证浏览器关闭后Cookie会话还存在,单位为秒
cookie.setMaxAge(int expiry);
//默认值为 -1 ： 代表浏览器关闭后，也就是会话结束后，cookie就失效了，也就没有了；
//expiry>0 :代表浏览器关闭后，cookie不会失效，仍然存在。并且会将cookie保存到硬盘中，直到设置时间过期才会被浏览器自动删除；
//expiry=0：删除cookie。不管是之前的expiry=-1还是expiry>0，当设置expiry=0时，cookie都会被浏览器给删除。

3. 设置有效路径
//有效路径  默认路径 ： 创建cookie的Servlet的上一层路径
cookie.setPath("/应用名/xx");
//哪些Servlet可以获取该cookie： 访问路径为/应用名/xx以及其子路径都可以访问到

4. 发送给浏览器
response.addCookie(cookie);

5. 获取cookie
Cookie[] cookies = request.getCookies();
if(cookies!=null) {
```

```
  for (Cookie cookie : cookies) {
    // 获取cookie的名称
    String name = cookie.getName();
    // 获取cookie的值
    String value = cookie.getValue();
    System.out.println(name+": : : "+value);
   }
}
```

3.1.4 特点

1、Cookie中只能以键值对的形式保存字符串类型的数据
2、Cookie中保存的数据有大小限制，一般不超过4KB
3、Cookie是保存在客户端的，安全性较差
4、Cookie默认有效期是浏览器关闭即销毁，所以如果想持久保存Cookie对象，一定要设置有效期，并且有效期时间单位是秒。

3.1.5 使用场景

1、记住用户名
登录时，在服务器端获取到用户名，然后创建一个cookie，将用户名存入cookie中，发送回浏览器端，然后浏览器下次在访问登录页面时，先拿到cookie，将cookie中的信息拿出来，看是否保存了该用户名，如果保存了，那么直接用他，如果没有，则自己手写用户名。cookie需要将所有信息都保存在客户端。因此cookie存在着一定的安全隐患，例如本地cookie中保存的用户名和密码被破译，或cookie被其他网站收集。

2、历史记录
比如购物网站，都会有我们的浏览记录的，实现原理其实也是用cookie技术，每浏览一个商品，就将其存入cookie中，到需要显示浏览记录时，只需要将cookie拿出来遍历即可。

3.1.6 案例：记住账号

我们通过代码演示使用场景中的第一种需求：
代码示例：
SaveUserServlet代码：

```
@WebServlet("/login")
public class SaveUserServlet extends HttpServlet {
 @Override
 protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
     //解决中文乱码处理
     req.setCharacterEncoding("utf-8");
     resp.setContentType("text/html;charset=utf-8");
```

```
        //模拟用户的登录场景,假设给定用户名和密码   用户名:admin    密码:123456
        //1.获取表单数据
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        //2.登录判断
        if("admin".equals(username)&&"123456".equals(password)){
            //登录成功-->将用户的信息保存到Cookie中，下次登录时不需要填写用户名及密码了
            Cookie c1 = new Cookie("name",username);
            Cookie c2 = new Cookie("pwd",password);
            //设置生存时间
            c1.setMaxAge(60*60*24*7);
            c2.setMaxAge(60*60*24*7);
            //返回给浏览器保存
            resp.addCookie(c1);
            resp.addCookie(c2);
            resp.sendRedirect("login_success.jsp");
        }else{
            //登录失败,重新登录，并给提示
            req.setAttribute("msg","用户名或者密码不正确");
            req.getRequestDispatcher("login.jsp").forward(req,resp);
        }


    }
}
```

登录页面login.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>登录页面</title>
</head>
<body>
 <%
        //获取浏览器Cookie
        Cookie[] cookies = request.getCookies();
        if(cookies!=null){
            //遍历每一个Cookie找到保存用户信息的Cookie
            for(Cookie c:cookies){
                if("name".equals(c.getName())){
                    //获取cookie中保存的用户名
                    String uname = c.getValue();
                    //保存到page共享域中
                    pageContext.setAttribute("username",uname);
                }
                if("pwd".equals(c.getName())){
                    String upwd = c.getValue();
                    pageContext.setAttribute("password",upwd);
                }
```

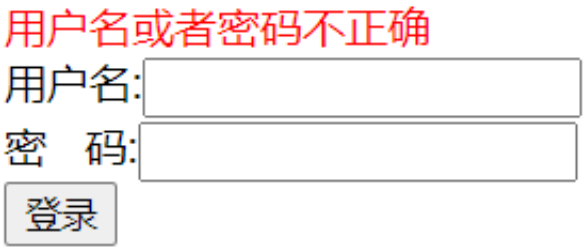
```
            }

        }
 %>
 <form action="login">
     <span id="msg" style="color: red">${msg}</span> <br>
     用户名:<input type="text" name="username" value="${username}"/> <br>
     密   码:<input type="password" name="password"
value="${password}"/> <br>
     <input type="submit" value="登录">
 </form>
</body>
</html>
```

登录成功页面 login_success.jsp

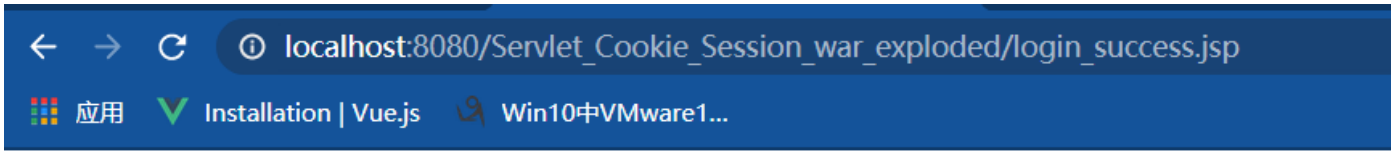
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>登录成功页面</title>
</head>
<body>
 欢迎用户登录成功!!!
</body>
</html>
```

演示效果：

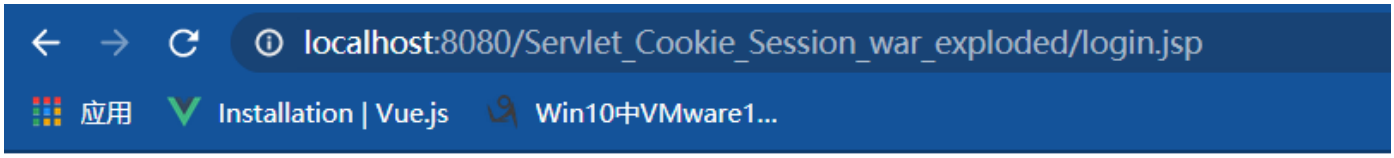
登录失败效果：



登录成功后再一次访问登录页面的效果



欢迎用户登录成功!!!



用户名:admin
密　码:••••••
登录

3.2 Session

3.2.1 理解

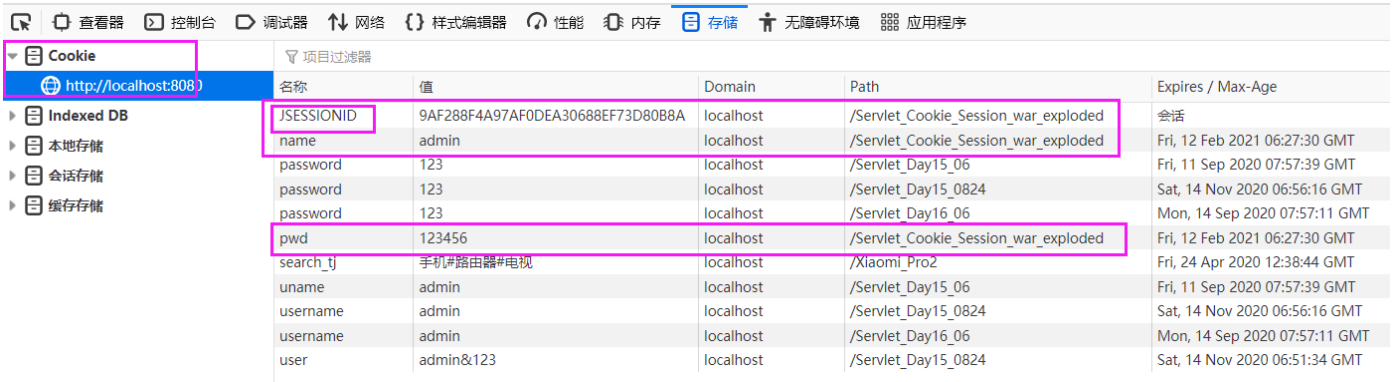
> 可以跨多个页面请求或访问网站来识别用户，并存储有关该用户的信息。servlet容器使用此接口在HTTP客户端和HTTP服务器之间创建会话。会话将在指定的时间段内持续，跨越用户的多个连接或页面请求。一个会话通常对应于一个用户，该用户可能会多次访问一个站点。服务器可以通过多种方式维护会话，例如使用cookie或重写URL。

3.2.2 原理

> 首先浏览器请求服务器访问web站点时，程序需要为客户端的请求创建一个session的时候，服务器首先会检查这个客户端请求是否已经包含了一个session标识、称为SESSIONID，如果已经包含了一个sessionid则说明以前已经为此客户端创建过session，服务器就按照sessionid把这个session检索出来使用，如果客户端请求不包含session id，则服务器为此客户端创建一个session并且生成一个与此session相关联的session id，sessionid 的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个sessionid将在本次响应中返回到客户端保存，保存这个sessionid的方式就可以是cookie，这样在交互的过程中，浏览器可以自动的按照规则把这个标识发回给服务器，服务器根据这个sessionid就可以找得到对应的session

原理图如下：

> 在运行记住密码项目时，通过浏览器查看我们发现Cookie中保存处理用户名和密码外，还有一个JSESSIONID，这个就是发送请求时创建的session对象对应的id。



3.2.3 使用

```
1.获取session对象
HttpSession session = request.getSession();
2. 保存数据到session中
session.setAttribute(String 绑定名, Object 绑定值);
3. 获取session数据
Object   session.getAttribute("绑定名");
4. 有效时间 单位秒  默认30分钟，可以去tomcat中conf下web.xml文件查看
session.setMaxInactiveInterval(int expiry);
5. 销毁session对象
session.invalidate();
```

3.2.4 特点

1、存储在服务器端
2、默认超时时间为30分钟
3、Session做为第二大域对象，在一个会话范围内的数据是可以共享的
4、Session存储的数据可以是任意类型

3.2.5 使用场景

1. 用户登录时的安全校验
2. 购物车

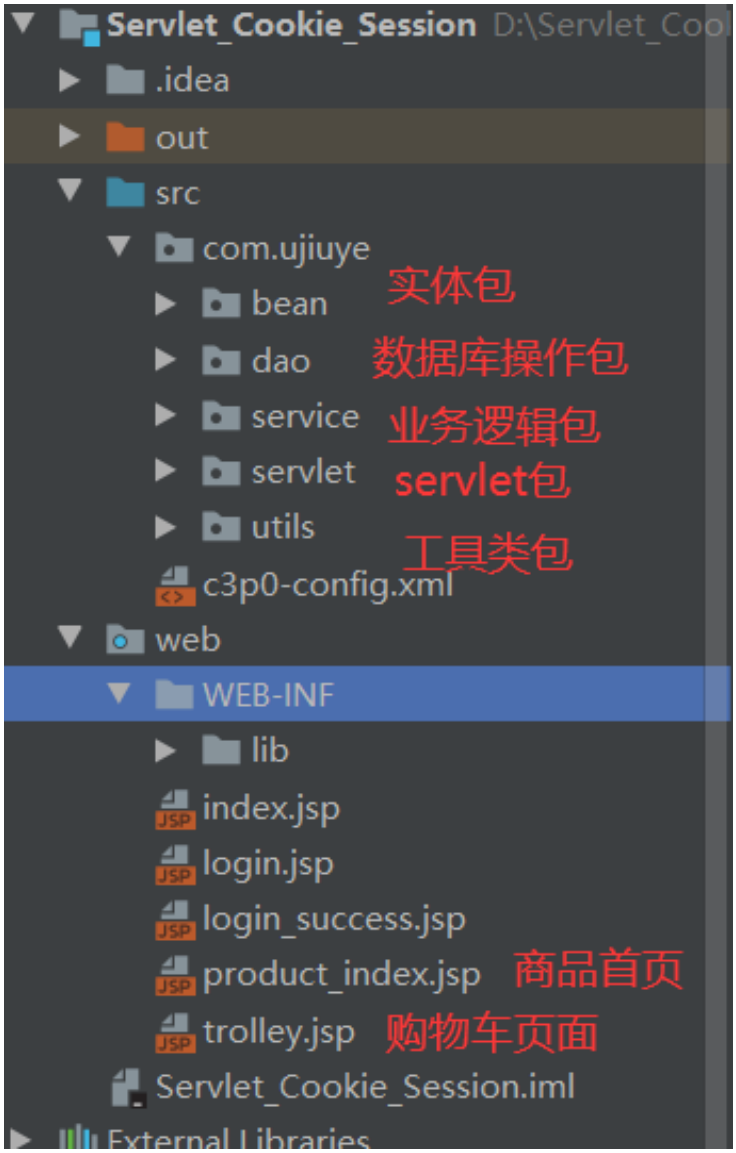
3.2.6 案例：购物车

需求：展示所有商品，通过商品操作将该商品添加到购物车，如图：

商品展示

商品编号	商品名称	商品价格	商品描述	上架时间	添加购物车
1	小米10	2000.0	还不错	2021-02-05	加入购物车
2	华为荣耀	1500.0	嗯，可以	2021-02-05	加入购物车

项目结构如下：



代码:

数据库product

```
##商品表创建
CREATE TABLE `product` (
  `pid` int(11) NOT NULL,
  `pname` varchar(200) DEFAULT NULL,
  `price` decimal(10,0) DEFAULT NULL,
  `pdesc` varchar(200) DEFAULT NULL,
  `pdate` date DEFAULT NULL,
  PRIMARY KEY (`pid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

##数据添加
insert  into `product`(`pid`,`pname`,`price`,`pdesc`,`pdate`) values
(1,'小米10',2000,'还不错','2021-02-05'),
(2,'华为荣耀',1500,'嗯，可以','2021-02-05');
```

实体类Product

```
package com.ujiuye.bean;
```

```java
import java.util.Date;

public class Product {
    private int pid;

    private String pname;

    private double price;

    private String pdesc;

    private Date pdate;

    public Product() {

    }

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public String getPname() {
        return pname;
    }

    public void setPname(String pname) {
        this.pname = pname;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }


    public String getPdesc() {
        return pdesc;
    }

    public void setPdesc(String pdesc) {
        this.pdesc = pdesc;
```

```
    }

    public Date getPdate() {
        return pdate;
    }

    public void setPdate(Date pdate) {
        this.pdate = pdate;
    }

}
```

工具类的编写 JdbcUtils

```
package com.ujiuye.utils;

import org.apache.commons.dbutils.QueryRunner;
import com.mchange.v2.c3p0.ComboPooledDataSource;

public class JdbcUtils {
  private static ComboPooledDataSource ds = null;
  public static QueryRunner qr = null;
  static{
    ds = new ComboPooledDataSource();
    qr = new QueryRunner(ds);
  }
}
```

数据库操作 ProductDao:

```
package com.ujiuye.dao;

import com.ujiuye.bean.Product;
import com.ujiuye.utils.JdbcUtils;
import org.apache.commons.dbutils.handlers.BeanHandler;
import org.apache.commons.dbutils.handlers.BeanListHandler;

import java.sql.SQLException;
import java.util.List;

public class ProductDao {
    //查询全部的商品
    public List<Product> queryProduct(){
        List<Product> list = null;
        try {
            list = JdbcUtils.qr.query("select * from product",new BeanListHandler<>
(Product.class));
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return list;
    }
    //查询单个商品信息
    public Product queryProductById(String id){
        Product p = null;
        try {
           p = JdbcUtils.qr.query("select * from product where pid = ?",new
BeanHandler<>(Product.class),id);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return p;
    }
}
```

业务逻辑ProductService:

```
package com.ujiuye.service;

import com.ujiuye.bean.Product;
import com.ujiuye.dao.ProductDao;

import java.util.List;

public class ProductService {
    ProductDao pd = new ProductDao();
    //查询全部的商品
    public List<Product> queryProduct(){
        return pd.queryProduct();
    }
    //查询单个商品信息
    public Product queryProductById(String id){
        return pd.queryProductById(id);
    }
}
```

首页商品展示Servlet:ProductIndexServlet

```
package com.ujiuye.servlet;

import com.ujiuye.bean.Product;
import com.ujiuye.service.ProductService;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;
```

```
@WebServlet("/index")
public class ProductIndexServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //处理中文乱码
        req.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");
        ProductService ps = new ProductService();
        List<Product> products = ps.queryProduct();
        //数据绑定并转发到商品首页
        req.setAttribute("list",products);
        req.getRequestDispatcher("product_index.jsp").forward(req,resp);

    }
}
```

添加购物车Servlet:

```
package com.ujiuye.servlet;

import com.ujiuye.bean.Product;
import com.ujiuye.service.ProductService;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

@WebServlet("/addTrolley")
public class AddTrolleyServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //处理中文乱码
        req.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");
        // 1. 获取请求参数
        String pid = req.getParameter("pid");
        // 2. 调用业务层 处理业务
        ProductService ps = new ProductService();
        Product product = ps.queryProductById(pid);
        // 3. 将当前商品保存到session
        HttpSession session = req.getSession();
        session.setMaxInactiveInterval(60*60);
        // 关闭浏览器  再访问 仍然有数据
        Cookie cookie = new Cookie("JSESSIONID", session.getId());
```

```
            cookie.setMaxAge(60*60);
            resp.addCookie(cookie);
            // 3.1 从session中获取一下是否存在list集合
            List<Product> list = (List<Product>) session.getAttribute("list");
            if(list==null) {
                // session中没有  创建一个list
                list = new ArrayList<>();
            }
            list.add(product);
            session.setAttribute("list", list);
            // 4. 转发或者重定向到购物车页面
            req.getRequestDispatcher("trolley.jsp").forward(req, resp);
        }
    }
```

演示效果：

购物车展示

商品编号	商品名称	商品价格	商品描述	上架时间
1	小米10	2000.0	还不错	2021-02-05
2	华为荣耀	1500.0	嗯，可以	2021-02-05
1	小米10	2000.0	还不错	2021-02-05
2	华为荣耀	1500.0	嗯，可以	2021-02-05
2	华为荣耀	1500.0	嗯，可以	2021-02-05
1	小米10	2000.0	还不错	2021-02-05
2	华为荣耀	1500.0	嗯，可以	2021-02-05
2	华为荣耀	1500.0	嗯，可以	2021-02-05
1	小米10	2000.0	还不错	2021-02-05

以上案例仅仅是为了演示Session对象的使用，购物车功能还有很多还需要待完善。

3.3 区别

1. Cookie和Session都是会话技术，Cookie是运行在客户端，Session是运行在服务器端。
2. Cookie有大小限制以及浏览器在存cookie的个数也有限制，Session是没有大小限制和服务器的内存大小有关。
3. Cookie只能保存字符串数据,Session可以保存任意类型数据
4. Cookie有安全隐患，通过拦截或本地文件找得到你的cookie后可以进行攻击。
5. Session是保存在服务器端上会存在一段时间(默认30分钟)才会消失，如果session过多会增加服务器的压力。