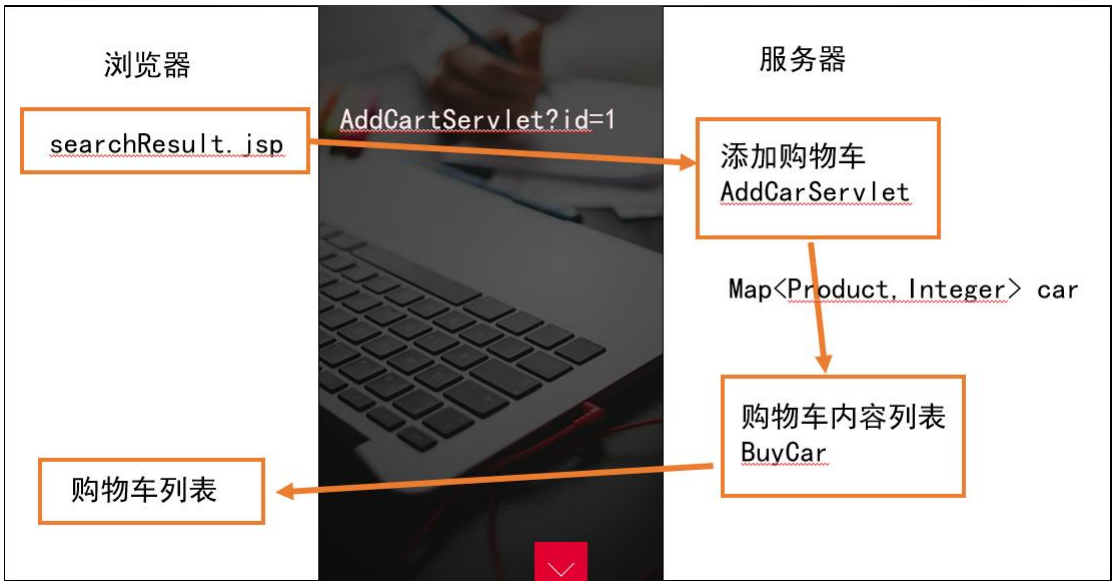
第三讲 JavaWeb 域对象

一、案例 1：添一件商品进购物车



在我们搜索结果的页面上，每件商品都有一个加入购物车的链接，点击这个链接，我们就向服务器端发送一个请求，处理这个请求的 Servlet，我们叫做 AddCarServlet
在这个 Servlet 中，查询一下商品的信息

修改 searchResult.jsp 页面，在加入购物车的链接上，添加 AddCarServlet，然后是参数 id
<a href="AddCarServlet?id=<%=product.getId() %>" class="j_car">加入购物车

map 集合就代表着，一个购物车
接下来我们在 AddCarServlet 中定义购物车 map，并将查询结果放入到 map 中
Map<Product,Integer> car = new HashMap<Product,Integer>();
car.put(product, 1);

二、案例 2：添多件商品进购物车

当请求到达 Servlet 后，获取商品信息，接下来的问题是无论这是放入到购物车的第几件商品，购物车都会重新创建，所以在实例化购物车之前，我们先要判断这个购物车有没有，如果有就在原有的购物车集合上添加新的商品，如果没有就实例化一个购物车

AddCartServlet

```
package com.shoppingstreet.servlet;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

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 javax.servlet.http.HttpSession;

import com.shoppingstreet.entity.Product;
import com.shoppingstreet.service.ProductService;
import com.shoppingstreet.service.impl.ProductServiceImpl;

/**
 * Servlet implementation class AddCartServlet
 */
@WebServlet("/AddCarServlet")
public class AddCarServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	/**
	 * @see HttpServlet#HttpServlet()
	 */
	public AddCarServlet() {
```

```
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String id = request.getParameter("id");
        ProductService productService = new ProductServiceImpl();
        Product product = productService.getProductById(Integer.valueOf(id));

        int num = 1;
        Map<Product,Integer> car = null;
        Object obj = request.getAttribute("car");
        if(obj==null){
            car = new HashMap<Product,Integer>();
        }
        else{
            car = (Map<Product,Integer>)obj;
            if(car.containsKey(product)){
                num = car.get(product)+1;
            }
        }

        car.put(product, num);
        request.setAttribute("car", car);

        request.getRequestDispatcher("BuyCar.jsp").forward(request, response);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
```

```
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

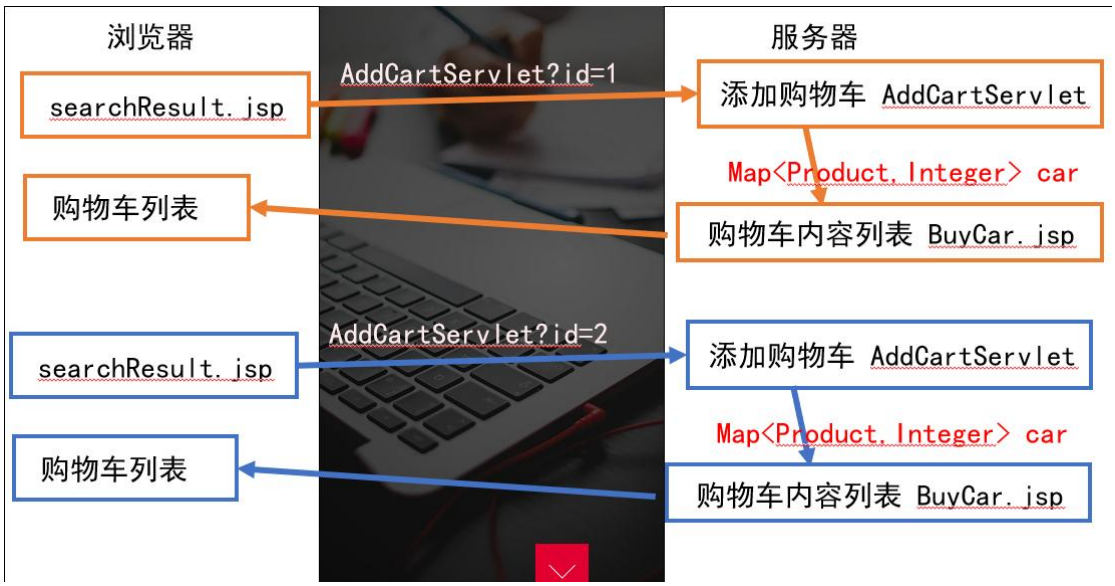
- **新的问题：**

我们第一次请求是把第一款的手机，放入到 request 的 attribute 属性中，转发到了 buyCar.jsp

当我们第二次请求传送 id 到服务器端的时候，注意这个时候服务器是新生成一个 request 对象，也就是说第一次请求和第二次请求的 request 对象不是同一个，request 的生命周期就是在一个请求和响应之间

因为 request 对象都不是同一个，所以上面放入到 reques 对象 Attribute 属性中的 map 在下面的 request 对象中自然是没有的

因此我们这里要解决的问题就是，如何在多次请求和响应之间传递值，也就是说在第一次请求中生成的 map
可以在第二次请求的时候访问到



解决思路 --- session

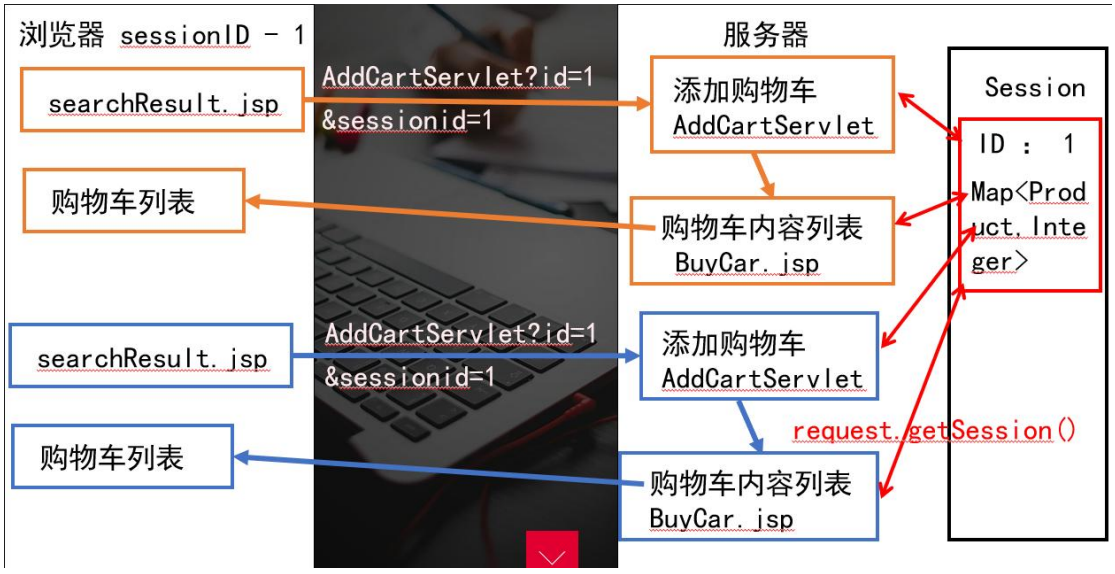
三、Session 会话

1. Session 定义

在服务器端内存开辟一个存储空间,把需要在多个请求和响应之间传递的值放入到这个空间中，以后无论是 servlet 还是 jsp 都可以在这个空间中取值

这样就解决了，数据无法在多次请求和响应之间传递的问题

而这个内存空间，给他取了个名字叫做 session
这种解决方案，机制，称之为会话

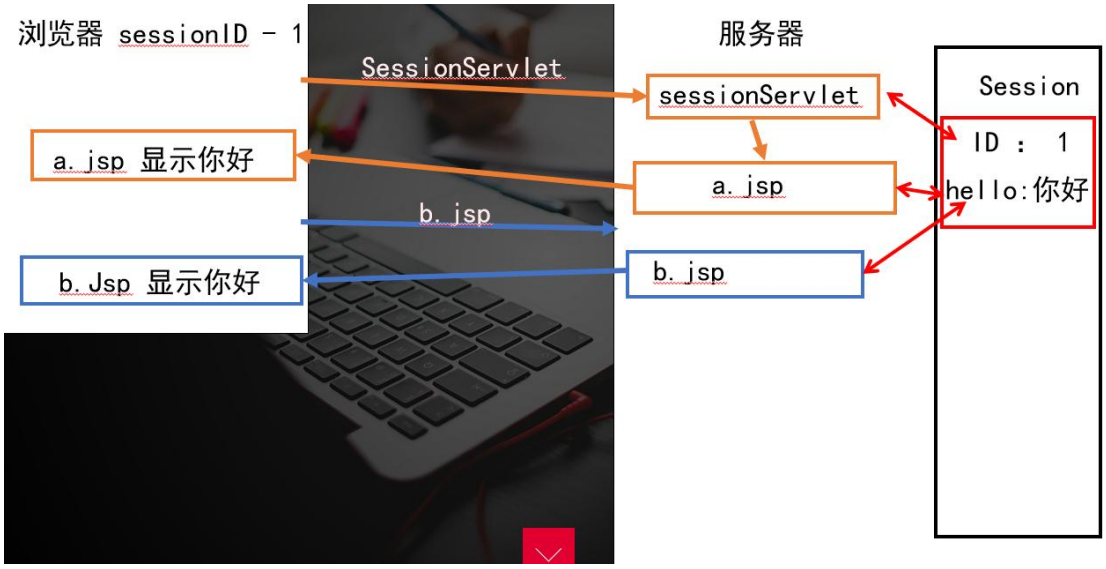


2. Session 相关方法

方法名称	说明
String getId()	获取 sessionid
void setMaxInactiveInterval(int interval)	设定 session 的非活动时间

int getMaxInactiveInterval()	获取 session 的有效非活动时间(以秒为单位)
void invalidate()	设置 session 对象失效
void setAttribute(String key, Object value)	以 key/value 的形式保存对象值
Object getAttribute(String key)	通过 key 获取对象值
void removeAttribute(String key)	从 session 中删除指定名称(key)所对应的对象

3. Session 案例



SessionServlet.java

```
import java.io.IOException;
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 javax.servlet.http.HttpSession;

/**
 * Servlet implementation class SessionServlet
```

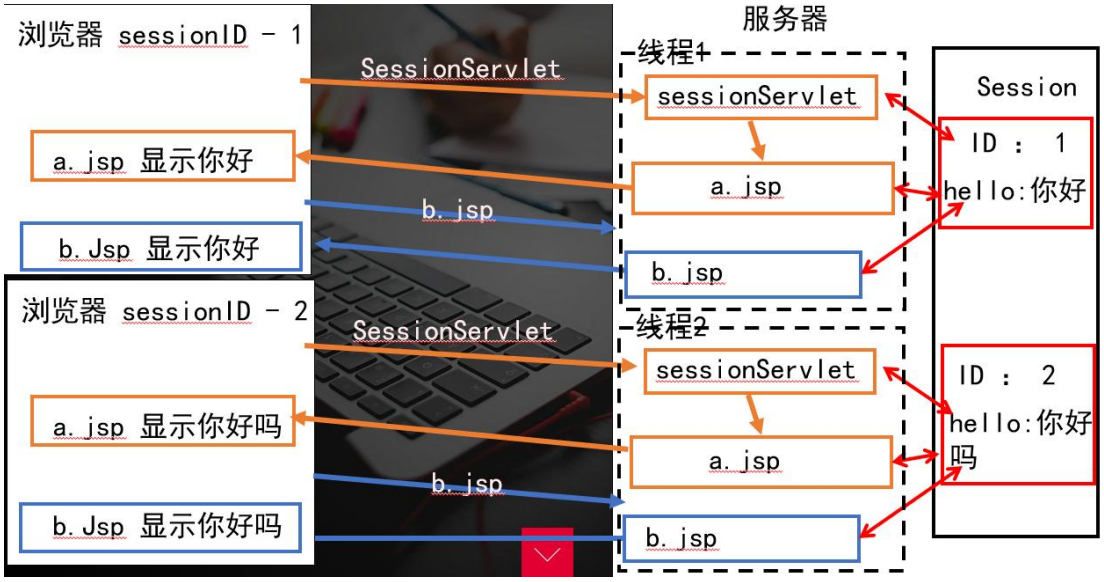
```
 */
@WebServlet("/SessionServlet")
public class SessionServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	/**
	 * @see HttpServlet#HttpServlet()
	 */
	public SessionServlet() {
		super();
		// TODO Auto-generated constructor stub
	}

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		HttpSession session = request.getSession();
		session.setAttribute("hello", "你好");
		session.setMaxInactiveInterval(10);
		request.getRequestDispatcher("a.jsp").forward(request, response);
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
		doGet(request, response);
	}

}
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>a.jsp</h1>
<%=session.getAttribute("hello") %>
<br/>
<%=session.getId() %>
</body>
</html>
```

4. session 与窗口的关系

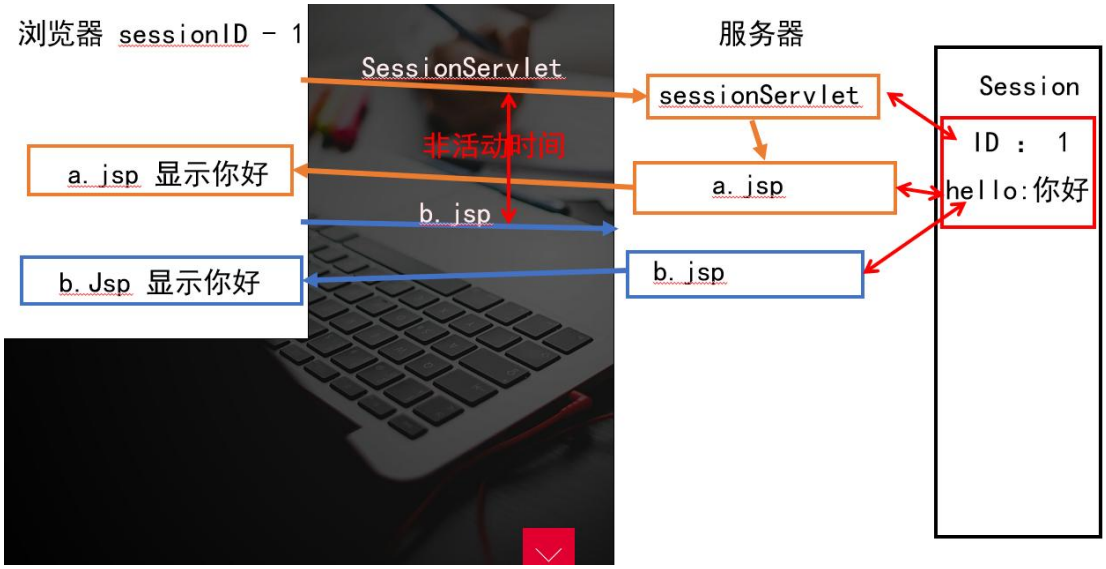
每个 session 对象都与一个浏览器窗口对应，重新开启一个浏览器窗口，可以重新创建一个 session 对象（不同版本浏览器可能有所差别）

当 session 对象创建的时候,系统会给它分配一个 sessionId,每次请求会将这个 sessioinId 发送给服务器端，所以服务器就靠着这个 id，来分辨



5. session 生命周期

因为 session 是把数据保存在服务器内存中，而服务器内存是有限的，所以 session 不会长时间的保存在服务器端



session 的生命周期的计算有些复杂

同一个浏览器两次请求之间会有一个时间间隔

我们如果给这个间隔取个名字：非活动时间

那么当非活动时间超过所设定的值之后，当前的这个 session 对象就会被服务器销毁

那么服务器默认的 session 非活动时间长度最长是多少呢

30 分钟

当然我们可以设置这个时间长度，比如用代码设置

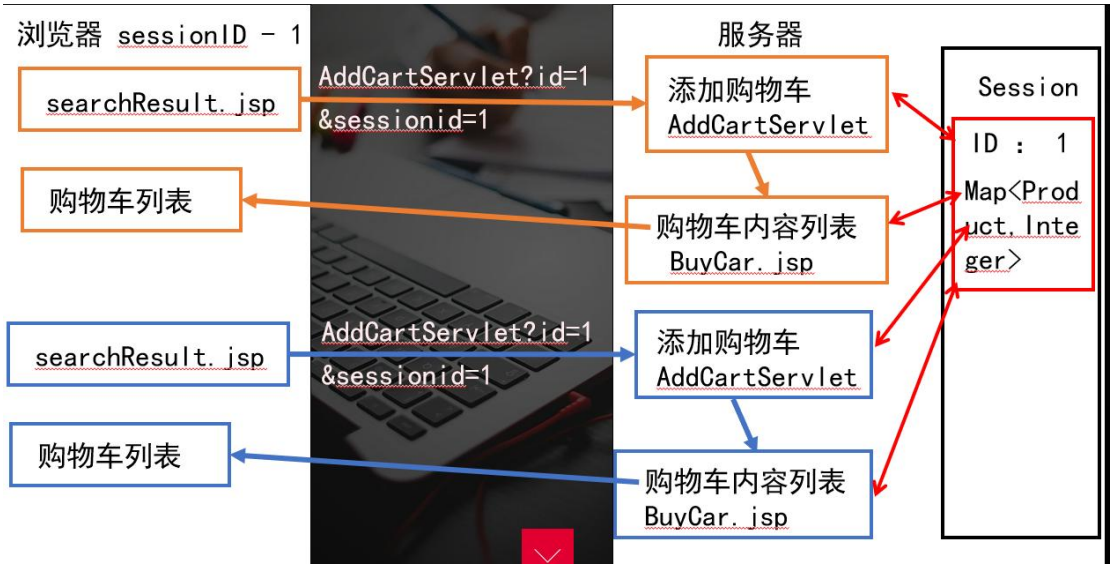
在 session 对象的方法列表中，就有一个
setMaxInactiveInterval 方法，单位是秒
测试一下
在 sessionServlet 中添加一行代码
session.setMaxInactiveInterval(10);运行这个 servlet
在同一个浏览器窗口，打开新的标签页直接访问 a.jsp 页面
如果速度够快，我们访问的还是同一个 session
如果超过 10 秒钟，我们看到的就是另一个 sessionId，从 session 取出的数据也是 null 值了

另外我们还可以在 web.xml 设置 session 的非活动时间长度

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

另外在一些特殊场合，要立即销毁 session 该怎么办
在 session 中有个 invalid 方法
可以立刻销毁当前的 session 对象

6. 购物车——Session



四、JSP 内置对象

1. 定义

JSP 内置对象是 Web 容器创建的一组对象

jsp 在本质上也是一个 servlet，所以在使用 session 的时候，没有任何特例，一定要先声明和初始化 session 才能够调用

那么 jsp 中，在哪初始化了这个 session 对象呢，要搞清楚这个问题

还是得进入 apache 的 work 目录，找到之前的 a.jsp 页面饭以后的 servlet，找到 jsp_service

方法

仔细看，这里面就有对 session 对象的初始化，所以我们在 jsp 中就可以直接调用 session，而不需要自己手动实例化了，像这样的对象，就称为 jsp 的内置对象

包括前面在 jsp 页面中，使用 request 对象等等

2. 九大内置对象

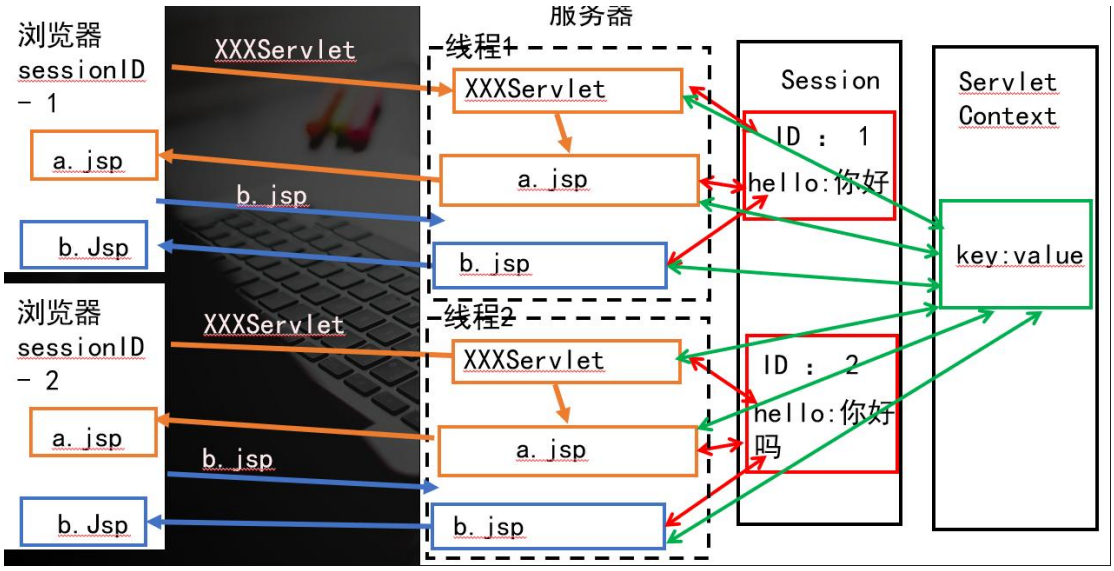
名称	类型	描述
out	javax.servlet.jsp.JspWriter	用于页面输出
request	javax.servlet.http.HttpServletRequest	得到用户请求
response	javax.servlet.http.HttpServletResponse	服务器向客户端的回应信息
config	javax.servlet.ServletConfig	服务器配置，可以获取初始化参数
session	javax.servlet.http.HttpSession	用于保存用户信息
application	javax.servlet.ServletContext	所有用户共享信息
page	java.lang.Object	指当前页转换后的 Servlet 的实例
pageContext	javax.servlet.jsp.PageContext	Jsp 的页面容器
exception	java.lang.Throwable	表示 jsp 页面所发生的异常，在错误页中才起作用

3. Application 与 ServletContext

session 代表的是在服务器端的一块内存空间，不同的浏览器窗口对应着的 session 对象是不一样的，

那么如果有这样一个值，它是需要在多个浏览器窗口共享调用的呢，虽然这种情况不多见，但确实有这个潜在的需求存在

这个时候就可以用到 application 这个内置对象了，它可以在多个窗口间共享访问



新建一个 Servlet，就叫做 appServlet 好了

在 AppServlet 中，放入一个值到 application 中

注意这是在 servlet 中，所以不能直接去调用 application

需要手动获取 servletContext 类型对象，不过对象名我们依然可以叫做 application

放一个值到 application 中去

ServletContext application = this.getServletContext();

application.setAttribute("appkey", "appvalue");

我们可以跳转到 a.jsp 页面然后去获取这个 application 中的值

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

接下来在页面 a 中，用内置对象 application 取出值

<%=application.getAttribute("appkey") %>

ok，值能够取到，接下来我们在 b 页面中也取出 application 中的值

也能取到值对吧，所以呢 application 能够跨多次请求和响应的，最后我们测试一下，跨窗口行不行

打开 ie 浏览器的窗口，直接访问 b 页面
没有问题，所以 application 能够跨多次请求，并且能够跨多个窗口共享数据

application 的生命周期又是啥样子的呢
其实当 tomcat 启动完毕后，application 就已经实例化成功了，一直到 tomcat 被关闭 application 对象才会被销毁掉
所以 application 对象的生命周期是贯穿整个 web 应用的

所以只要 tomcat 没有重启，无论是哪个浏览器，无论是什么时候发出的请求，它所访问的 application 对象都是同一个

4. 对象作用域

说到内置对象的作用域，其实一共有四个
不过 page 的作用域太小，就只是在一个 jsp 页面中，
没啥大的实际意义
大家了解一下就可以了

然后就是 request 作用域，对应的是一次请求，所以每次请求都是一个新的 request 对象

session 作用域：对应一个用户会话
一个用户指的就是一个窗口，或者一个 sessioinid
会话指的就是多次请求和响应
因此一个用户会话，指的就是同一个 sessionId，在多次请求响应之间
这是 session 的作用域

application 作用域，对应整个应用上下文
当 tomcat 启动完毕后，application 就已经实例化成功了，一直到 tomcat 被关闭 application 对象才会被销毁掉
所以 application 对象的生命周期是贯穿整个 web 应用的