

Java企业级应用开发

第2章：Servlet核心技术

- 第1节 **Servlet** 基础
- 第2节 **Servlet** 进阶
- 第3节 **Servlet** 处理请求
- 第4节 **Servlet** 处理响应

- 理解**Servlet**的作用和适用场合。
- 理解**Servlet**在**J2EE**体系中的地位。
- 理解**Servlet**的开发与配置的步骤。
- 理解**Servlet**的生命周期和回调函数的作用
- 理解**Servlet Request**、**Response**的构建和组成
- 会使用**Request**实现获取请求头、请求体。
- 会使用**Response**实现构建响应体正文、设置响应头参数。
- 能利用响应消息流实现文件下载。
- 懂得验证码的作用并能熟练构建。

■ 知识点预览

#	知识点	重点	难点	应用	说明
1	J2EE架构体系	√			讲解Java EE体系结构的组成。
2	Servlet概述	√			讲解Servlet组件使用的意义与适用场合。
3	开发Servlet组件	√		√	讲解如何利用Eclipse开发Servlet。
4	配置Servlet组件	√		√	讲解如何配置Servlet以满足各种请求。
5	接收客户端请求	√		√	讲解如何接收各种不同的客户端请求。
6	处理客户端请求	√		√	讲解如何获取客户端上传的数据。
7	响应客户端请求	√		√	讲解如何将处理完的请求回发到客户端。
8	设置索引页	√		√	讲解通过索引页访问Servlet的方法。

■ 什么是Java EE?

- ◆ Java平台共分为三个主要版本**Java EE**、Java SE和Java ME
- ◆ Java EE是Java平台企业版（Java Platform, Enterprise Edition）。
- ◆ Java EE是Sun公司为企业级应用推出的标准平台。

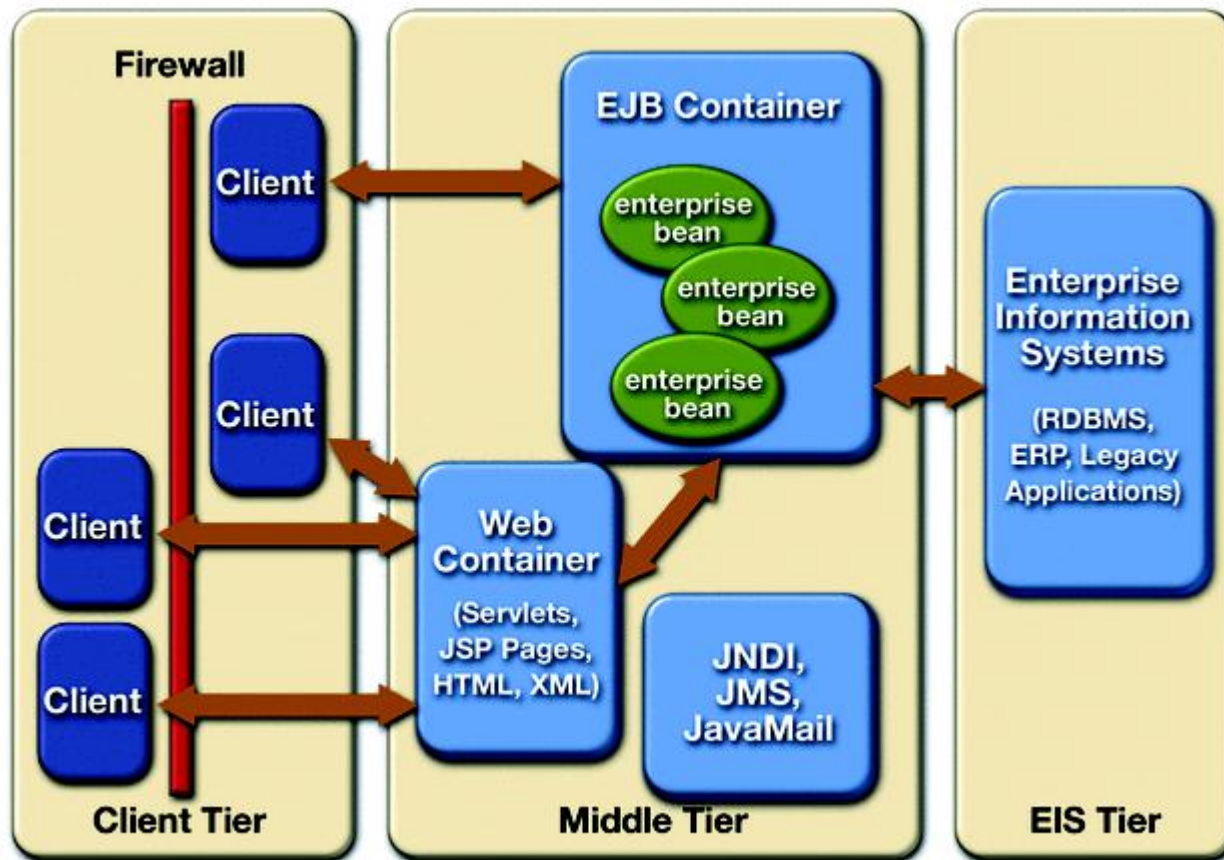
■ Java EE能做什么?

- ◆ Java EE核心是一组技术规范与指南，其中包含了各类组件、服务架构及技术层次。
- ◆ Java EE不仅仅是指一种标准平台，它更多的表达着一种软件架构和设计思想。
- ◆ 传统意义上的B/S架构系统（如：Web应用）就可以基于Java EE所提供的丰富的组件与服务实施开发。

Java EE包含了哪些技术与服务？

缩写	技术与服务名	中文名称
EJB	Enterprise Java Beans	企业JavaBean
JAAS	Java Authentication and Authorization Service	Java认证和授权服务
JACC	Java EE Authorization Contract for Containers	基于容器的授权服务
JAF	Java Beans Activation Framework	数据分装与数据识别套件
JAX-RPC	Java API for XML-Based Remote Procedure Calls	Java应用访问远程服务套件
JAX-WS	Java API for XML Web Services	Java应用访问Web服务套件
JAXM	Java API for XML Messaging	Java应用对于SOAP的支持API
JCA	J2EE Connector Architecture	跨容器平台通用连接标准
JDBC	Java Database Connectivity	Java应用访问数据的标准API
JPA	Java Persistence API	Java持久化技术标准API
JMS	Java Message Service	分布式系统通讯中间件
JMX	Java Management Extensions	系统集成与扩展插件标准
JNDI	Java Naming and Directory Interface	Java命名与目录结构
JSF	Java Server Faces	基于事件驱动模型的Web框架
JSP	Java Server Pages	Java动态服务页
JSTL	Java Server Pages Standard Tag Library	Java Web标准标记库
JTA	Java Transaction API	Java 事物管理API
JavaMail	Java Mail API	Java 邮件服务API
Servlet	Java Servlet API	Java 服务端应用API
StAX	Streaming APIs for XML Parsers	Java流模式解析XML标准API
WS	Web Services	Java跨平台数据交互技术

Java EE常用技术如何交互？



■ 为什么要使用Servlet组件？

◆ Java Web应用程序核心在于：

- 获取客户端用户的请求。
- 对客户端请求实施处理。
- 将处理完毕的结果响应给客户端。

◆ Java Web应用程序的业务运用场景，例如：

- 将用户需要购买的商品保存入“购物车”。
- 成为某网站的注册用户。
- 在论坛中发帖、回帖、删帖。
- 下单并查看订单状态。

◆ Servlet可以获取请求并处理各种复杂的客户端请求，同时可以将处理完毕的结果响应回客户端。

◆ Servlet是Java EE规范中一个重要的技术，由原SUN公司提出标准，目前最高版本3.0。

■ 什么是Servlet组件？

- ◆ Servlet是一个**普通Java Class**。
- ◆ Java Class要能够被称之为Servlet，必须继承

javax.servlet.http.HttpServlet

- ◆ Servlet是Java Web开发过程中**最重要的组件**，是**唯一**一个可以用来**接收客户端请求，并对请求作出处理的组件**。
- ◆ Servlet可以获取用户提交请求的请求体数据与请求头数据。
- ◆ Servlet可以分别接收GET请求和POST请求。
- ◆ Servlet可以获取用户的上传数据请求。
- ◆ Servlet可以计算与处理请求中所涉及的业务逻辑。
- ◆ Servlet可以将处理完的结果响应到客户端。
- ◆ Servlet提供各种不同格式的响应数据（HTML、图片等）。

■ 创建Servlet的步骤

- ◆ 步骤一、新建Java Class文件，例如：**FirstAction**，继承 `javax.servlet.http.HttpServlet`

```
public class FirstAction extends HttpServlet{  
  
}
```

- ◆ 步骤二、重写父类的doGet与doPost方法，这样当不同的请求发送到服务端，doGet和doPost的方法就会根据请求类型的不同被执行。

```
/** 接收客户端的GET类型请求*/
```

```
@Override
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException { }
```

```
/** 接收客户端的POST类型请求*/
```

```
@Override
```

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException { }
```

■ 配置Servlet

◆ Servlet除了需要Java代码实现外，**还需要通过配置来实现。**

➤ Servlet通过配置来应对不同的客户端请求。

- 一般客户端会发送名目繁多的请求，这些请求往往需要一个或者多个Servlet来应对，并根据请求作出不同的处理。
- Servlet配置就是为了告诉Java Web容器软件（Tomcat），什么类型的请求应该交给哪些Servlet处理。
- 所有Servlet的配置都需要编写在web.xml

➤ Servlet通过编写业务代码处理请求中涉及的业务。

- Servlet接收到请求后，根据请求与Java Web提供的服务进行业务计算和业务处理，例如：
 - ✓ 查询数据库中数据。
 - ✓ 保存用户的交互数据。
- Servlet业务代码同样需要将处理完的结果响应回客户端。

■ 配置Servlet的步骤

◆ 步骤一、在Web.xml文件配置所有研发完毕的Servlet组件。

```
<!--  
    配置Servlet组件，告诉Tomcat Web容器，  
    Servlet组件的名字和位置[全名（类名+包名） ]  
-->  
<servlet>  
    1  
    <!-- 由于Servlet名字太长，所以为该Servlet起一个别名 -->  
    <servlet-name>first</servlet-name>  
    <!-- Servlet的名字和位置[全名（类名+包名） ] -->  
    <servlet-class>com.servlet.FirstAction</servlet-class>  
</servlet>
```

■ 配置Servlet的步骤

◆ 步骤二、配置哪些请求可以访问到指定的Servlet。

<!-- 配置访问Servlet组件的URL，
超级链接只有满足以下配置才能访问到指定的Servlet

-->

<servlet-mapping>

1

<servlet-name>first</servlet-name>

<!-- 核心配置属性：超级连接尾部为/la时候，
login别名对应的FirstAction类能获得本次客户端请求
例如:http://127.0.0.1/WebApp/f
(尾部是f所有请求被FirstAction Servlet获取)

-->

<url-pattern>/f</url-pattern>

</servlet-mapping>

■ 访问Servlet的方法

◆ 客户端通过超级链接访问Servlet

- 客户端通过超级链接访问Servlet必须保证URL的尾部与Servlet配置的 **url-pattern** 部分保持一致。
- 例如：访问FirstAction, URL应为: <http://127.0.0.1/WebApp/f>。
- 通过超级链接访问的请求类型始终为: GET

```

package com.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstAction extends HttpServlet{

```

```

    /** 接收客户端的GET类型请求*/

```

```

    @Override

```

```

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)

```

```

    throws ServletException, IOException {

```

```

        doPost(req,resp);

```

```

    }

```

```

    /** 接收客户端的POST类型请求*/

```

```

    @Override

```

```

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)

```

```

    throws ServletException, IOException {

```

```

        response.setContentType("text/html");

```

```

        PrintWriter out = response.getWriter();

```

```

        out.println("<!DOCTYPE HTML PUBLIC");

```

```

        out.println("<HTML>");

```

```

        out.println("    <HEAD><TITLE>A Servlet");

```

```

        out.println("    <BODY>");

```

```

        out.print("        Hello World !");

```

```

        out.println("    </BODY>");

```

```

        out.println("</HTML>");

```

```

        out.flush();

```

```

        out.close();

```

```

    }

```

```

}

```

```

<servlet>

```

```

    <!-- 由于Servlet名字太长，所以为该Servlet起一个别名 -->

```

```

    <servlet-name>first</servlet-name>

```

```

    <!-- Servlet的名字和位置[全名(类名+包名)] -->

```

```

    <servlet-class>com.servlet.FirstAction</servlet-class>

```

```

</servlet>

```

```

<servlet-mapping>

```

```

    <servlet-name>

```

```

    <url-pattern>

```

```

</servlet-mapping>

```

```

</servlet-mappings>

```

```

Hello World !

```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```


```

```

localhost:8080/testspringmvc/f

```

■ 从请求中获取数据

- ◆ 请求被发送到Servlet中，可以在doGet或doPost中获取请求数据。
- ◆ doGet和doPost方法的HttpServletRequest中保存着请求数据。
 - 包含请求头的所有参数数据
 - 包含请求体中用户提交的数据（POST请求）
- ◆ 获取客户端用户输入的数据：

@Override

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {
```

```
// 获取客户端发送的请求类型：GET/POST
```

```
String method = req.getMethod();
```

```
}
```


■ 通过绘制网页响应客户端

- ◆ 业务处理完毕后，Servlet需要考虑响应机制。
 - Servlet需要考虑响应回客户端的数据类型。
 - Servlet需要考虑哪些数据需要被回发会客户端。
- ◆ 利用HttpServletResponse设置响应数据与数据类型。
 - 设置响应数据类型（设置在响应头中）。

@Override

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {
```

```
//设置响应头参数：本次响应的数据格式为text/html  
resp.setContentType("text/html");
```

```
}
```

■ 通过绘制网页响应客户端

◆ 利用HttpServletResponse设置响应数据与数据类型。

➤ 设置响应客户端的数据

@Override

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
    // 将获取的客户端请求类型发回客户端提供给用户（响应内容）
```

```
    String responseBody = "Current request method is <B>:"
                                                                    + method + "</B>";
```

```
    //将所有HTML响应数据通过网络回发到客户端。
```

```
    //获取发送响应的PrintWriter对象
```

```
    PrintWriter out = resp.getWriter();
```

```
    //装在需要响应到客户端的响应数据
```

```
    out.println(responseBody);
```

```
    //发送响应
```

```
    out.close();
```

```
}
```

localhost:8080/testspringmvc/formtest.html

用户名: 啊SD

密 码:

性 别: ☒ 男 ☐ 女

学 历: 小学 ▼

爱 好: ☒ 上网 ☒ 游戏 ☐ 聊天 ☐ 旅游 ☐ 逛街

简 介: 这个家伙很懒...

提交

重置



localhost:8080/testspringmvc/servlet/NormFormPostServlet

用户啊SD

密码:12

性别:1

学历:11

爱好:21

爱好:22

💎?💎?💎:这个家伙很懒...

host0:0:0:0:0:0:1 ss is null

■ 良好的编程习惯：为Web应用设置索引页

- ◆ Web应用程序都需要通过设置首页来简化URL的访问。
- ◆ Web应用程序一般不会将Servlet设置为首页。
- ◆ index.html是Web应用设置首页的首选（通过web.xml设置）。

```
<welcome-file-list>  
  <welcome-file>index.html</welcome-file>  
</welcome-file-list>
```

- ◆ 通过在index.html设置JS访问Servlet

```
<body>  
  <script type="text/javascript">  
    window.location.href = "f"; //f是需要访问Servlet的URL-Pattern  
  </script>  
</body>
```

- Servlet的运行过程与生命周期由Servlet容器控制。
- 一个来自于浏览器的Servlet请求一般按如下顺序被响应：
 - ◆ 1) Servlet容器检测是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行4)，否则，执行2)。
 - ◆ 2) 装载并创建该Servlet的实例对象。
 - ◆ 3) 调用Servlet实例对象的init()方法。
 - ◆ 4) 创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表响应消息的HttpServletResponse对象，然后调用Servlet的service()方法，并将上述请求和响应对象作为参数传递进去。

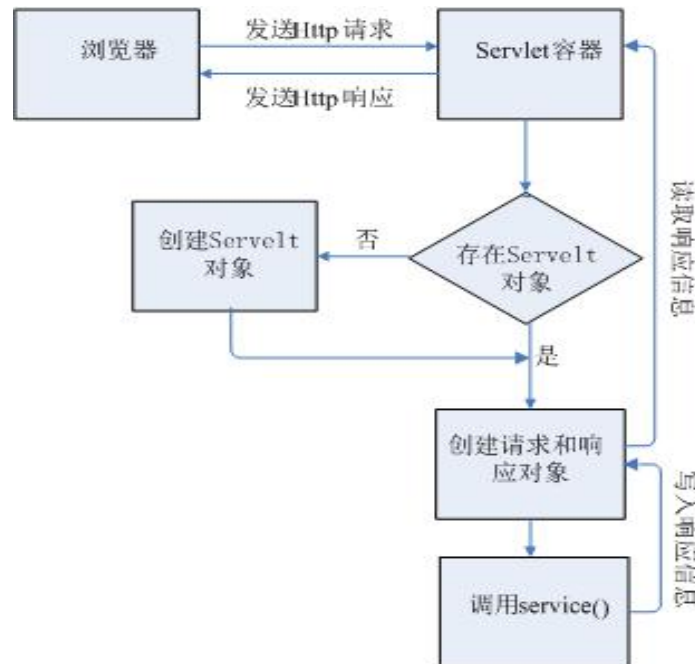


WORKFORCE
DEVELOPMENT PROGRAM



CAMPUS SOLUTION GROUP

- ◆ Web应用程序被停止或重新启动之前，Servlet容器将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。



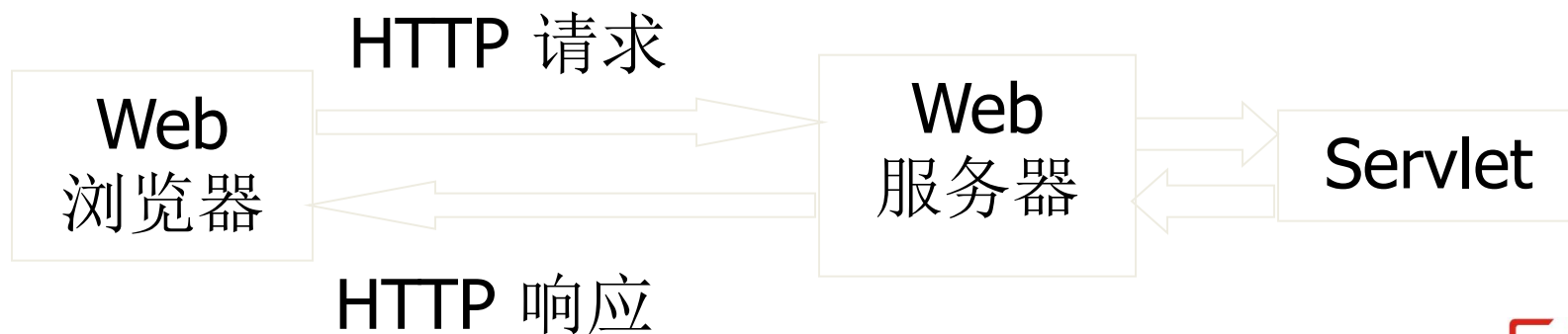
■ Servlet的功用

- 读取客户程序发送来的显式数据（表单数据）
- 读取客户程序发送来的隐式数据（请求报头）
- 生成相应的结果
- 发送显式的数据给客户程序（HTML）
- 发送隐式的数据给客户程序（状态代码和响应报头）

- Servlet (**Server Applet**)，全称Java Servlet。是用Java编写的服务器端程序。**其主要功能在于交互式地浏览和修改数据，生成动态Web内容**。狭义的Servlet是指Java语言实现的一个接口，广义的Servlet是指任何实现了这个Servlet接口的类，一般情况下，人们将Servlet理解为后者。

- 目前Servlet引擎一般是第三方的插件，它通过一定的方法连接到Web服务器，Servlet引擎把它识别为Servlet请求的那些HTTP请求截获下来处理，而其他的HTTP请求由Web服务器按照通常的方式来处理，Servlet引擎会装载合适的Servlet到内存中，如果Servlet还没有运行的话，会分配一个可以使用的线程来处理请求，再把Servlet的输出返回到发出请求的Web客户机。
- Servlet在服务器内部运行，通过客户端提交的请求启动运行

- 用Java 编写的服务器端程序
- 运行在 Web 服务器上
- 执行服务器端处理
- HttpServlet是Servlet的一种，是专门用来处理WEB应用的服务端组件，他在原来的基础上添加了专门处理HTTP协议几种方法(doGet、doPost、doHead等等)；并且重写service这个Servlet中核心方法，根据Http头部请求信息分别调用各种处理方法(doGet、doPost等)





- Servlet的请求首先会被HTTP服务器接收，HTTP服务器只负责静态HTML页面的解析，对于Servlet的请求转交给Servlet容器，Servlet容器会根据web.xml文件中的映射关系，调用相应的Servlet，Servlet将处理的结果返回给Servlet容器，并通过HTTP服务器将响应传输给客户端。
- Servlet技术具有如下特点：
 - 方便：Servlet提供了大量的实用工具例程，如处理很难完成的HTML表单数据、读取和设置HTTP头，以及处理Cookie和跟踪会话等。
 - 跨平台：Servlet用Java类编写，可以在不同操作系统平台和不同应用服务器平台下运行。
 - 灵活性和可扩展性：采用Servlet开发的Web应用程序，由于Java类的继承性及构造函数等特点，使得应用灵活，可随意扩展。
- 除了上述几点外，Servlet还具有功能强大、能够在各个程序之间共享数据、安全性强等特点，此处就不再详细说明，读者作为了解即可。

- 在J2EE中跟Servlet相关的一个包是`javax.servlet`，其中最基本的Servlet被声明为一个接口`javax.servlet.Servlet`，这是Servlet最高层次的一个抽象，它是和网络协议无关的。同样在`javax.servlet`中，实现了一个类`javax.servlet.GenericServlet`，这个类实现了Servlet接口，也是和协议无关的。而这个类是构建其他和协议相关的Servlet子类型的通用的父类(如`HttpServlet`)。
- Servlet所适用的网络协议可以是多种多样的，比如HTTP，FTP，SMTP，TELNET等，但是就目前而言，只有HTTP服务已经形成了标准的Java组件。对应的软件包有两个`javax.servlet.http`和`javax.servlet.jsp`，分别对应我们要讲解的Servlet和JSP编程。我们通常所说的Servlet编程主要就是指针对HTTP的Servlet编程，用到的就是`javax.servlet.http`包中的类（典型的的就是`HttpServlet`类），实际上Java Servlet编程的概念要更广一些，在这里我们也就约定俗成的使用Servlet来指代HTTP Servlet的编程。

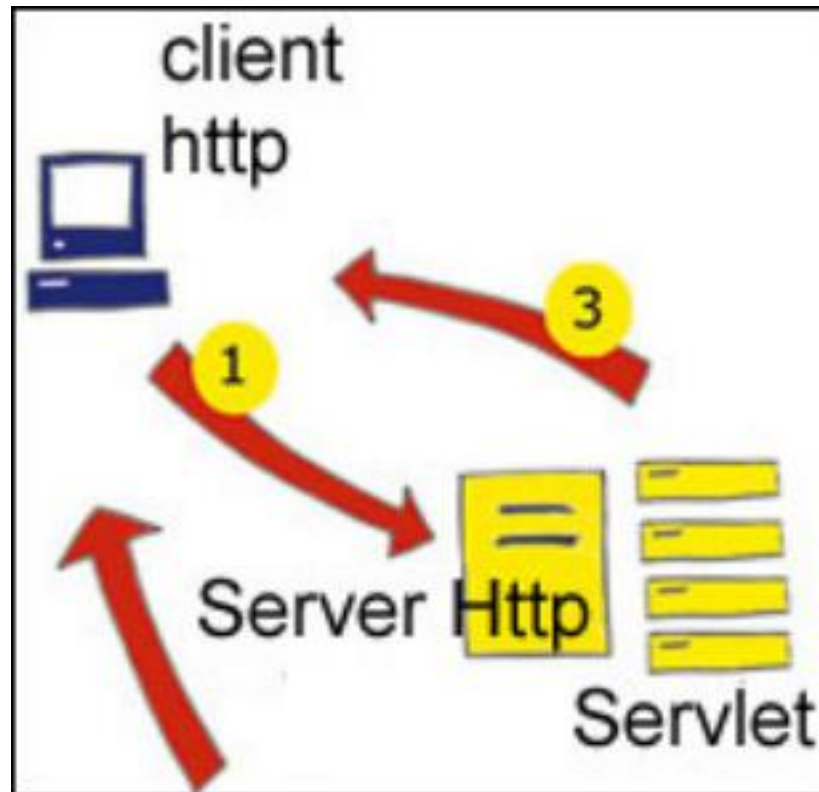
■ Servlet包含两个包:

◆ javax.servlet

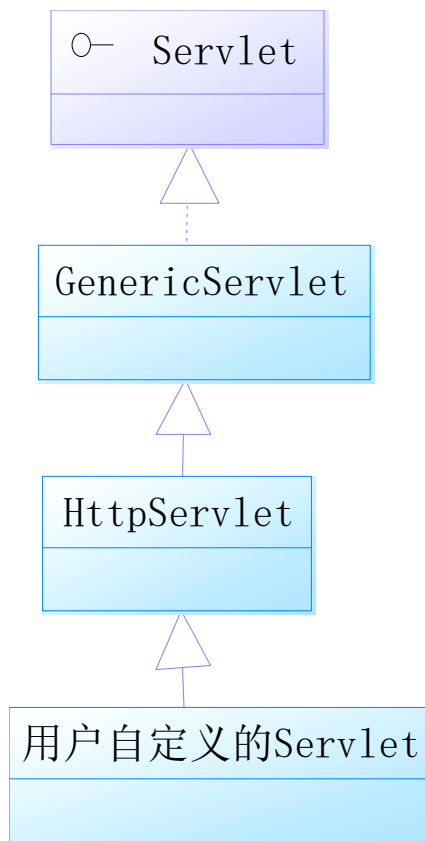
➤ 称为GenericServlet

➤ javax.servlet.http

● 称为HttpServlet



■ Servlet包的层次结构





■ 总结Servlet的创建过程？

◆ 请求URL路径与Servlet的匹配由：

- Web.xml配置。
- Web容器（Tomcat）负责通过<url-pattern>匹配。

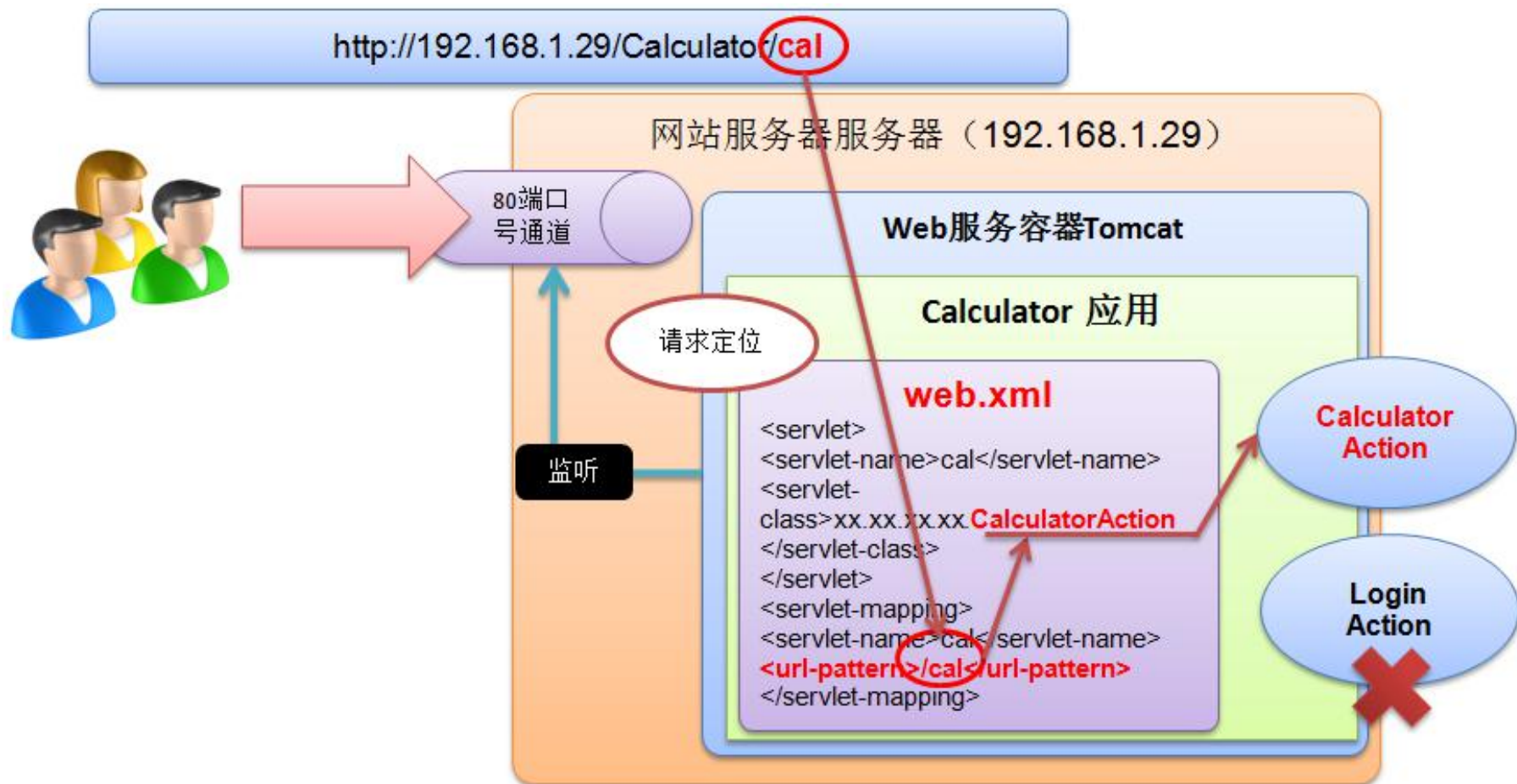
◆ Servlet的创建

- Web容器负责确定是否需要创建（如果已创建则不重复操作）。
- Web容器负责负责最终的Servlet创建。
- Servlet的创建永远都只会被执行一次。

◆ Servlet容器

- Web容器创建完Servlet，Servlet实例保存在Servlet容器中。
- 每个Web应用（站点）都有独立的Servlet容器。

谁负责将URL定位到Servlet?



■ URL匹配Servlet的规则:

- ◆ URL的**路径部分**一定要和Servlet的<url-pattern>节点的配置保持一致，否则将获取404的错误信息。

URL: http://127.0.0.1/WebApp/**cal**

```
<servlet-mapping>  
  <servlet-name>cal</servlet-name>  
  <url-pattern>/cal</url-pattern>  
</servlet-mapping>
```

URL: http://127.0.0.1/WebApp/**/cus/mgr/cal**

```
<servlet-mapping>  
  <servlet-name>cal</servlet-name>  
  <url-pattern>/cus/mgr/cal</url-pattern>  
</servlet-mapping>
```

■ URL匹配Servlet的规则:

◆ <url-pattern>节点定义规则:

➤ 支持通配符

URL: http://127.0.0.1/Calculator/abc.action

```
<servlet-mapping>  
  <servlet-name>c</servlet-name>  
  <url-pattern>*.action</url-pattern>  
</servlet-mapping>
```

URL: http://127.0.0.1/Calculator/abc.do

```
<servlet-mapping>  
  <servlet-name>d</servlet-name>  
  <url-pattern>*.do</url-pattern>  
</servlet-mapping>
```

■ 表单匹配Servlet的规则:

- ◆ 默认情况下，网页使用的都是相对路径。
- ◆ HTML路径是相对于当前网页<base>标签的href属性。

```
<head>  
  <base href="http://127.0.0.1/WebApp/" />  
</head>  
<body>  
  <form action="cal" method="post">  
    <input type="submit" name="query" value="query" />  
  </form>  
</body>
```

与base标签的href同路径

- ◆ 以上代码，在用户点击提交按钮后，请求的URL将会改为：

URL: http://127.0.0.1/WebApp/cal

■ 表单匹配Servlet的规则:

- ◆ 如果没有设置<base>标签那么action属性以当前浏览器的URL为相对路径。

浏览器 URL: **http://127.0.0.1/WebApp/cal.html**

```
<head>
</head>
<body>
  <form action="cal" method="post">
    <input type="submit" name="query" value="query" />
  </form>
</body>
```

与base标签的href同路径

- ◆ 以上代码，在用户点击提交按钮后，请求的URL将会改为:

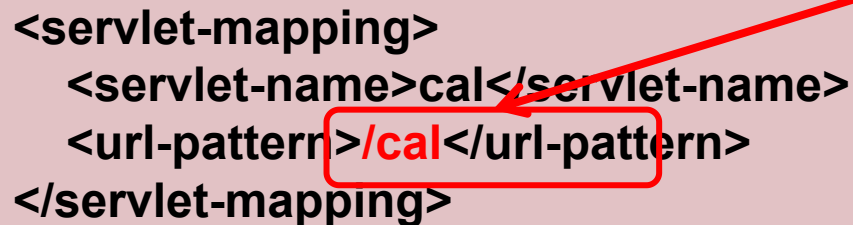
http://127.0.0.1/WebApp/cal

■ 表单匹配Servlet的规则:

◆ 表单提交后的URL一定要和Servlet的<url-pattern>配置保持一致。

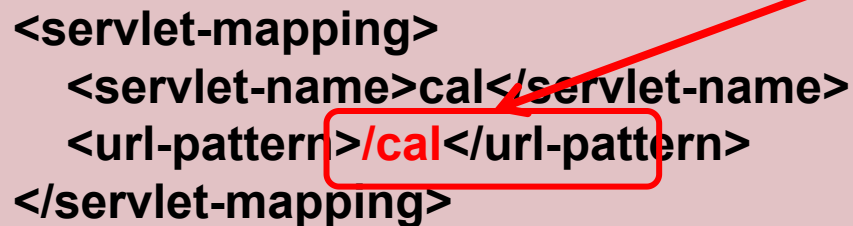
浏览器 URL: http://127.0.0.1/WebApp/cal

```
<servlet-mapping>
  <servlet-name>cal</servlet-name>
  <url-pattern>/cal</url-pattern>
</servlet-mapping>
```



浏览器 URL: http://127.0.0.1/WebApp/cal

```
<servlet-mapping>
  <servlet-name>cal</servlet-name>
  <url-pattern>/cal</url-pattern>
</servlet-mapping>
```



■ Servlet的生命周期

- ◆ 默认情况下：Servlet在第一次被客户端访问是会被Web容器构建。构建完毕的**Servlet将会一直保存在Web应用的Servlet容器内**。
- ◆ 如果Web容器不销毁（例如：Tomcat不停止）Servlet将会一直处于“生存状态”。
- ◆ 与普通Java SE的对象对比可以发现：
 - Servlet的生命周期由于与容器保持一致，因此Servlet对象所处的作用域比较持久。
 - Servlet的生命周期往往与Web容器的运行与停止密切相关。


```
public class HelloWorldServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD>");  
        out.println("<TITLE>Hello World </TITLE>");  
        out.println("</HEAD>");  
        out.println("<BODY>");  
        out.println("<B>Hello, World </B>");  
        out.println("</BODY>");  
        out.println("</HTML>");  
        out.close();  
    }  
}
```

部署:

将编译后的Servlet类, 拷贝到发布目录下WEB-INF/classes下, 如果属于包中类, 需要按照包的结构建立目录;

编辑WEB-INF下web.xml, 在相应位置添入如下:

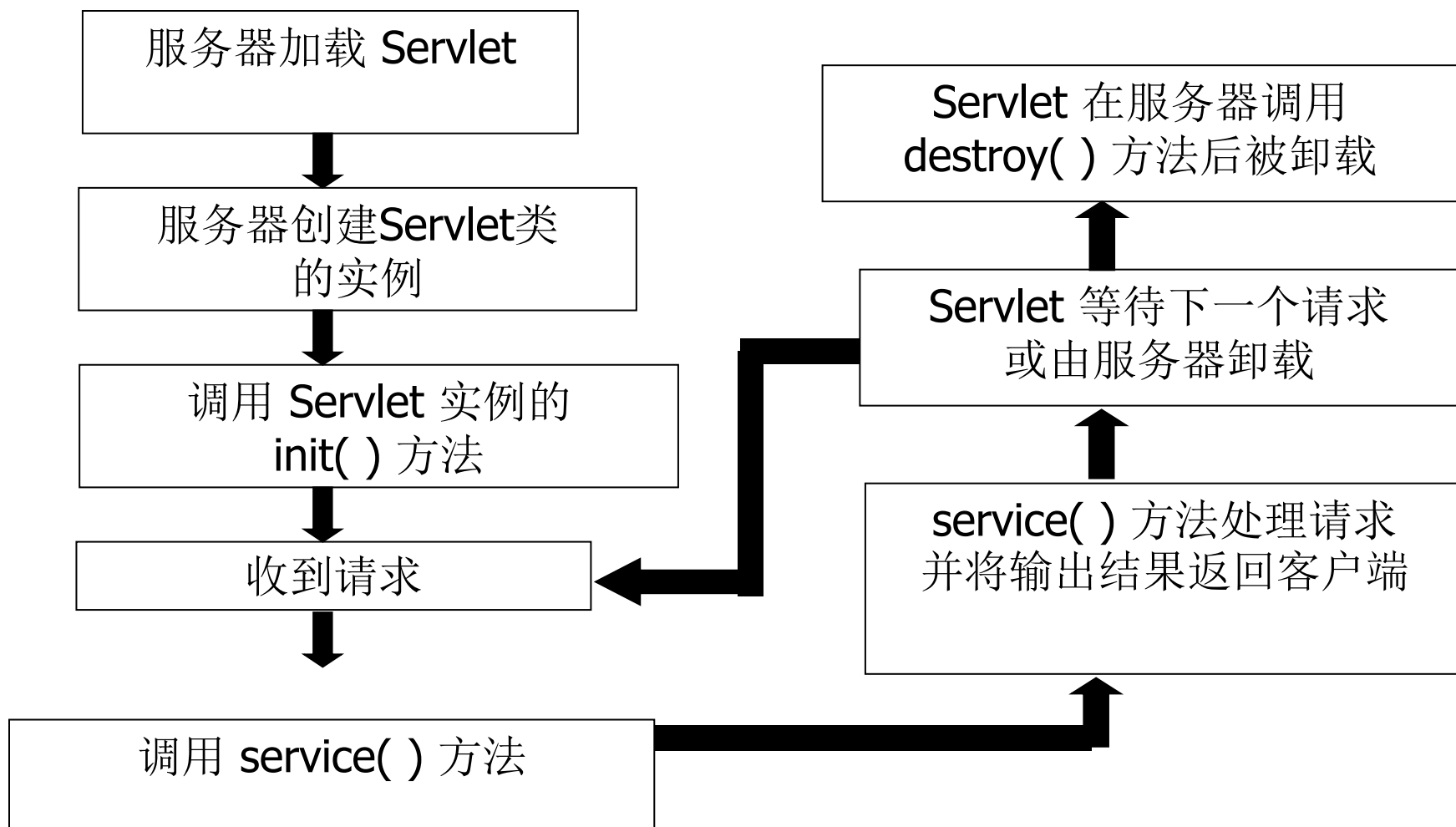
```
<servlet>
  <servlet-name>servlet名称</servlet-name>
  <servlet-class>对应的类</servlet-class>
</servlet>
...
<servlet-mapping>
  <servlet-name>servlet名称</servlet-name>
  <url-pattern>/servleturl1</url-pattern>
</servlet-mapping>
```

...

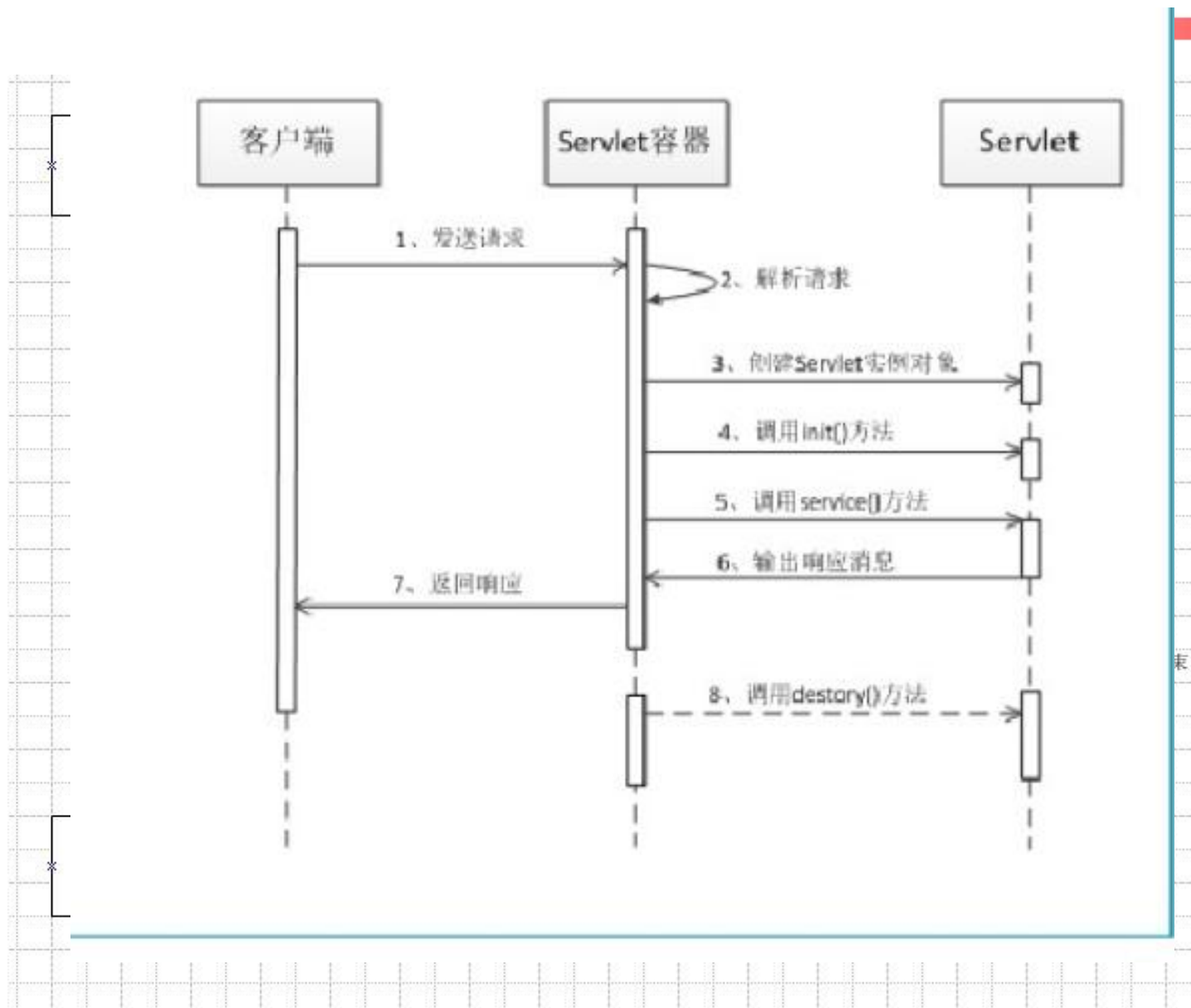
运行:

在浏览器输入发布的URL+servlet映射的地址即可

Servlet的生命周期



Servlet的生命周期



- Servlet 生命周期
- Servlet 服务器(容器)负责管理Servlet的生命周期
- 三个阶段涉及的方法
 - ◆ Init()
 - ◆ service() [doPost(),doGet()]
 - ◆ destroy()
- service阶段使用请求/响应方式进行通信

- 服务器构造Servlet实例后调用
- 执行 Servlet 初始化 - 创建或加载 Servlet 在处理请求时使用的对象
- 可以覆盖 init() 方法，添加其他初始化内容

- service()方法一般不去覆盖它；service()方法用于分析用户的请求, 根据用户的请求类型;调用不同的处理方法;如:doPost(), doGet()等
- 通常我们重写的是doPost()或doGet方法

- 卸载 Servlet 时，调用 destroy () 方法
- 释放获得的资源
- 服务器在完成所有服务调用之后，或者在经过服务器特定的秒数之后调用 destroy() 方法


```

public class HelloWorldServlet extends HttpServlet {
    @Override
    public void init(ServletConfig arg0) throws ServletException {
        System.out.println(" Initialization code... ");
        String username = arg0.getInitParameter("username");
        System.out.println("username: " + username);
    }
    @Override
    public void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        super.service(req, resp);
        System.out.println(" service code... ");
        String method = req.getMethod();
        if(method.equals("GET"))
        {
            long lastModified = getLastModified(req);
            if(lastModified == -1L)
            {
                doGet(req, resp);
            }
        }
    }
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        System.out.println(" do Get code... ");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello World </TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<B>Hello, World </B>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}

```

```

<servlet>
  <servlet-name>helloServlet</servlet-name>
  <servlet-
class>com.servlet.HelloWorldServlet</servlet-
class>
  <init-param>
    <param-name>username</param-
name>
    <param-value>xiaoan</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>helloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>

```

注意:Servlet中的成员变量的使用

- 在Servlet中定义成员变量;使用时一定要注意;
 - ◆ 根据Servlet的生命周期,可以知道,一个Servlet在web应用只有一个实例,这个实例被多个线程共享。
 - ◆ 当多个客户端并发访问同一个Servlet时,web服务器会为每一个客户端的访问请求创建一个线程,并在这个线程上调用Servlet的service方法,因此service方法内如果访问了同一个资源的话,就有可能引发线程安全问题。

```
public class ServletDemo3 extends HttpServlet {  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        /**  
        * 当多线程并发访问这个方法里面的代码时，会存在线程安全问题吗  
        * i变量被多个线程并发访问，但是没有线程安全问题，因为i是doGet方法里面的局部变量，  
        * 当有多个线程并发访问doGet方法时，每一个线程里面都有自己的i变量，  
        * 各个线程操作的都是自己的i变量，所以不存在线程安全问题  
        * 多线程并发访问某一个方法的时候，如果在方法内部定义了一些资源(变量，集合等)  
        * 那么每一个线程都有这些东西，所以就不存在线程安全问题了  
        */  
        int i=1;  
        i++;  
        response.getWriter().write(i);  
    }  
  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        doGet(request, response);  
    }  
  
}
```

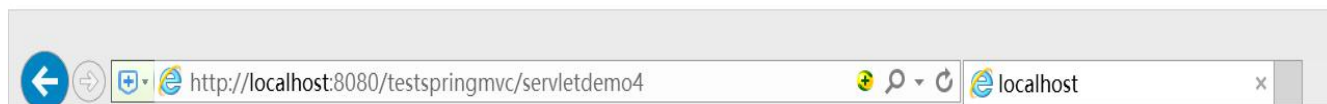
```
public class ServletDemo3 extends HttpServlet {
```

```
    int i=1;
```

```
    // 1.1.1 测试ServletDemo3
```



```
6
```



```
6
```

```
}
```

- `getServletConfig()` //获取Servlet配置对象
 - ◆ `getInitParameter()`
- `getServletContext()` //获取Servlet上下文即Servlet环境
- `getServletInfo()` //获取Servlet描述信息

```
<servlet>
    <servlet-name>servlet名称
</servlet-name>
    <servlet-class>对应的类
</servlet-class>
    <init-param>
        <param-name>
            pageSize
        </param-name>
        <param-value>
            10
        </param-value>
    </init-param>
</servlet>
```

```
//此servlet依赖初始参数才能运行
class LoginServlet extends HttpServlet{
    String pageSize;
    //第一种：在init时读入
    public void init(ServletConfig sc){
        pageSize =
            sc.getInitParameter("pageSize");
        //aa = sc.getInitParameter("aa");
    }
    //第二种：该类中其他地方
    public void doGet(HttpServletRequest req,HttpServletResponse rsp){
        ServletConfig sc =
            this.getServletConfig();
        strURL=sc.getInitParameter("pageSize");
    }
}
```

- servlet主要用于处理用户请求;生成动态网页; 而用户的请求有多种, 那么servlet如何处理不同请求呢?
- 在HttpServlet类中有几个方法;可以在自定义的servlet中重写对应的方法, 去处理客户对应的请求;
- HttpServlet处理请求的方法有:
 - ◆ doHeader //用于处理HEADER请求
 - ◆ doGet //用于处理GET请求, 也可以自动的支持HEADER请求
 - ◆ doPost //用于处理POST请求
 - ◆ doPut //用于处理PUT请求
 - ◆ doDelete //用于处理DELETE请求

- 虽然请求的种类较多,Servlet也提供了对应的处理方法;但在目前的应用中,主要是Get与Post请求;
- Post请求主要是由用户提交表单(且表单是以post方式提交)时产生的请求;
- Get请求主要是如:打开IE浏览器直接在地址栏输入Servlet的访问地址时产生的请求;表单也可以以get方式提交产生Get请求;

- 表单是HTML中使用最广泛的传递信息的手段,也是用户与系统交互主要方式; 掌握Servlet与表单的交互,就在客户端与服务器之间架起了一座桥梁。
- 首先我们来处理以post提交表单的请求(即post请求):
 - ◆ HTML中请单元素如:<form name="form1" **method="POST"** **action="/servlet/NormFormPostServlet"**>其中method用于设置表单提交方式; action用于指定表单提交给谁处理.

http://localhost:8099/test2/form/normpostform.html - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 前进 停止 刷新 主页 搜索 收藏夹 历史记录 打印 邮件 任务栏 快速启动 任务栏 快速启动

地址(1) http://localhost:8099/test2/form/normpostform.html 转到

用户名:	<input type="text"/>
密 码:	<input type="password"/>
性 别:	<input type="radio"/> 男 <input type="radio"/> 女
学 历:	小学
爱 好:	<input type="checkbox"/> 上网 <input type="checkbox"/> 游戏 <input type="checkbox"/> 聊天 <input type="checkbox"/> 旅游 <input type="checkbox"/> 逛街
简 介:	<input type="text" value="这个家伙很懒..."/>
<input type="button" value="提交"/> <input type="button" value="重置"/>	

完毕 本地 Intranet

```
<html>
<head><title>普通POST表单</title></head>
<body><center>
<form name="form1" method="POST" action="/test2/servlet/NormFormPostServlet">
  <table width="100%" border="1">
    <tr>
      <td width="30%" align="right">用户名:</td>
      <td width="70%"><input name="suser" type="text" id="suser"></td>
    </tr>
    <tr>
      <td align="right">密   码:</td>
      <td><input name="spwd" type="password" id="spwd"></td>
    </tr>
    <tr>
      <td align="right">性   别:</td>
      <td><input type="radio" name="ssex" value="1">
        男
           <input type="radio" name="ssex" value="2">
        女</td>
    </tr>
  </table>
</form>
</body>
</html>
```

```
<tr><td align="right">学 历:</td>
  <td><select name="sxl" id="sxl">
    <option value="11">小学</option> <option value="12">初中</option>
    <option value="13">高中</option> <option value="14">大学</option>
  </select>
</td>
</tr><tr>
  <td align="right">爱 好:</td>
  <td><input name="sah" type="checkbox" id="sah" value="21"> 上网
    &nbsp;&nbsp;<input name="sah" type="checkbox" id="sah" value="22"> 游戏
    &nbsp; <input name="sah" type="checkbox" id="sah" value="23"> 聊天
    &nbsp; <input name="sah" type="checkbox" id="sah" value="24"> 旅游
    &nbsp; <input name="sah" type="checkbox" id="sah" value="25"> 逛街</td>
</tr><tr>
  <td align="right">简 介:</td>
  <td><textarea name="sjj" id="sjj">这个家伙很懒...</textarea></td>
</tr>
<tr>
  <td colspan="2" align="center"><input type="submit" name="Submit" value="提交">
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type="reset" name="Submit2" value="重置"></td>
</tr>
</table>
</form>
</center></body>
</html>
```

- 表单以POST方式提交, 即POST请求; servlet调用doPost()处理;
- 调用doPost()时系统会传入两个参数
 - ◆ HttpServletRequest对象封装请求中数据(HTTP头部)及客户端信息(如: 客户机的IP地址等)
 - ◆ HttpServletResponse 客户端响应(如输出显示数据)

■ 获取post请求提交的数据方法:

- ◆ String request.getParameter(“表单元素名”);
 - //获取用户在表单输入的数据(如:用户名,密码等);其表单元素名在表单中只对应一个输入;如表单元素名对应多个表单中元素,用此方法则只能获取第一个输入;
- ◆ String[] request.getParameterValues(“表单元素名”);
 - //获取用户在表单输入的数据(如:爱好 等);其中**表单元素名在表单中对应多个元素**;
- ◆ getParameterNames(): 如果你想要得到一个当前请求的所有参数的完整列表, 那么调用该方法。

■ 其它方法:

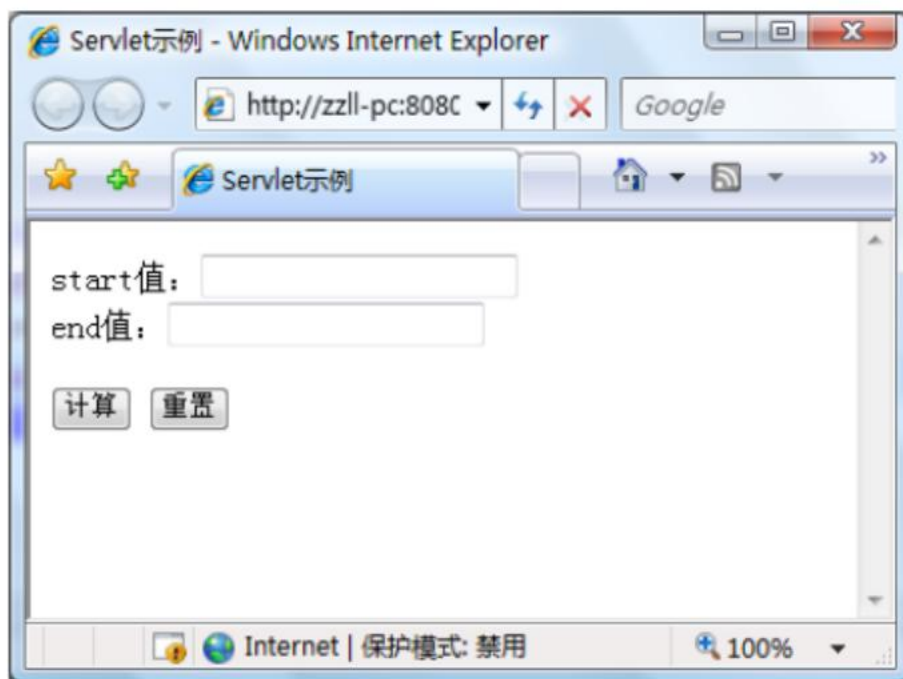
- ◆ String request.getRemoteAddr() //获取客户端IP
- ◆ String request.getScheme() //获取协议名
- ◆ String request.getServerName() //获取ip或机器名或域名
- ◆ String request.getServerPort() //获取端口号
- ◆ String request.getContextPath() //获取上文路径名
- ◆ request.setCharacterEncoding(arg0) //设置request对象的数据字符集

```
public class NormFormPostServlet extends HttpServlet {  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {  
        //获取表单数据.  
        String suser=request.getParameter("suser");  
        String spwd=request.getParameter("spwd");  
        String ssex=request.getParameter("ssex");  
        String sxl=request.getParameter("sxl");  
        String[] sah=request.getParameterValues("sah");  
        String sjj=request.getParameter("sjj");  
        //输出显示在网页上.  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("用户名:"+suser+"<br>");  
        out.println("密码:"+spwd+"<br>");  
        out.println("性别:"+ssex+"<br>");  
        out.println("学历:"+sxl+"<br>");  
        for(int i=0;i<sah.length;i++) { out.println("爱好:"+sah[i]+"<br>"); }  
        out.println("简介:"+sjj+"<br>");  
    }  
}
```

```
Enumeration<String> paramNames =  
request.getParameterNames();  
//获取所有的参数名  
while (paramNames.hasMoreElements()) {  
    String name =paramNames.nextElement();  
    //得到参数名  
    String value=request.getParameter(name);  
    //通过参数名获取对应的值  
    System.out.println(name+"="+value);  
}
```

- 上面servlet在获取表单中用户输入的数据时;当数据中包括中文时会出现乱码.
- 解决方法:
 - ◆ 首先设置响应结果字符集; `response.setCharacterEncoding("UTF-8");`
或`response.setContentType("text/html; charset=UTF-8");`
 - ◆ 然后在获取数据前设置request编码:
`request.setCharacterEncoding("UTF-8");`
 - ◆ 或对获取到的数据转编码:`new String (suser.getBytes("ISO-8859-1"),"UTF-8");` //注:http协议在网络中传输数据采用的编码是:ISO-8859-1.

练习：servlet计算start到end的累加和




■ 代码参考

前台页面:

```
<form action="servlet/getdata" method="post">  
  start: <input type="text" name="start"><br>  
  end: <input type="text" name="end"><br>  
  <input type="submit" value="计算">  
  <input type="reset" value="重置">  
</form>
```

后台servlet: getdata.java

```
public void service(...){  
  ...  
  int start=Integer.parseInt(request.getParameter("start"));  
  int end=Integer.parseInt(request.getParameter("end"));  
  int sum=0;  
  for(int i=start;i<=end;i++)sum+=i;  
  ...  
}
```

 转换一下

```
//在Servlet中除了可以用从请求中获取用户输入的数据外;还可以获取客户端
//相关信息;
public class NormFormPostServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse
    response)
    throws ServletException, IOException {
    String cname = request.getRemoteHost(); // 获取客户端主机
    String cip = request.getRemoteAddr(); // 获取客户端IP
    String pro = request.getScheme(); // 获取协议名
    String sip = request.getServerName(); // 获取ip或机器名或域名
    int sport = request.getServerPort(); // 获取端口号
    String sap = request.getContextPath(); // 获取上文路径名

    //输出显示在网页上.
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("你的IP:" + cip + "<br>");
    //...
}
}
```

- 表单以GET方式提交, form设置如: `<form name= “form1”
method= “GET”
action= “/test2/servlet/NormFormPostServlet” >`
- GET请求由Servlet中的doGet()方法处理; 处理方式方法和doPost完全相同;
- 一个servlet可同时处理POST, GET请求; 则实现doPost或doGet后, 在 doGet()中直接调用doPost ();或在doPost()中直接调用doGet();

```
public class NormFormPostServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        //获取表单数据.  
        String suser=request.getParameter("suser");  
        String spwd=request.getParameter("spwd");  
        String ssex=request.getParameter("ssex");  
        String sxl=request.getParameter("sxl");  
        String[] sah=request.getParameterValues("sah");  
        String sjj=request.getParameter("sjj");  
        //输出显示在网页上.  
        response.setContentType("text/html");  
        response.setCharacterEncoding("gb2312");  
        PrintWriter out = response.getWriter();  
        out.println("用户名:"+suser+"<br>");  
        out.println("密码:"+spwd+"<br>");  
        out.println("性别:"+ssex+"<br>");  
        out.println("学历:"+sxl+"<br>");  
        for(int i=0;i<sah.length;i++)  
            out.println("爱好:"+sah[i]+"<br>");  
        out.println("简介:"+sjj+"<br>");  
    }  
}
```

- 除表单可以发送Get请求外, 在IE地址栏直接输入servlet地址发送请求是最常见的Get请求;
- 在地址栏输入地址发送Get请求, 也可以传数据到服务端; 在上面表单以Get方式提交发送请求时大家应该已经看到, 这时的数据是以查询串的形式发送; 如: `http://localhost:8099/norServlet?参数名1=值1&参数名2=值2&参数名3=值3`;
- `Request.getQueryString()`; //可获取整个查询串; 获取参数名对应的数据用:
- `request.getParameter(“参数名”);`
`request.getParameterValues(“参数名”);`
- 例(servlet代码在下页)
<http://localhost:8099/test2/servlet/GetMethodServlet?param1=11¶m2=21¶m3=31¶m3=32¶m33>

```
public class GetMethodServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse  
        response)  
        throws ServletException, IOException {  
        request.setCharacterEncoding("gb2312");  
        String v1 = request.getParameter("param1");  
        String v2 = request.getParameter("param2");  
        String[] v3 = request.getParameterValues("param3");  
        response.setContentType("text/html; charset=gb2312");  
        PrintWriter out = response.getWriter();  
  
        String queryString=request. getQueryString();//获取请求字符串  
        out.println( "?号后查询串是:" +queryString+ " <br" );  
        out.println("v1: " + v1 + "<br>");  
        out.println("v2: " + v2 + "<br>");  
        for (int i = 0; i < v3.length; i++)  
            out.println("v3: " + v3 + "<br>");  
    }  
}
```

■ Get请求与Post请求的特点:

- ◆ Get请求以地址栏中查询串形式向服务端传数据
- ◆ Post请求将数据封装入请求报头中向服务端传数据

■ 何时使用Get请求

- ◆ 数据量较小(不能大于2k)
- ◆ 要传数据不涉及隐私(如:密码)

■ 何时使用Post请求

- ◆ 数据量较大
- ◆ 数据涉及个人隐私(如:密码)

- 包含文件域元素的表单可实现文件上传等实用功能;
- 表单form设置如: `<form action= “ ” method= “POST” enctype= “multipart/form-data” name= “form1” >` 其中method必需是POST且加属性 `enctype= “multipart/form-data”` ;
- 上传单个文件, 您应该使用单个带有属性 `type=“file”` 的 `<input .../>` 标签。
- 对于文件域表单servlet无法通过元素名获取值: `getParameter()`; `getParameterValues()`; 无法使用
可通过 `request.getInputStream()` 获取流; 再分析流来取出相应数据; 但分析流很复杂; 一般我们引入第三方包来分析处理; 这将在后面介绍。

表单 `method` 属性应该设置为 `POST` 方法, 不能使用 `GET` 方法。

```
■ <%@ 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>文件上传实例 </title>  
■ </head>  
■ <body>  
■ <h1>文件上传实例 </h1>  
■ <form method="post" action="upload" enctype="multipart/form-data">  
■     选择一个文件:  
■     <input type="file" name="uploadFile" />  
■     <br/><br/>  
■     <input type="submit" value="上传" />  
■ </form>  
■ </body>  
■ </html>
```

- 需要引入的 jar 文件：commons-fileupload-1.3.2、commons-io-2.5.jar。

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response){
    DiskFileItemFactory factory = new DiskFileItemFactory();
    factory.setSizeThreshold(MEMORY_THRESHOLD);
    ....
    factory.setRepository(new File(System.getProperty("java.io.tmpdir")));
    ServletFileUpload upload = new ServletFileUpload(factory);
    List<FileItem> formItems = upload.parseRequest(request);
    if (formItems != null && formItems.size() > 0) {
        for (FileItem item : formItems) {
            // 处理不在表单中的字段
            if (!item.isFormField()) {
                String fileName = new File(item.getName()).getName();
                String filePath = uploadPath + File.separator + fileName;
                File storeFile = new File(filePath);
                item.write(storeFile);
            }
        }
    }
}
```

■ 请求:	HttpServletRequest
■ 响应:	HttpServletResponse
■ Servlet配置相关:	ServletConfig
■ Servlet异常类:	ServletException
■ 会话跟踪:	HttpSession
■ Servlet上下文:	ServletContext
■ 请求转发器:	RequestDispatcher
■ Cookie:	Cookie

- HttpServletRequest对象代表客户端的请求，当客户端通过HTTP协议访问服务器时，**HTTP请求头中的所有信息都封装在这个对象中**，通过这个对象提供的方法，可以获得客户端请求的所有信息。

getScheme()	//获取协议名
getServerName()	//获取ip或机器名或域名
getServerPort()	//获取端口号
getContextPath()	//获取上文路径名
getSession()	//获取通信会话HttpSession对象
getQueryString()	//获取请求字符串
getRemoteHost()	//获取客户端主机
getRemoteAddr()	//获取客户端IP
getRequestDispatcher()	//获取请求转发对象，注意参数一定要以“/”开头
getCookies()	//获取客户端发送来的Cookie对象数组
getServletPath()	//获取Servlet对应的路径
setCharacterEncoding(arg0)	//设置request对象的数据字符集

■ 获得客户机信息

- getRequestURL方法返回客户端发出请求时的完整URL。
- getRequestURI方法返回请求行中的资源名部分。
- getQueryString 方法返回请求行中的参数部分。
- getPathInfo方法返回请求URL中的额外路径信息。额外路径信息是请求URL中的位于Servlet的路径之后和查询参数之前的内容，它以“/”开头。
- getRemoteAddr方法返回发出请求的客户机的IP地址。
- getRemoteHost方法返回发出请求的客户机的完整主机名。
- getRemotePort方法返回客户机所使用的网络端口号。
- getLocalAddr方法返回WEB服务器的IP地址。
- getLocalName方法返回WEB服务器的主机名。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String requestUrl = request.getRequestURL().toString();//得到请求的URL地址
    String requestUri = request.getRequestURI();//得到请求的资源
    String queryString = request.getQueryString();//得到请求的URL地址中附带的参数
    String remoteAddr = request.getRemoteAddr();//得到来访者的IP地址
    String remoteHost = request.getRemoteHost();
    int remotePort = request.getRemotePort();
    String remoteUser = request.getRemoteUser();
    String method = request.getMethod();//得到请求URL地址时使用的方法
    String pathInfo = request.getPathInfo();
    String localAddr = request.getLocalAddr();//获取WEB服务器的IP地址
    String localName = request.getLocalName();//获取WEB服务器的主机名

}

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}

}
```

- `getParameter(String)` 方法(常用)
- `getParameterValues(String name)` 方法(常用)
- `getParameterNames()` 方法(不常用)
- `getParameterMap()` 方法(编写框架时常用)


```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Html的Form表单元素</title>
</head>
<fieldset style="width:500px;">
    <legend>Html的Form表单元素</legend>
    <form action="{pageContext.request.contextPath}/servlet/RequestDemo03" method="post">
        编 &nbsp;&nbsp;号(文本框):
        <input type="text" name="userid" value="NO." size="2" maxlength="2"><br>
        用户名(文本框): <input type="text" name="username" value="请输入用户名"><br>
        密 &nbsp;&nbsp;码(密码框):
        <input type="password" name="userpass" value="请输入密码"><br>
        性 &nbsp;&nbsp;别(单选框):
        <input type="radio" name="sex" value="男" checked>男
        <input type="radio" name="sex" value="女">女<br>
        部 &nbsp;&nbsp;门(下拉框):
        <select name="dept">
            <option value="技术部">技术部</option>
            <option value="销售部" SELECTED>销售部</option>
            <option value="财务部">财务部</option>
        </select><br>
        兴 &nbsp;&nbsp;趣(复选框):
        <input type="checkbox" name="inst" value="唱歌">唱歌
        <input type="checkbox" name="inst" value="游泳">游泳
        <input type="checkbox" name="inst" value="跳舞">跳舞
        <input type="checkbox" name="inst" value="编程" checked>编程
        <input type="checkbox" name="inst" value="上网">上网
        <br>
        说 &nbsp;&nbsp;明(文本域):
        <textarea name="note" cols="34" rows="5">
        </textarea>
```

```
public class RequestDemo03 extends HttpServlet {public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    String userid = request.getParameter("userid");
    String username = request.getParameter("username");//获取填写的用户名
    String userpass = request.getParameter("userpass");//获取填写的密码
    String sex = request.getParameter("sex");//获取选中的性别
    String dept = request.getParameter("dept");//获取选中的部门
    String[] insts = request.getParameterValues("inst");
    String note = request.getParameter("note");//获取填写的说明信息
    String hiddenField = request.getParameter("hiddenField");//获取隐藏域的内容
    String instStr="";
    for (int i = 0; insts!=null && i < insts.length; i++) {
        if (i == insts.length-1) {
            instStr+=insts[i];
        }else {
            instStr+=insts[i]+",";
        }
    }
    ...
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

- Web服务器收到客户端的http请求，会针对每一次请求，分别创建一个用于代表请求的request对象、和代表响应的response对象。

request和response对象即然代表请求和响应，那我们要获取客户机提交过来的数据，只需要找request对象就行了。要向客户机输出数据，只需要找response对象就行了。

■ **setContentType(arg0)**

- ◆ 设置JSP页面的显示方式,如'text/html', 'text/html; charset=gb2312' 等

■ **setCharacterEncoding(arg0)**

- ◆ 设置页面的编码方式:比如gb2312, gbk, big5, 还有UTF-8等

■ **addCookie(Cookie)**

- ◆ 往客户端添加一个Cookie

■ **sendRedirect(String)**

- ◆ 重定向请求

■ **response.setHeader(name, contect);**

- ◆ 设置html头head。

ServletOutputStream	getOutputStream() Returns a ServletOutputStream suitable for writing binary data in the response.
java.io.PrintWriter	getWriter() Returns a PrintWriter object that can send character text to the client.

getOutputStream和getWriter这两个方法互相排斥，调用了其中的任何一个方法后，就不能再调用另一方法。

getOutputStream和getWriter方法分别用于得到输出二进制数据、输出文本数据

```
public class ResponseDemo01 extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        outputChineseByOutputStream(response); //使用OutputStream流输出中文  
    }  
    public void outputChineseByOutputStream(HttpServletResponse response) throws  
        IOException {  
        String data = "中国";  
        OutputStream outputStream = response.getOutputStream();  
        //获取OutputStream输出流  
        response.setHeader("content-type", "text/html; charset=UTF-8");  
        byte[] dataByteArr = data.getBytes("UTF-8");  
        //将字符转换成字节数组, 指定以UTF-8编码进行转换  
        outputStream.write(dataByteArr); //使用OutputStream流向客户端输出字节数组  
    }  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        doGet(request, response);  
    }  
}
```

```
public class ResponseDemo01 extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
        String data = "中国";  
        //通过设置响应头控制浏览器以UTF-8的编码显示数据，如果不加这句话，那么浏览器  
显示的将是乱码  
        response.setCharacterEncoding("UTF-8");  
        //设置将字符以"UTF-8"编码输出到客户端浏览器  
        PrintWriter out = response.getWriter();//获取PrintWriter输出流  
        out.write("<meta http-equiv=' content-type'  
content=' text/html; charset=UTF-8' />");  
        out.write(data);//使用PrintWriter流向客户端输出字符  
    }  
    public void doPost(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
        doGet(request, response);  
    }  
}
```

```

public class ResponseDemo03 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setHeader("refresh", "5");//设置refresh响应头控制浏览器每隔5秒钟刷新一次
        //1. 在内存中创建一张图片
        BufferedImage image = new BufferedImage(80, 20, BufferedImage.TYPE_INT_RGB);
        //2. 得到图片
        //Graphics g = image.getGraphics();
        Graphics2D g = (Graphics2D)image.getGraphics();
        g.setColor(Color.WHITE);//设置图片的背景色
        g.fillRect(0, 0, 80, 20);//填充背景色
        //3. 向图片上写数据
        g.setColor(Color.BLUE);//设置图片上字体的颜色
        g.setFont(new Font(null, Font.BOLD, 20));
        g.drawString(makeNum(), 0, 20);
        //4. 设置响应头控制浏览器浏览器以图片的方式打开
        response.setContentType("image/jpeg");//等同于response.setHeader("Content-Type", "image/jpeg");
        //5. 设置响应头控制浏览器不缓存图片数据
        response.setDateHeader("expries", -1);
        response.setHeader("Cache-Control", "no-cache");
        response.setHeader("Pragma", "no-cache");
        //6. 将图片写给浏览器
        ImageIO.write(image, "jpg", response.getOutputStream());
    }

    private String makeNum(){
        Random random = new Random();
        String num = random.nextInt(9999999)+
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < 7-num.length(); i++) {
            sb.append("0");
        }
        num = sb.toString()+num;
        return num;
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}

```


■ 重定向

- ◆ 请求重定向指：一个web资源收到客户端请求后，通知客户端去访问另外一个web资源，这称之为请求重定向。
- ◆ 应用场景：用户登陆，用户首先访问登录页面，登录成功后，就会跳转到某个页面，这个过程就是一个请求重定向的过程
- ◆ HttpServletResponse: `sendRedirect(String url);`

```
public class ResponseDemo04 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        /**1. 调用sendRedirect方法实现请求重定向,
        * sendRedirect方法内部调用了
        * response.setHeader("Location",
        "/JavaWeb_HttpServletResponse_Study_20140615/index.jsp");
        * response.setStatus(HttpServletResponse.SC_FOUND); //设置302状态码, 等同于
        response.setStatus(302);
        */
        response.sendRedirect("/JavaWeb_HttpServletResponse_Study_20140615/index.jsp");

        //2. 使用response设置302状态码和设置location响应头实现重定向实现请求重定向
        //response.setHeader("Location",
        "/JavaWeb_HttpServletResponse_Study_20140615/index.jsp");
        //response.setStatus(HttpServletResponse.SC_FOUND); //设置302状态码, 等同于
        response.setStatus(302);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

■ 请求转发

- ◆ 请求转发 发生在服务器端; 一个web组件(如:servlet)接收到一个请求后, 自身不做处理, 而是转发后另一个web组件(如:servlet)处理; 在服务器端完成;
- ◆ 客户端地址栏地址不发生变化;两个web组件间共享request。
- ◆ 请求转发与重定相比: 请求转发相当于只是一个请求;效率更高。
- ◆ RequestDispatcher: forward(HttpServletRequest, HttpServletResponse);

■ 包含

- ◆ RequestDispatcher: include(HttpServletRequest, HttpServletResponse);
- ◆ 包含是指一个web组件将另一个web组件的结果插入到包含语句处; 实现组件内容合并。两个web组件间共享request。
- ◆ /**
- ◆ * 2.forward
- ◆ * 客户端请求某个web资源, 服务器跳转到另外一个web资源, 这个forward也是给服务器用的,
- ◆ * 那么这个"/"就是给服务器用的, 所以此时"/"代表的就是web工程
- ◆ */
- ◆ this.getServletContext().getRequestDispatcher("/index.jsp").forward(request, response);

```

public class CheckAccount extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

```

← → ↻ ⓘ localhost:8080/testspringmvc/login.jsp

username:

password:

提交

```
letResponse resp) throws ServletException, IOException {
```

```

if ((username != null) && (username.trim().length() > 0)) {
    if ((pwd != null) && (pwd.trim().length() > 0)) {
        System.out.println("xxx success");
        session.setAttribute("username", username);
        req.setAttribute("username", username);
        // resp.sendRedirect("success.jsp");
        req.getRequestDispatcher("success.jsp").forward(req, resp);
        return;
    }
}
resp.sendRedirect("fail.jsp");
return;
}

```

← → ↻ ⓘ localhost:8080/testspringmvc/login?username=abc&pwd=123

■ 配置Servlet初始化参数

- Servlet的配置文件web.xml中，可以使用一个或多个<init-param>标签为servlet配置一些初始化参数

```
<servlet>
  <servlet-name>ServletConfigDemo1</servlet-name>
  <servlet-class>gac.servlet.study.ServletConfigDemo1</servlet-class>
  <init-param>
    <param-name>name</param-name>
    <param-value>gac</param-value>
  </init-param>
  <init-param>
    <param-name>password</param-name>
    <param-value>123</param-value>
  </init-param>
  <init-param>
    <param-name>charset</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</servlet>
```

- 通过ServletConfig获取Servlet的初始化参数
- 当servlet配置了初始化参数后，web容器在创建servlet实例对象时，会自动将这些初始化参数封装到ServletConfig对象中，并在调用servlet的init方法时，将ServletConfig对象传递给servlet

```
public class ServletConfigDemo1 extends HttpServlet {  
    private ServletConfig config;  
    public void init(ServletConfig config) throws ServletException {  
        this.config = config;  
    }  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        //获取在web.xml中配置的初始化参数  
        String paramVal = this.config.getInitParameter("name");//获取指定的初始化参数  
        response.getWriter().print(paramVal);  
        response.getWriter().print("<hr/>");  
        //获取所有的初始化参数  
        Enumeration<String> e = config.getInitParameterNames();  
        while(e.hasMoreElements()) {  
            String name = e.nextElement();  
            String value = config.getInitParameter(name);  
            response.getWriter().print(name + "=" + value + "<br/>");  
        }  
    }  
}
```

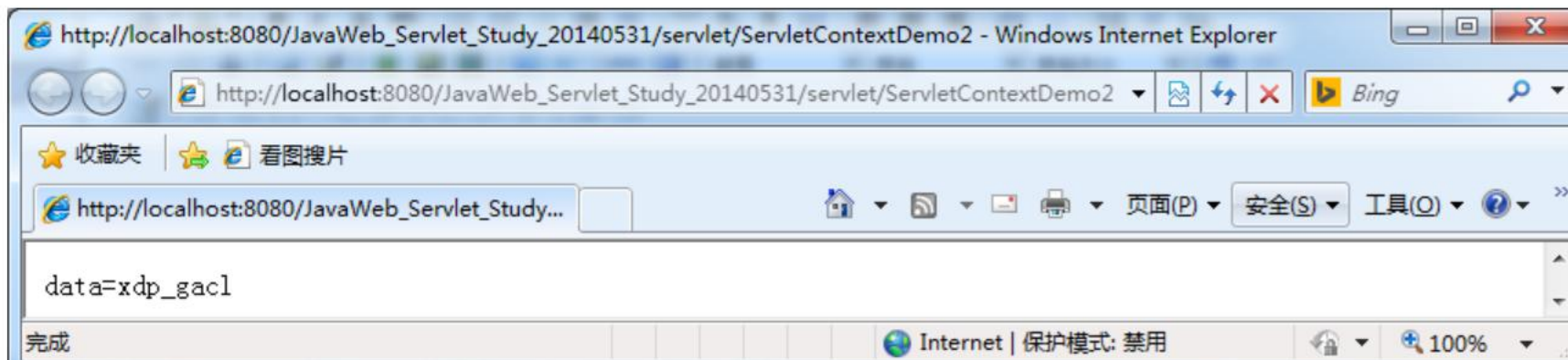

- WEB容器在启动时，它会为每个WEB应用程序都创建一个对应的ServletContext对象，它代表当前web应用。
- ServletConfig对象中维护了ServletContext对象的引用，开发人员在编写servlet时，可以通过ServletContext.getServletContext方法获得ServletContext对象。

由于一个WEB应用中的所有Servlet共享同一个ServletContext对象，因此Servlet对象之间可以通过ServletContext对象来实现通讯。
ServletContext对象通常也被称之为context域对象。

```
public class ServletContextDemo1 extends HttpServlet {  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        String data = "xdp_gacl";  
        /**  
        * ServletConfig对象中维护了ServletContext对象的引用，开发人员在编写  
servlet时，  
        * 可以通过ServletConfig.getServletContext方法获得ServletContext对象。  
        */  
        ServletContext context = this.getServletConfig().getServletContext();  
        获得ServletContext对象  
        context.setAttribute("data", data); //将data存储到ServletContext对象中  
    }  
  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        doGet(request, response);  
    }  
}
```

```
■ public class ServletContextDemo2 extends HttpServlet {  
■     public void doGet(HttpServletRequest request,  
    HttpServletResponse response)  
■         throws ServletException, IOException {  
■         ServletContext context = this.getServletContext();  
■         String data = (String) context.getAttribute("data");//  
    从ServletContext对象中取出数据  
■         response.getWriter().print("data="+data);  
■     }  
  
■     public void doPost(HttpServletRequest request,  
    HttpServletResponse response)  
■         throws ServletException, IOException {  
■         doGet(request, response);  
■     }  
■ }
```

- 先运行ServletContextDemo1，将数据data存储到ServletContext对象中，然后运行ServletContextDemo2就可以从ServletContext对象中取出数据了，这样就实现了数据共享



- (HttpServletRequest) 客户端发送一个请求到服务端时产生，服务端处理完这个请求时结束。
- 相关方法：
 - ◆ public void setAttribute(String var, Object obj);
 - var 作用域变量名; obj 作用域变量对应的值;
 - ◆ public Object getAttribute(String var);
 - var 作用域变量名;
 - ◆ Public void removeAttribute(String var);
 - var 作用域变量名;

```
■ public class CheckAccount extends HttpServlet {  
■     @Override  
■     protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
■         doGet(req, resp);  
■     }  
  
■     @Override  
■     public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
■         HttpSession session = req.getSession();  
■         String username = req.getParameter("username");  
■         String pwd = req.getParameter("pwd");  
■         RequestDispatcher dispatcher = null;  
■         if ((username != null) && (username.trim().equals("abc"))) {  
■             if ((pwd != null) && (pwd.trim().equals("123"))) {  
■                 System.out.println("xxx success" + username);  
■                 session.setAttribute("username", username);  
■                 req.setAttribute("username", "reqxxxxxxxxxx");  
■                 // resp.sendRedirect("success.jsp");  
■                 req.getRequestDispatcher("success.jsp").forward(req, resp);  
■                 return;  
■             }  
■         }  
■         resp.sendRedirect("fail.jsp");  
■         return;  
■     }  
■ }
```

```
■ <%@ page language="java" pageEncoding="UTF-8"%>
■
■ <html>
■ <head><title>The Counter Page</title></head>
■ <body>
■   welcome to
■     <% String username=(String )session.getAttribute("username");
■       out.print("session..." +username);
■       String usernamexx=(String )request.getAttribute("username");
■       out.print("request..." +usernamexx); %>
■ <p>

■   </body>

■ </html>
■
■ </body>
■ </html>
```

- WEB开发中，服务器可以为每个用户浏览器创建一个会话对象（session对象），注意：一个浏览器独占一个session对象(默认情况下)。因此，在需要**保存用户数据**时，**服务器程序可以把用户数据写到用户浏览器独占的session中**，当用户使用浏览器访问其它程序时，其它程序可以从用户的session中取出该用户的数据，为用户服务

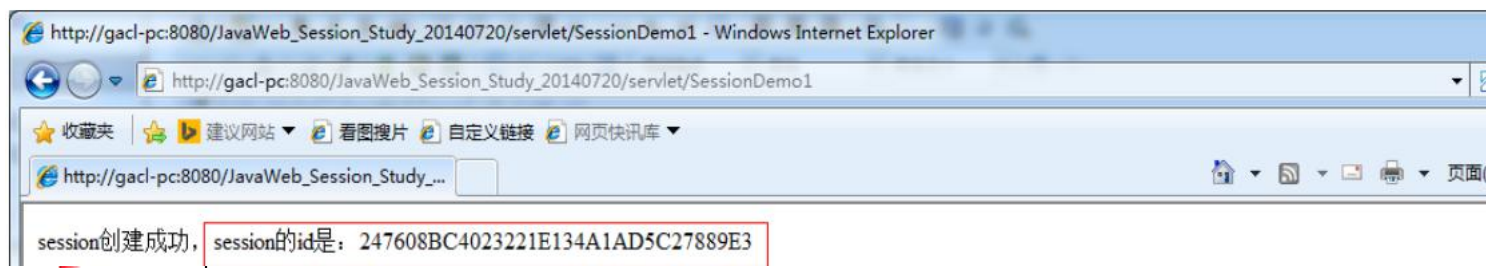
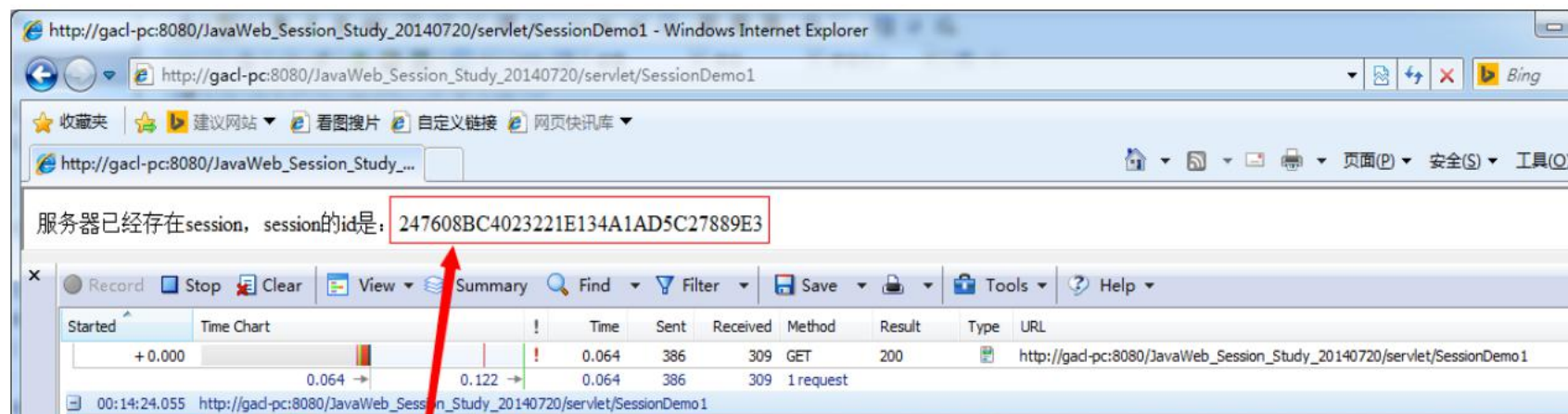
服务器是如何实现一个session为一个用户浏览器服务

- 服务器创建session出来后，会把session的id号，以cookie的形式回写给客户机，这样，只要客户机的浏览器不关，再去访问服务器时，都会带着session的id号去，服务器发现客户机浏览器带session id过来了，就会使用内存中与之对应的session为之服务

- (HttpSession) session产生后在session的有效期内有效
- 每个客户端独享自己一个session对象
- 相关方法:
 - ◆ get/set/removeAttribute() //操作属性的方法
 - ◆ getCreateTime() //毫秒,以1970-01-01为起点 获取Session创建时间;
 - ◆ getLastAccessedTime(); //(单位:毫秒) 获取最后一次访问的时间;
 - ◆ setMaxInactiveInterval(-1); //(单位:秒) 设置超时时间;
 - ◆ getMaxInactiveInterval(); //(单位:秒) 获取超时时间;
 - ◆ isNew(); //判断是否是新创建的;
 - ◆ invalidate(); //注销Session 让Session对象与用户断开;
 - ◆ getServletContext(); //获取上下文对象;

```
public class SessionDemo1 extends HttpServlet {  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
    {  
        HttpSession session = request.getSession();  
    }  
}
```

点击刷新按钮，再次请求服务器，此时就可以看到浏览器再请求服务器时，会把存储到cookie中的session的Id一起传递到服务器端了，如下图所示：



- (ServletContext) application用来存储和获取当前应用内的变量;
- 对所有客户端共享; web应用加载时启动时候产生web应用关闭时消亡
- 可以理解成Servlet容器环境信息构成了一个ServletContext
- 相关方法
 - ◆ Set/get/removeAttribute(); //操作作用域的变量;
 - ◆ getRequestDispatcher(String) //获取转发器;
 - ◆ getRealPath(String) //获取绝对路径;

- Cookie是服务器端或javascript用于维护客户端的一种方式;Cookie是驻留在客户端的一个文本文件;
- 作用:
 - ◆ 在客户端保存用户相关信息
- Cookies的两个主要用途: 储存用户信息和个性化定制。
- Cookies最典型的应用是判定注册用户是否已经登录网站

- 客户端用户如果设置禁止 **COOKIES**, 则 **Cookie** 不能建立。并且在客户端, 一个浏览器能
- 创建的 **Cookie** 数量最多为 300 个, 并且每个不能超过 4KB, 每个 Web 站点能设置的
- **Cookie** 总数不能超过 20 个。
- **Cookie** 在生成时就会被指定一个 **Expire** 值,
- 这就是 **Cookie** 的生存周期, 在这个周期内 **Cookie** 有效, 超出周期 **Cookie** 就会被清除。
- 有些页面将 **Cookie** 的生存周期设置为 “0” 或负值,
- 这样在关闭浏览器时, 就马上清除 **Cookie**, 不会记录用户信息, 更加安全。
- 如果在一台计算机中安装多个浏览器, 每个浏览器都会在各自己独立的空间存放 **cookie**。
- 因为 **cookie** 中不但可以确认用户, 还能包含计算机和浏览器的信息,
- 所以一个用户用不同的浏览器登录或者用不同的计算机登录, 都会得到不同的 **cookie** 信息。
- 另一方面, 对于在同一台计算机上使用同一浏览器的多用户群,
- **cookie** 不会区分他们的身份, 除非他们使用不同的用户名登录。
- **Cookie** 是利用了网页代码中的 **HTTP** 头信息进行传递的, 浏览器的每一次网页请求,
- 都可以伴随 **Cookie** 传递, 例如, 浏览器的打开或刷新网页操作。
- 服务器将 **Cookie** 添加到网页的 **HTTP** 头信息中, 伴随网页数据传回到你的浏览器,
- 浏览器会根据你电脑中的 **Cookie** 设置选择是否保存这些数据。
- 如果浏览器不允许 **Cookie** 保存, 则关掉浏览器后, 这些数据就消失。
- **Cookie** 在电脑上保存的时间是不一样的, 这些都是由服务器的设置不同决定的。
- **Cookie** 有一个 **Expires** (有效期) 属性, 这个属性决定了 **Cookie** 的保存时间,
- 服务器可以通过设定 **Expires** 字段的数值, 来改变 **Cookie** 的保存时间。
- 如果不设置该属性, 那么 **Cookie** 只在浏览网页期间有效, 关闭浏览器,
- 这些 **Cookie** 自动消失, 绝大多数网站属于这种情况。通常情况下, **Cookie** 包含 **Server**、**E**

- 通过HttpServletRequest对象获取客户端的Cookie信息：
getCookies()
- 通过HttpServletResponse对象往客户端添加Cookie信息：
addCookie(Cookie)
- Cookie类相关方法：
 - ◆ set/getName()
 - ◆ set/getValue()
 - ◆ set/getMaxAge() 失效时间
 - ◆ set/getPath() 适用的路径
 - ◆ set/getDomain() 适用的域名


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

    Cookie cookie1=new Cookie("login_name","admin");
    cookie1.setMaxAge(60*60*24*360);
    Cookie cookie2=new Cookie("login_pwd","123456");
    cookie2.setMaxAge(60*60*24*360);
    Cookie cookie3=new
        Cookie("login_sex", java.net.URLEncoder.encode
            ("男", "gb2312")); //中文要转编码.
    cookie3.setMaxAge(60*60*24*360);
    response.addCookie(cookie1);
    response.addCookie(cookie2);
    response.addCookie(cookie3);

}
```



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

    Cookie[] cookies = request.getCookies();
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("  <HEAD><TITLE>get cookies</TITLE></HEAD>");
    out.println("  <BODY>");

    for (int i = 0; i < cookies.length; i++) {
        out.println(cookies[i].getName() + " = ");
        out.println(cookies[i].getValue());
        // 如果值为中文
        // out.println(java.net.URLDecoder.decode(cookies[i].getValue(), " GBK" ));
        out.println("<br>");
    }

    out.println("  </BODY>");
    out.println("</HTML>");
    out.flush();
    out.close();
}
```

■ Servlet 3.0规范新特性

- ◆ 通过注释简化了开发工作

 - 通过引入Web片段减少了框架的配置

- ◆ 引入异步处理以提高响应性能

- ◆ 现有API的改进

■ 为了能运行使用Servlet 3.0开发的Servlet, Servlet 容器必须运行在Java SE 6或更高版本中

■ Servlet中的注释

- ◆ 可以将@WebServlet用于继承自javax.servlet.http.HttpServlet的Servlet类，从而无需在Web部署描述符（web.xml）中建立Servlet条目了
- ◆ 注释@WebServlet具有name、urlPatterns和initParams，等，可以通过它们来定义Servlet的行为。

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

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

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        // TODO Auto-generated method stub
        System.out.println("do .... get");
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @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);
    }
}
```

■ Servlet中的异步处理

- ◆ 异步处理允许线程发出一个资源调用，然后直接返回到容器而无需等待。该线程进而可以执行其他任务
 - 使得Servlet 3.0的性能有了很大的提升
- ◆ 可以通过@WebServlet和@WebFilter注释的asyncSupported属性来支持Servlet 3.0的异步处理

- Filter，或称过滤器，是servlet 2.3 新增加的功能。Filter可认为是Servlet的一种“加强版”，它主要用于对用户请求进行预处理，以及对服务器响应进行后处理。Filter负责过滤的Web组件可以是Servlet、JSP、HTML。

■ Filter的用处

- 能够在Servlet被调用前检查Request对象，修改Request header和Request内容。
- 能够在Servlet被调用后检查Response对象，修改Response header和Response内容。

在实际开发中，使用Filter可以实现代码的复用。

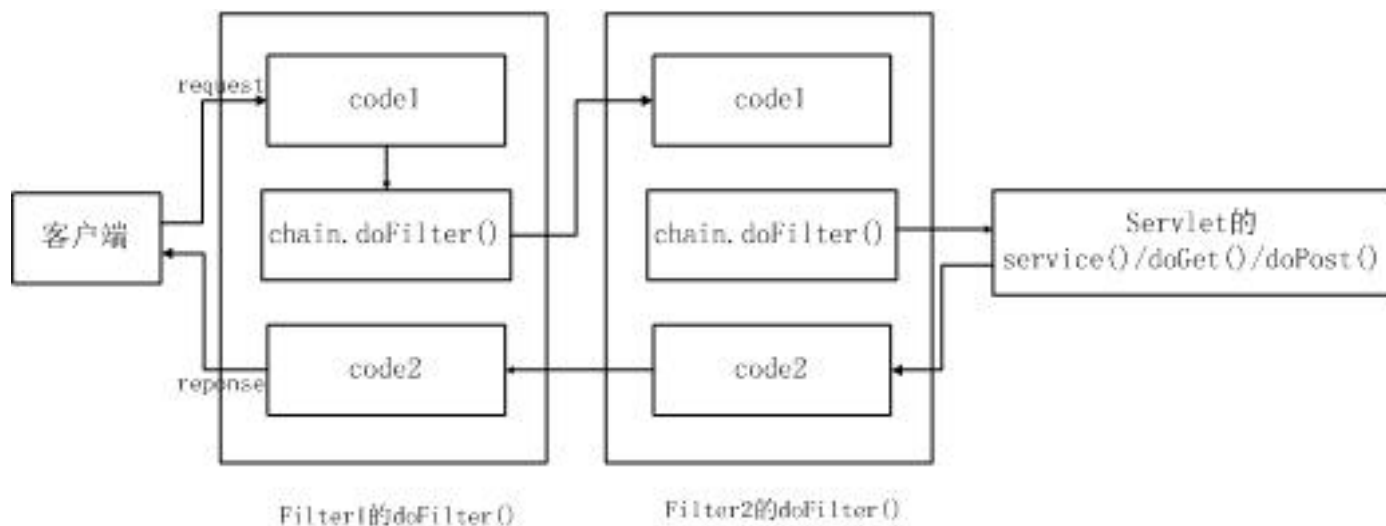
■ Filter 接口定义了如下三个方法。

- `void init(FilterConfig config)`: 完成Filter初始化。Servlet容器创建Filter实例后调用一次Init方法。在这个方法中可读取web.xml文件中的初始化参数。
- `void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`: 完成实际的过滤操作。
- `void destroy()`: Servlet容器在销毁Filter实例前调用该方法, 释放Filter占用的资源。

- FilterChain接口 ， FilterChain接口中定义了 doFilter([ServletRequest](#) request, [ServletResponse](#) response) 方法。
- FilterChain 调用链中的下一个过滤器
 - ◆ 如果没有下一个过滤器，则到达用户最终想访问的Web组件。

- Filter可拦截多个用户请求或响应，一个请求或响应也可被多个Filter拦截。拦截之后，Filter就可以进行一些通用处理：如访问权限控制、编码转换、记录日志等。
- 权限控制Filter示例
 - ◆ 该Filter会验证用户是否登录，若用户没有登录，系统直接跳转到登录页面，从而避免了未验证用户通过粘贴已登录用户的URL地址串非法闯入系统的隐患。

- 在进行Filter配置时，可以把多个过滤器串联组装成一条链，然后依次执行其中的doFilter()方法。



```
//@WebFilter("/AuthorityFilter")
public class AuthorityFilter implements Filter {
    public AuthorityFilter() {
        // TODO Auto-generated constructor stub
    }

    public void destroy() {
        // TODO Auto-generated method stub
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
    IOException, ServletException {
        // TODO Auto-generated method stub
        // place your code here
        System.out.println("do filter");
        System.out.println("do filter");
        // pass the request along the filter chain
        HttpServletRequest requ=(HttpServletRequest) request;
        String requestpath=requ.getServletPath();
        HttpSession session=requ.getSession();
        String requestPath=requ.getServletPath();
        if((session.getAttribute("user")==null)&&(!requestPath.endsWith("login.html"))
            &&(!requestPath.endsWith("CheckAccount")))
        ) {
            System.out.println("fail xxxx");
            /*PrintWriter out = response.getWriter();
            out.println("<html><head><title>登录结果</title></head>");
            out.println("<body> please login <br>");
            out.println("-----<br>");
            out.println("</body></html>");
            out.close();*/
            HttpServletResponse resp=(HttpServletResponse) response;
            resp.sendRedirect("login.html");
        }
    }
}
```

- 作业:
- 书Page 83 编写servlet
- Page 89 编写Filter

- 在Eclipse环境中如何开发Java Web项目
 - ◆ 在Eclipse中，新建Java Web项目，设置项目名称为“ServletDemo”。
 - ◆ 新建一个XHTML文件名，取名为“login.html”。

<body>

请输入用户名和密码:

```
<!-- 登录表单，该表单提交到一个Servlet -->
```

```
<form id="login" method="post" action="
    /ServletDemo/Servlet/LoginServlet">
```

用户名:

密 码： <input type="password" name="pass"
width="50"/>

</body>

- ◆ 在生成的项目上，点击右键，点击“New” — “Servlet”，新建一个Servlet，在弹出的对话框中，设置Java package为“edu.hdu.web”，类名为“LoginServlet”。
- ◆ 在对话框中，设置Servlet名为“LoginServlet”，Servlet的URL为“/Servlet/LoginServlet”。也可手工创建和修改web.xml，以配置Servlet和URL的映射关系。
- ◆ 编写edu.hdu.web.LoginServlet类。由于表单提交的是post请求，因此需要修改LoginServlet类中的“doPost”方法，以完成Servlet响应。

LoginServlet程序的示例源代码为：

```
@WebServlet("/Servlet/LoginServlet")  
public class LoginServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
  
    public LoginServlet() {  
        super();  
    }  
}
```



```
protected void doGet(HttpServletRequest request,  
    HttpServletResponse response) throws ServletException,  
    IOException {  
    doPost(request, response);  
}
```

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    request.setCharacterEncoding("utf-8"); //处理中文输入乱码
    String username = request.getParameter("username");
    String password = request.getParameter("pass");
    response.setContentType("text/html;charset=utf-8"); //处理
    中文输出乱码
```

```
PrintWriter out = response.getWriter();
    out.println("<html><head><title>登录结果</title></head>");
    out.println("<body> 您输入的用户名是: " + username +
"<br>");
out.println("-----<br>");
    out.println("您输入的密码是: " + password +
"</body></html>");
    out.close();
}
}
```

- Cookie的基本概念。一个Cookie就是一个存储在浏览器端的键/值对，因此Cookie功能都必须要有浏览器的支持才行。当用户打开的网页中包含创建Cookie的程序代码时，服务器端则创建Cookie数据，然后将这个Cookie传送到客户端的计算机上，存储在浏览器当中，此后服务器端的网页都可以访问这个Cookie的数据内容。

FilterIP.java

```
package servlets;  
  
public class FilterIP implements Filter {  
    private FilterConfig filterconfig;  
    private String filterIP="";  
  
    @Override  
    public void init(FilterConfig config) throws ServletException {  
        this.filterconfig=config;  
        filterIP=config.getInitParameter("FilterIP");  
    }  
  
    @Override  
    public void destroy() {  
        this.filterconfig=null;  
    }  
}
```

定义两个变量

从web.xml中获取过滤器配置

过滤器初始化
获取被过滤的IP

根据参数名获取过滤器配置的参数值

销毁过滤器

//后续

@Override

```
public void doFilter(ServletRequest request, ServletResponse response,  
    FilterChain chain) throws IOException, ServletException {
```

```
    response.setContentType("text/html;charset=gb2312");
```

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

```
    //定义错误处理页面:
```

```
    RequestDispatcher dispatcher=request.getRequestDispatcher("error.jsp");
```

```
    String remoteIP=request.getRemoteAddr(); //获取客户端IP地址
```

```
    if( remoteIP.equals(filterIP) ){
```

```
        dispatcher.forward(request, response); //转到错误处理页面
```

```
        //不转向可直接: out.print("对不起, 你的IP不能访问本站");
```

```
    }
```

```
    else{
```

```
        chain.doFilter(request, response); //继续客户端请求
```

```
    }
```

```
}
```

```
}//end class
```

Servlet过滤器工作原理（续集）

步骤2：在web.xml中手工配置Servlet过滤器：

