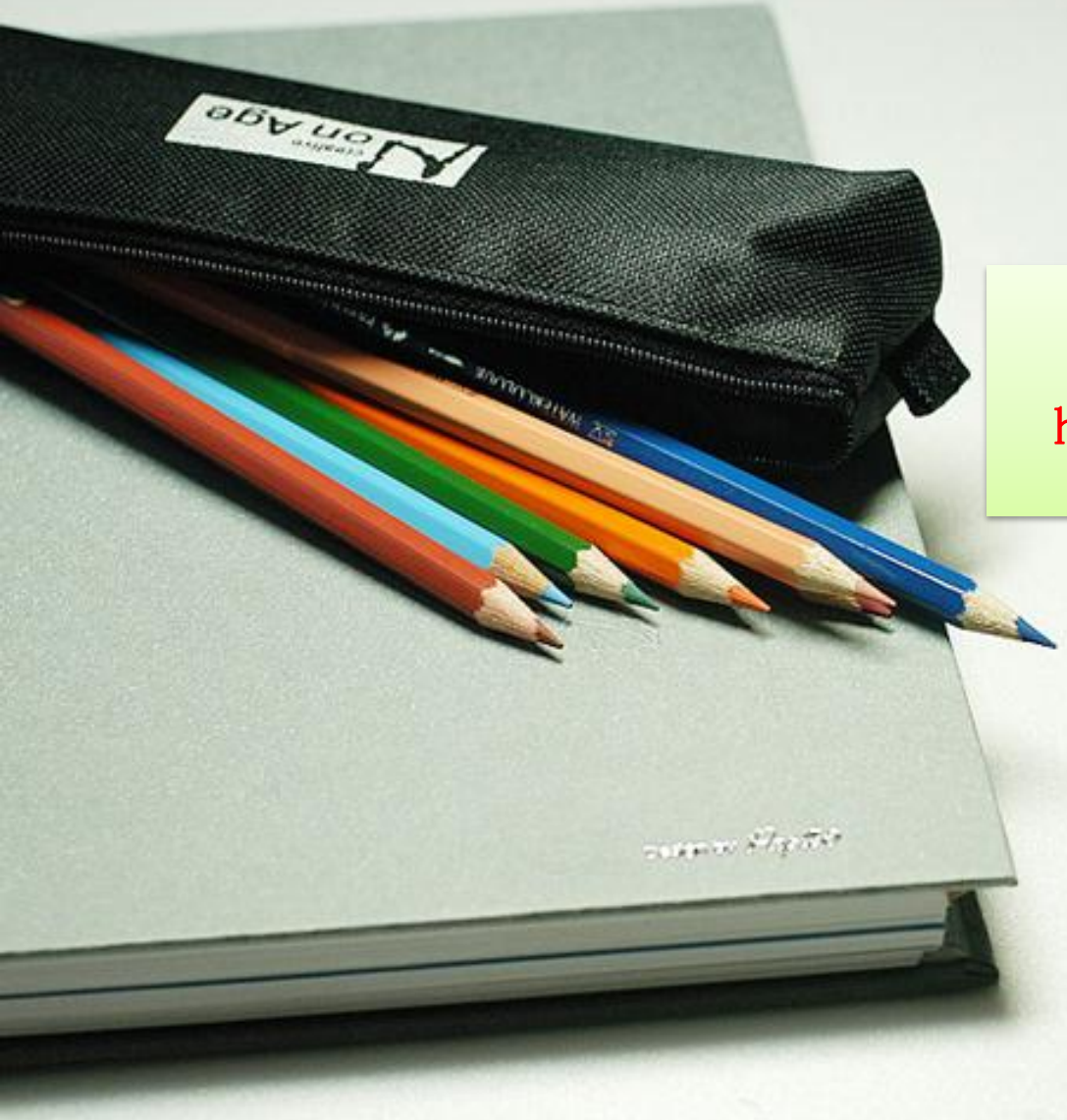


北京圣思园科技有限公司
http://www.shengsiyuan.com

主讲人：张龙

Servlet技术

- 教学目标
  - 了解Servlet的功能
  - 了解Servlet的生命周期
  - 了解Servlet的API
  - 掌握创建并发布HttpServlet的方法
  - 理解ServletContext与JavaWeb应用的关系

Servlet简介

- Java Servlet是和平台无关的服务器端组件，它运行在Servlet容器中。Servlet容器负责Servlet和客户的通信以及调用Servlet的方法，Servlet和客户的通信采用“请求/响应”的模式。
- Servlet可完成如下功能：
  - 创建并返回基于客户请求的动态HTML页面。
  - 创建可嵌入到现有 HTML 页面中的部分 HTML 页面（HTML 片段）。
  - 与其它服务器资源（如数据库或基于Java的应用程序）进行通信。

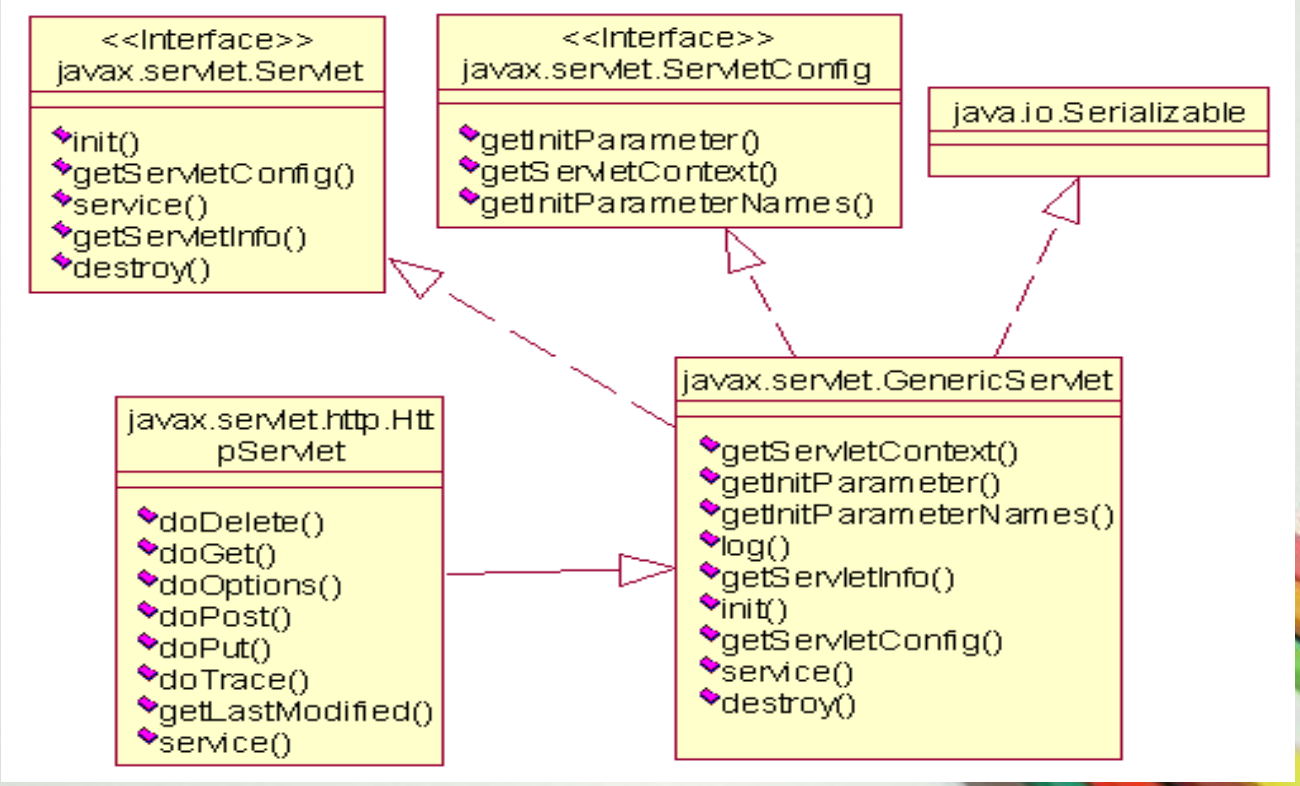
Servlet API

- Servlet的框架是由两个Java包组成：
  - javax.servlet包：定义了所有的Servlet类都必须实现或扩展的通用接口和类。
  - javax.servlet.http包：定义了采用HTTP协议通信的HttpServlet类。

Servlet API

- Servlet的框架的核心是javax.servlet.Servlet接口，所有的Servlet都必须实现这一接口。在Servlet接口中定义了五个方法，其中有三个方法代表了Servlet的生命周期:
  - init方法：负责初始化Servlet对象；
  - service方法：负责响应客户的请求；
  - destroy方法：当Servlet对象退出生命周期时，负责释放占用的资源。

Servlet API


<<Interface>>
javax.servlet.Servlet
init()
getServletConfig()
service()
getServletInfo()
destroy()
<<Interface>>
javax.servlet.ServletConfig
getInitParameter()
getServletContext()
getInitParameterNames()
java.io.Serializable
javax.servlet.GenericServlet
getServletContext()
getInitParameter()
getInitParameterNames()
log()
getServletInfo()
init()
getServletConfig()
service()
destroy()
javax.servlet.http.HttpServlet
doDelete()
doGet()
doOptions()
doPost()
doPut()
doTrace()
getLastModified()
service()

Servlet API

- 如果你的Servlet类扩展了HttpServlet类，你通常不必实现service方法，因为HttpServlet类已经实现了service方法，该方法的声明形式如下：

```
protected void service(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
                                      IOException;
```

- 在HttpServlet的service方法中，首先从HttpServletRequest对象中获取HTTP请求方式的信息，然后再根据请求方式调用相应的方法。例如：如果请求方式为GET，那么调用doGet方法；如果请求方式为POST，那么调用doPost方法。

ServletRequest接口

- ServletRequest接口中封装了客户请求信息，如客户请求方式、参数名和参数值、客户端正在使用的协议，以及发出客户请求的远程主机信息等。ServletRequest接口还为Servlet提供了直接以二进制方式读取客户请求数据流的ServletInputStream。
- ServletRequest的子类可以为Servlet提供更多的和特定协议相关的数据. 例如: HttpServletRequest 提供了读取HTTP Head信息的方法。

ServletRequest接口的主要方法

- getAttribute 根据参数给定的属性名返回属性值
- getContentType 返回客户请求数据MIME类型
- getInputStream 返回以二进制方式直接读取客户请求数据的输入流
- getParameter 根据给定的参数名返回参数值
- getRemoteAddr 返回远程客户主机的IP地址
- getRemoteHost 返回远程客户主机名
- getRemotePort 返回远程客户主机的端口

ServletResponse 接口

- ServletResponse 接口为Servlet提供了返回响应结果的方法。它允许Servlet设置返回数据的长度和MIME类型, 并且提供输出流ServletOutputStream。
- ServletResponse子类可以提供更多和特定协议相关的方法。例如: HttpServletResponse 提供设定HTTP HEAD信息的方法。

ServletResponse 接口的主要方法

- getOutputStream　返回可以向客户端发送二进制数据的输出流对象ServletOutputStream
- getWriter　返回可以向客户端发送字符数据的PrintWriter对象
- getCharacterEncoding　返回Servlet发送的响应数据的字符编码
- getContentType　返回Servlet发送的响应数据的MIME类型
- setContentType　设置Servlet发送的响应数据的MIME类型

Servlet的生命周期

- Servlet 的生命周期可以分为三个阶段:
  - 初始化阶段
  - 响应客户请求阶段
  - 终止阶段
- 在javax.servlet.Servlet接口中定义了三个方法init(), service(), destroy()，它们将分别在Servlet的不同阶段被调用。

Servlet的初始化阶段

- 在下列时刻Servlet容器装载Servlet：
  - Servlet容器启动时自动装载某些Servlet
  - 在Servlet容器启动后，客户首次向 Servlet 发出请求
  - Servlet的类文件被更新后，重新装载Servlet

- Servlet被装载后，Servlet容器创建一个 Servlet 实例并且调用 Servlet 的 init()方法进行初始化。在Servlet的整个生命周期中，init方法只会被调用一次。

Servlet的响应客户请求阶段

- 对于到达Servlet容器的客户请求，Servlet容器创建特定于这个请求的ServletRequest对象和ServletResponse对象，然后调用 Servlet 的service方法。service方法从ServletRequest对象获得客户请求信息、处理该请求，并通过ServletResponse对象向客户返回响应结果。

Servlet的终止阶段

- 当Web应用被终止，或Servlet容器终止运行，或Servlet容器重新装载Servlet的新实例时，Servlet容器会先调用 Servlet的destroy方法。在destroy方法中，可以释放Servlet所占用的资源。

创建用户自己的HttpServlet类的步骤

- （1） 扩展 HttpServlet 抽象类。
- （2） 覆盖HttpServlet的部分方法，如覆盖doGet()或doPost()方法。
- （3） 获取HTTP 请求信息，例如通过HttpServletRequest对象来检索 HTML 表单所提交的数据或 URL 上的查询字符串。无论是HTML表单数据还是URL 上的查询字符串，在HttpServletRequest 对象中都以参数名/参数值的形式存放，你可以通过getParameter(String name)方法检索参数信息。

创建用户自己的HttpServlet类的步骤（续）

- （4） 生成 HTTP 响应结果。通过 HttpServletResponse 对象可以生成响应结果。HttpServletResponse 对象有一个 getWriter()方法，该方法返回一个 PrintWriter 对象。使用 PrintWriter 的 print()或 println()方法可以向客户端发送字符串数据流。

创建HelloServlet类

- 第一步： 扩展 HttpServlet 抽象类。

```
public class HelloServlet extends HttpServlet
```

- 第二步：覆盖doGet()方法

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws IOException ,ServletException
```

创建HelloServlet类(复习)

- 第三步：获取HTTP 请求中的参数信息

```
String clientName=request.getParameter("clientName");
if(clientName!=null)
    clientName=
    new              String(clientName.getBytes("ISO-8859-
  1"),"GB2312");
else
    clientName="我的朋友";
```

创建HelloServlet类(复习)

- 第四步：生成 HTTP响应结果

```
PrintWriter out;
String title="HelloServlet";
String heading1="This is output from HelloServlet by
                doGet:";
// set content type.
response.setContentType("text/html;charset=GB2312");
```

创建HelloServlet类(复习)

- 第四步：生成 HTTP 响应结果（续）

```
// write html page.
out = response.getWriter();

out.print("<HTML><HEAD><TITLE>"+title+"</TITLE>"
);
out.print("</HEAD><BODY>");
out.print(heading1);
out.println("<h1><P> "+clientName+" : 您好</h1>");
out.print("</BODY></HTML>");
//close out.
out.close();
```

发布HelloServlet (复习)

- 在web.xml中为HelloServlet类加上如下<servlet>和<servlet-mapping>元素：

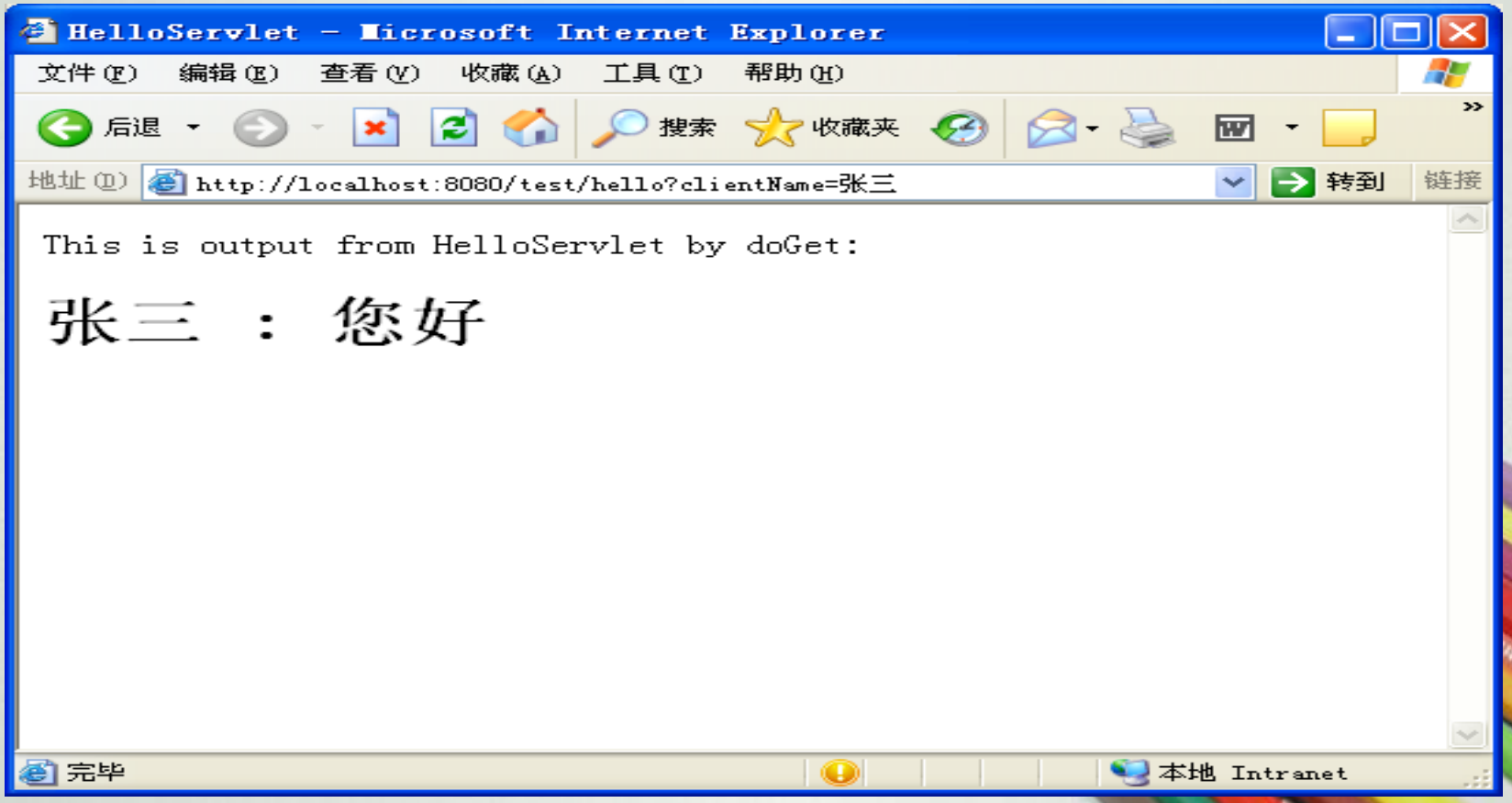
```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.test.servlet.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

运行HelloServlet（复习）

- 通过如下URL访问HelloServlet

  http://localhost:8080/test/hello?clientName=张三

运行HelloServlet（复习）



Servlet对象何时被创建

- 默认情况下，当Web客户第一次请求访问某个Servlet时，Web容器创建这个Servlet的实例
- 如果设置了<servlet>元素的<load-on-startup>子元素，Servlet容器在启动Web应用时，将按照指定的顺序创建并初始化这个Servlet。

```
<servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.test.servlet.HelloServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
</servlet>
```

Web应用何时被启动

- 当Servlet容器启动时，会启动所有的Web应用
- 通过控制台启动Web应用

ServletContext和Web应用关系

- 当Servlet容器启动Web应用时，并为每个Web应用创建惟一的ServletContext对象。你可以把ServletContext看成是一个Web应用的服务器端组件的共享内存。在ServletContext中可以存放共享数据，它提供了读取或设置共享数据的方法：
  - setAttribute（String name,Object object）把一个对象和一个属性名绑定，将这个对象存储在ServletContext中。
  - getAttribute（String name）根据给定的属性名返回所绑定的对象

创建CounterServlet

- **CounterServlet在ServletContext中存放了一个count属性：**

```
ServletContext context = getServletContext();
// 从ServletContext读取count属性
Integer count = (Integer)context.getAttribute("count");
if ( count == null ) {
     count = new Integer(0);
     context.setAttribute("count", new Integer(0));
}
```

创建CounterServlet

- 每次访问CounterServlet，它会把count属性的值增1，然后把它存储到ServletContext中：

```
// 创建新的count对象，其值增1
count = new Integer(count.intValue() + 1);
// 将新的count属性存储到ServletContext中
context.setAttribute("count", count);
```

测试CounterServlet

- （1）通过如下URL访问CounterServlet:
  http://localhost:8080/test/counter
- 当你第一次访问该Servlet，你会在浏览器上看到count值为0。
- （2）刷新该页面，你会看到每刷新一次count值增加1，假定最后一次刷新后count值为5。
- （3）再打开一个新的浏览器，访问CounterServlet。此时count值为6。
- （4）重新启动Tomcat服务器，然后再访问CounterServlet，你会看到count值又被初始化为0。

Servlet的多线程同步问题(重要)

- Servlet/JSP技术和ASP、PHP等相比，由于其多线程运行而具有很高的执行效率。
- 由于Servlet/JSP默认是以多线程模式执行的，所以，在编写代码时需要非常细致地考虑多线程的同步问题。
- 如果在编写Servlet/JSP程序时不注意到多线程同步的问题，这往往造成编写的程序在少量用户访问时没有任何问题，而在并发用户上升到一定值时，就会经常出现一些莫明其妙的问题，对于这类随机性的问题调试难度也很大。

会导致同步问题的HelloServlet2

```
public class HelloServlet2 extends HttpServlet  {
    String clientName=null;
    public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException
    ,ServletException
    {
    clientName=request.getParameter("clientName");
     if(clientName!=null)
     clientName=new String(clientName.getBytes("ISO-8859-
    1"),"GB2312");
     else
     clientName="我的朋友";
```

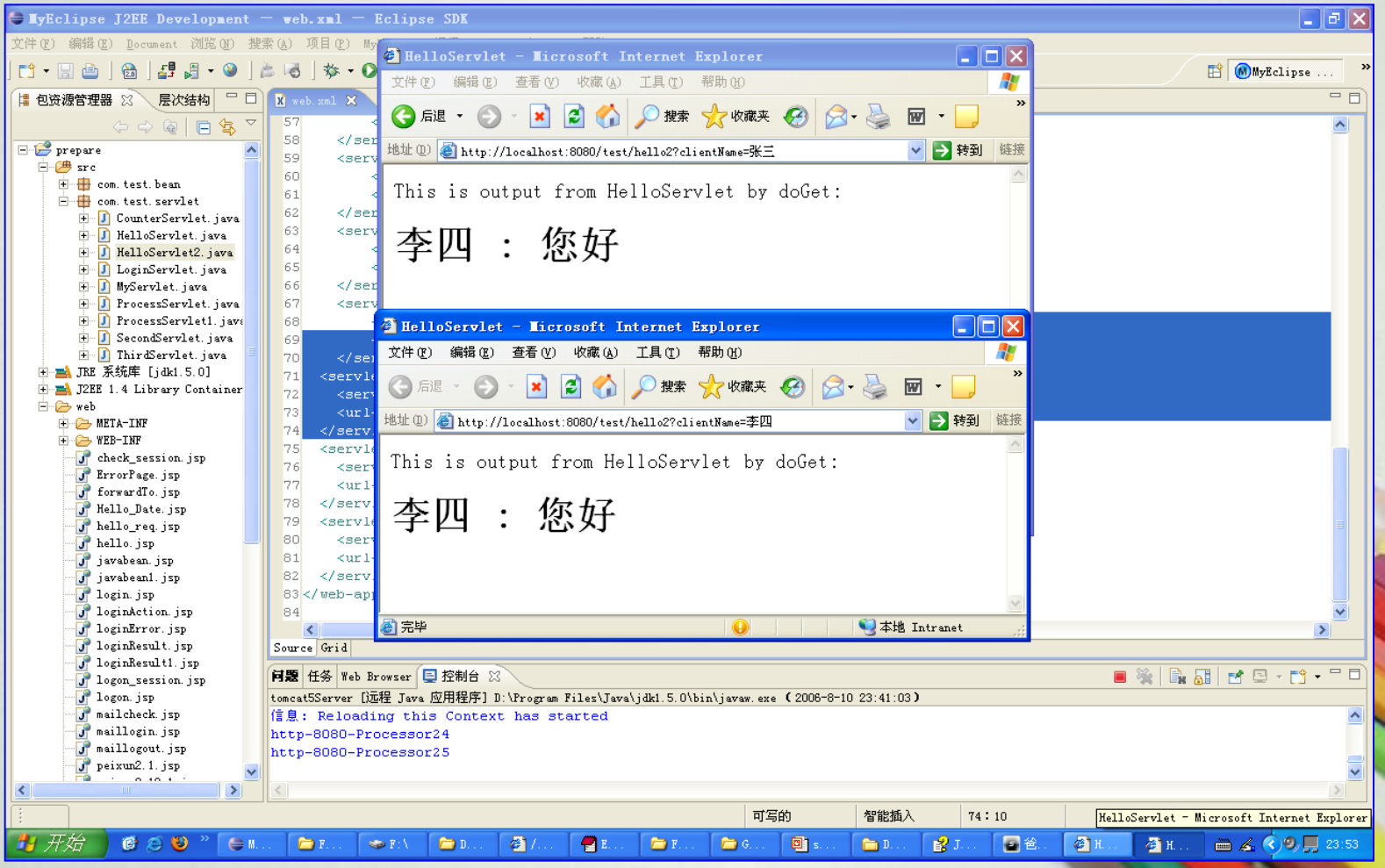
会导致同步问题的HelloServlet2

```
System.out.println(Thread.currentThread().getName());
    try{Thread.sleep(10000); }catch(Exception e){}

    // 第四步：生成 HTTP 响应结果。
     PrintWriter out;
     ……
     out.println("<h1><P> "+clientName+" : 您好</h1>");
     out.print("</BODY></HTML>");
     //close out.
     out.close();

   }
}
```

通过两个浏览器同时访问HelloServlet2



解决同步问题的方案

- Servlet实现javax.servlet.SingleThreadModel（Servlet2.4中已经废弃该接口），此时Servlet容器将保证Servlet实例以单线程方式运行，也就是说，同一时刻，只会有一个线程执行Servlet的service()方法。
- 去除实例变量，使用局部变量，参见HelloServlet
- 使用同步代码块：

  synchronized{…}

Cookie

- Cookie的英文原意是“点心”，它是用户访问Web服务器时，服务器在用户硬盘上存放的信息，好像是服务器送给客户的“点心”。
- 服务器可以根据Cookie来跟踪用户，这对于需要区别用户的场合（如电子商务）特别有用。

Cookie

- 一个Cookie包含一对Key/Value。下面的代码生成一个Cookie并将它写到用户的硬盘上：

```
Cookie  theCookie=new Cookie("cookieName","cookieValue");
response.addCookie(the Cookie);
```

Cookie

- 从用户硬盘上获取Cookie
- 参见程序：CookieServlet.java
- 参见程序：jspCookie.jsp

比较Servlet和JSP

- 有许多相似之处，都可以生成动态网页
- JSP的优点是擅长于网页制作，生成动态页面，比较直观。JSP的缺点是不容易跟踪与排错。
- Servlet是纯Java语言，擅长于处理流程和业务逻辑。Servlet的缺点是生成动态网页不直观。

练习题1

- 问题：**HttpServletRequest**对象是由谁创建的?
- 选项:
  - **(A)**由**Servlet**容器负责创建，对于每个**HTTP**请求， **Servlet**容器都会创建一个**HttpServletRequest**对象
  - **(B)**由**JavaWeb**应用的**Servlet**或**JSP**组件负责创建，当**Servlet**或**JSP**组件响应**HTTP**请求时，先创建**HttpServletRequest**对象

练习题2

- 问题：从**HTTP**请求中，获得请求参数，应该调用哪个方法**?**
- 选项**:**
  - **(A)**调用**HttpServletRequest**对象的**getAttribute()**方法
  - **(B)**调用**ServletContext**对象的**getAttribute()**方法
  - **(C)**调用**HttpServletRequest**对象的**getParameter()**方法

练习题3

- 问题：**ServletContext**对象是由谁创建的?
- 选项:
  - **(A)**由**Servlet**容器负责创建，对于每个**HTTP**请求， **Servlet**容器都会创建一个**ServletContext**对象
  - **(B)**由**JavaWeb**应用本身负责为自己创建一个**ServletContext**对象
  - **(C)**由**Servlet**容器负责创建，对于每个**JavaWeb**应用，在启动时，**Servlet**容器都会创建一个**ServletContext**对象

练习题4

- 运行**LifeServlet**类，它用于测试**Servlet**的生命周期。通过浏览器访问**:**

  http://localhost:8080/test/life

  分析它的输出结果。

练习题5

- 分析**ServletRequest**、**ServletResponse**、**Servlet**、**ServletContext**等对象的生命周期，何时被创建，何时被销毁。

练习题6

- 问题：jspForward1.jsp要把请求转发给jspForward2.jsp，应该在jspForward1.jsp中如何实现?
- 选项：
  - (A) jspForward2.jsp
  - (B) <jsp:forward page="jspForward2.jsp">

练习题7

- 问题：jspForward1.jsp要把请求转发给jspForward2.jsp，在转发的时候，希望把用户名 “小新”传给jspForward2.jsp，如何实现?
- 选项：
  - (A) request.setParameter(“小新”);
  - (B) request.setAttribute(“username”, “小新”);
  - (C) jspForward2.jsp

练习题8

- 问题：当浏览器第二次访问该JSP网页时的输出结果是什么？

```
<!% int a=0;     %>
<%
    int b=0;
    a++;
    b++;
%>

 a:<%= a %> <br>
 b:<%= b %>
```

练习题9

- 问题:request.getAttribute()和request.getParameter()方法有什么异同?
- 选项:
  - (A)前者返回Object类型的对象，后者返回String类型的对象
  - (B)request.getAttribute()和request.setAttribute()对应
  - (c) request.getParameter()和request.setParameter()对应。
  - (D)当两个Web组件之间为链接关系时，被链接的组件通过getParameter()方法来获得请求参数
  - (E)当两个Web组件之间为转发关系时，转发目标组件通过getAttribute()方法来和转发源组件共享request范围内的数据。
  - (F)request.getParameter()方法传递的数据，会从Web客户端传到Web服务器端，代表HTTP请求数据。
  - (G)request.setAttribute()和getAttribute()方法传递的数据只会存在于Web容器内部，在具有转发关系的Web组件之间共享。