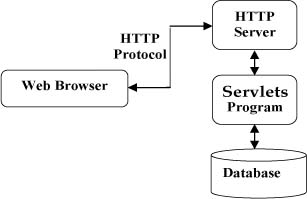
# Servlet概述

## 1.1概述

Servlet是运行在web服务器上的JAVA程序，学习servlet的前提是已经掌握了javaSE，Servlet一般作为在来自 Web 浏览器的请求和在 HTTP 服务器上的数据库或应用程序的中间层。(类似于MVC的controller)



## 1.2 servlet的任务和功能

1.读取由客户端（浏览器）发送的显式数据。比如提交的 HTML 表单(form)

2.读取由客户端（浏览器）发送的隐式 HTTP 请求数据。这包括 cookies、媒体类型和浏览器能理解的压缩格式等等。

3.处理数据并生成结果。这个过程可能需要访问数据库，执行 RMI 或 CORBA 调用，调用 Web 服务，或者直接计算响应。

4.发送显式数据（即文档）到客户端（浏览器）。该文档可以以多种多样的格式被发送，包括文本文件（HTML 或 XML）、二进制文件（GIF 图像）、Excel 等。

5.发送隐式的 HTTP 响应到客户端（浏览器）。这包括告诉浏览器或其他客户端被返回的文档类型（例如 HTML），设置 cookies 和缓存参数，以及其他类似的任务。

## 1.3 servlet的安装与配置

Servlet的环境需要安装JAVA的jdk包，并设置相应的环境变量，除此之外我们还需要一个tomcat服务器软件，tomcat是一个开源的，免费的，适用于中小型项目

下载并解压了tomcat服务器后，打开其中的bin目录，点击startup.bat即可运行该软件(同文件夹下的shutdown.bat可以关闭服务器)，可以通过浏览器Localhost:8080来查看tomcat是否已运行

由于JAVA标准版中并没有servlet类，因此还需要为你的编译器指定servlet类的路径(配置环境变量)，网上有太多类似内容，不再赘述

## 1.4 tomcat的配置

### 1.4.1 虚拟目录

所有的想要运行的网站应用，都得放在服务器软件指定目录(如：D:\Downloads\apache-tomcat-8.5.23\webapps\ROOT)，在浏览器地址栏输入：http://localhost:8080/，访问的其实就是tomcat默认项目根目录(ROOT目录)下的index.jsp文件。这明显不符合我们的要求，虚拟目录可以帮助我们解决这个问题

打开tomcat根目录下的conf文件夹，找到server.xml，使用文本编辑器打开，查找HOST节点元素，默认的HOST节点如下：

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true">

<!-- SingleSignOn valve, share authentication between web applications

Documentation at: /docs/config/valve.html -->

<!--

<Valve className="org.apache.catalina.authenticator.SingleSignOn" />

-->

<!-- Access log processes all example.

Documentation at: /docs/config/valve.html

Note: The pattern used is equivalent to using pattern="common" -->

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log" suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

1. **name属性：**这是你的主机名，它被IP地址唯一标识(localhost一般意为本地主机，在windows的HOSTS文件中将其IP设为127.0.0.1)，发布到internet上后，这里应当填写你的域名
2. **appBase：**基准目录 ，基准目录下的Root 目录被自动设置为根目录

**修改虚拟目录只需要在HOST节点内加入：**

 <Context path="/appName/filePath" docBase="/opt/filePath" reloadable="true" debug="0"/>

**配置完成后记得保存配置文件并重启服务器！！！！！！！！**

1. path ： 虚拟web文件路径，为空时意为根目录，当访问path目录下的文件时，实则访问docBase目录下的文件。
2. docBase :文件自定义的根目录。注意如果项目中有需要上传文件，则将上传的文件路径设置为docBase，而不是path。(如果使用了框架，则此处填写框架的入口)

eg：<Context path="" docBase="D:\test"></Context>

假设有文件D:\test \1.html，此时输入localhost:8080/1.html即访问D:\test \1.html

<Context path="hello" docBase="D:\test"></Context>

假设有文件D:\test\index.html，此时输入http://localhost:8080/hello/index.html即访问D:\test\index.html

### 1.4.2 虚拟主机

在项目很小时，为了节约成本，一台服务器需要跑多个项目(对企业来说更多的是多台服务器跑一个项目)，虚拟主机技术可以帮助我们实现这一点

同样打开conf/server.xml文件，查找Engine子节点，它应当是HOST的父节点

<Engine name="Catalina" defaultHost="localhost"> </Engine>

每当我们需要多配置一台虚拟主机(一个新项目)，只需要在Engine下多增加一个HOST子节点即可，HOST节点的属性中，name填写域名，appBase填写网站项目的根目录

配置虚拟主机记得修改windows的HOSTS文件，除非你已经买了域名

**配置完成后记得保存配置文件并重启服务器！！！！！！！！**

eg：

<Host name="1.com" appBase="D:\test" unpackWARs="true" autoDeploy="true">

<Context path="/hello" docBase="D:\test\public" debug="0" privileged="true"/>

</Host>

假设有文件D:\test\public\index.html，输入<http://1.com:8080/hello/index.html>即可访问该文件

## 1.5 servlet的生命周期

1.通过调用 init () 方法 servlet 被初始化。

2.Servlet 调用 service() 方法来处理客户端的请求。

3.通过调用 destroy() 方法 servlet 终止。

4.最后，servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

### 1.5.1 init()方法

init()方法只在servlet创建时调用，当用户调用一个 Servlet 时，就会创建一个 Servlet 实例。每一个用户请求都会产生一个新的线程，适当的时候移交给 doGet 或 doPost 方法

**一个servlet在运行时只会生成一个实例**

public void init() throws ServletException {

// 初始化代码...

}

}

### 1.5.2 service()方法

通过调用 service() 方法来处理来自客户端（浏览器）的请求，并把生成的响应反馈给客户端。

每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法，因此我们不需要对该方法做任何操作，只需要重写对应的doGet、doPost、doPut，doDelete 等方法即可

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

{

//代码

}

### 1.5.3 doGet()和doPost()方法

所有的get请求和post都是通过doGet()和doPost()进行处理，doPut()和doDelete()相对使用较少

public void doGet(HttpServletRequest request,HttpServletResponse response)

throws ServletException, IOException

{

// Servlet 代码

}

public void doPost(HttpServletRequest request,HttpServletResponse response)

throws ServletException, IOException

{

// Servlet 代码

}

**这里一定要注意：不能出现super.doGet()或者super.doPost()，不然会出现405错误**

可以看一下父类(**HttpServlet**)的源码：

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 String protocol = req.getProtocol();  
 String msg = lStrings.getString("http.method\_get\_not\_supported");  
 if (protocol.endsWith("1.1")) {  
 resp.sendError(405, msg);  
 } else {  
 resp.sendError(400, msg);  
 }  
}

### 1.5.4 destroy()方法

destroy()方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy()方法会执行诸如Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘等清理操作

**一个servlet可以手动调用destroy来销毁，此外当容器(如tomcat)关闭时候也会自动销毁该实例（而不是关闭浏览器或者标签）**

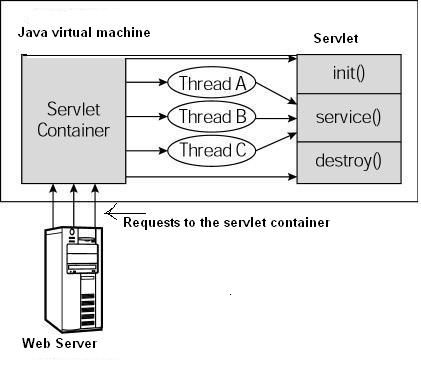
destroy()方法会将servlet对象被标记为垃圾回收

public void destroy() {

// 终止化代码...

}

### 1.5.5 servlet架构



1. web服务器收到客户端(如浏览器)的request请求
2. web服务器将请求委派给servlet容器(如tomcat)处理，处理完成后在将结果返回给服务器，由web服务器将结果反馈给用户(tomcat)

这里需要额外解释一下web服务器，web容器和应用服务器的区别

(1) **Web server**，Web服务器，是指能够为发出请求的浏览器提供文档的程序。服务器是一种被动程序，只有浏览器发出请求的时候才会响应。应用层使用的是HTTP协议。目前最主流的三个Web服务器是Apache、Nginx和IIS。

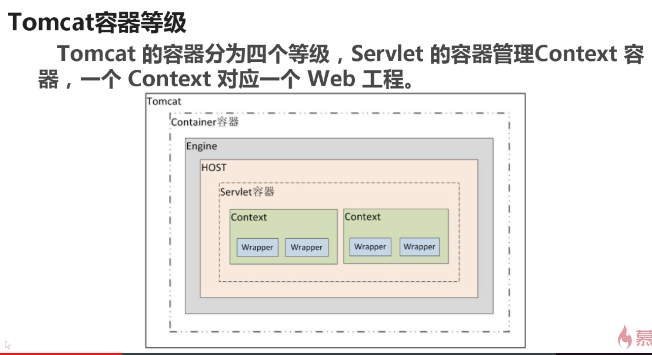
(2)**Web容器**是一种服务器程序，在服务器端口提供相应服务。最典型的例子如Tomcat，因此Tomcat没办法为每个虚拟主机单独配置一个端口，而是所有虚拟主机共用一个端口——8080。一个服务器可以有多个容器。

(3) **应用服务器**

Web服务器设计服务于HTTP内容，而应用服务器不只限于HTTP。Web服务器服务于静态内容，有插件支持动态语言，

应用服务器也具有Web服务器的这些东西，除此它还支持程序级的服务，如连接池，事务支持，信息服务等。

**Tomcat既是servlet的web容器，但是也可以单独作为web服务器使用**



1. Servlet 容器加载 Servlet，之后调用service() 方法，由service()方法决定由doGet()还是doPost()处理请求
2. Servlet容器处理由多个线程产生的多个请求，每个线程执行一个单一的 Servlet 实例的 service() 方法。也就是说servlet是单例多线程的，多个http请求都由一个servlet对象进行处理(因此servlet是线程不安全的)，但每个http请求都会有一个单独的线程处理
3. servlet容器将处理的结果反馈给web服务器，由web服务器将其转发给用户，当然由于tomcat也可以作为web服务器，因此如果服务器上只安装了tomcat，tomcat也可以自行承担起web服务器的转发作用

# Servlet的使用

## 2.1 HelloWorld

Servlet 是服务 HTTP 请求并实现 javax.servlet.Servlet 接口的 Java 类，实际上相应的api包里已经为我们提供了相应的类——**javax.servlet.http.HttpServlet，**我们只需要继承HttpServlet类即可

首先从一个helloworld开始。

首先创建一个java web project，新建一个类

}

**public** **class** HelloWorld **extends** HttpServlet {

**private** String str = **null**;// str是线程不安全的，servlet对象中开启的任何线程都可以随意修改该变量，请尽量避免这种写法

@Override

**public** **void** init() **throws** ServletException {

str = "helloWorld!";

}

@Override

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html");// 设置HTTP响应头，告诉HTTP头这次发送的响应内容是text/html类型的

PrintWriter out = response.getWriter();

str = "<h1>" + str + "</h1>";

out.println(str);

}

@Override

**public** **void** destroy() {

// **TODO** Auto-generated method stub

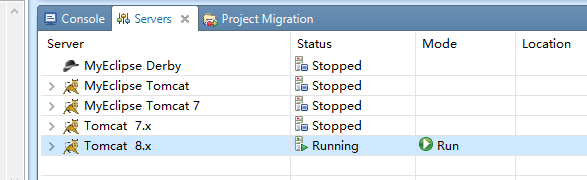
**super**.destroy();

}

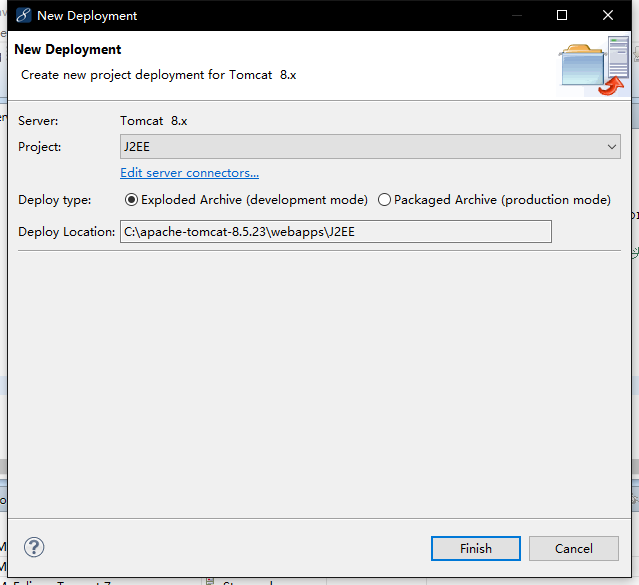
}

点击编译，然后进入到项目对应的文件夹下，找到该java文件编译后产生的.class文件，接下来，我们需要将其部署到tomcat上

首先，加载你的tomcat到myeclpse上并运行



右键添加当前JAVA项目到tomcat中



打开tomcat的conf/tomcat-user.xml文件，在<tomcat-users>节点中添加以下子节点(账号密码自己设置，重启tomcat)

<role rolename="admin-gui"/>

<role rolename="admin-script"/>

<role rolename="manager-gui"/>

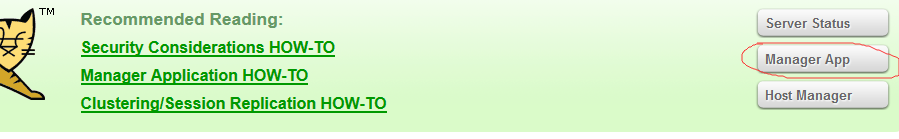
<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<user username="admin" password="admin" roles="manager-gui,manager-script,manager-jmx,manager-status,admin-script,admin-gui"/>

打开localhost:8080，点击manager app查看项目是否已被添加，点击项目即可访问该项目，但我们还未指定helloworld接口的url，并不能出现helloworld



最后一步，打开项目根目录，在WebRoot \WEB-INF下新建一个web.xml文件(建议使用myeclpse新建web.xml)，并输入

<web-app>

<servlet>

<!—随意起，但要和下面保持一致 -->

<servlet-name>ServletName</servlet-name>

<!--Servlet的类-->

<servlet-class>xxxpackage.xxxServlet</servlet-class>

</servlet>

<servlet-mapping>

<!—必须和上面的保持一致 -->

<servlet-name>ServletName</servlet-name>

<!--映射的url路径 -->

<url-pattern>/aaa/xxx</url-pattern>

</servlet-mapping>

</web-app>

**上述配置将url /aaa/xxx 和 servlet xxxpackage.xxxServlet绑定到了一起，至此，访问http://你的域名/aaa/xxx实际上就是访问xxxpackage.xxxServlet这个servlet**

servlet 用来声明一个servlet的数据

servlet-name 指定servlet的名称

servlet-class 指定servlet的类名称

loadon-startup 容器启动时自动加载本servlet，中间数字代表优先越小越高级，eg：< loadon-startup >1</ loadon-startup >

servlet-mapping 用来定义servlet所对应的URL，包含两个子元素

servlet-name 指定servlet的名称

url-pattern 指定servlet所对应的URL

## 2.2 servletContext上下文

什么是应用上下文(servletContext)？

假设说我们有一个WEB应用，这个WEB应用中有10个SERVLET，在这里，这个WEB应用就拥有一个它自己的仓库-ServletContext，而这10个Servlet则共享这个大仓库，并且各自拥有属于他们自己的小仓库-ServletConfig。

ServletContext就是一个全局的概念，它属于应用，但我们有时候不想让某些参数被其他Servlet应用，仅仅想在自己的Servlet中被共享，这时候就需要把它保存在ServletConfig中，换句话说，从***单个Servlet***的角度来看，ServletConfig是它的全局，而从一个***Servlet集合（Web应用）***来看，ServletContext是它的全局。ServletContext是在应用项目启动时，就初始化好的，其生命周期和整个web应用的生命周期相关，与单个servlet的生命周期无关

那么如何配置servletContext和servletConfig呢？这就需要在我们的web.xml中进行配置

### 2.5.1配置servletConfig

在Servlet的配置文件中，可以使用**一个或多个<init-param>**标签为servlet配置一些初始化参数。（配置在某个servlet标签或者整个web-app下）， 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。

**配置：**

<servlet>  
 <servlet-name>testServlet</servlet-name>  
 <servlet-class>com.test.helloWorld.Test</servlet-class>  
 <init-param>  
 <param-name>myParams</param-name>  
 <param-value>你好666</param-value>  
 </init-param>  
</servlet>  
<servlet-mapping>  
 <servlet-name>testServlet</servlet-name>  
 <url-pattern>/test</url-pattern>  
</servlet-mapping>

**使用：**

@Override  
public void doGet(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException  
{  
 response.setContentType("text/html");  
 ServletConfig servletConfig= this.getServletConfig();  
 String myParams=servletConfig.getInitParameter("myParams");  
 PrintWriter w=response.getWriter();  
 w.println(myParams);  
}

### 2.5.2配置servletContext

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。servlet之间通过servletContext实现数据共享**，servletContext的用途如下：

**1.一个servlet将数据共享给其他servlet**

在一个servlet中向servletContext写入数据：

ServletContext context =this.getServletContext(); //servletContext域对象  
context.setAttribute("data","共享数据"); //向域中存了一个data属性

从另一个servlet中读取该data属性：

ServletContext context =this.getServletContext();  
String value = (String)context.getAttribute("data"); //获取域中的data属性

**2.初始化全局共享数据**

**配置：**

<context-param>  
 <param-name>phone</param-name>  
 <param-value>15288888888</param-value>  
</context-param>

**读取：**

ServletContext servletContext=this.getServletContext();  
String myParams1=servletContext.getInitParameter("phone");

**3.** **通过servletContext对象实现servlet转发**

具体参见2.5.1请求转发

**4. 通过servletContext对象读取资源文件**

略(用到再写)

## 2.3 读取表单数据

使用servlet获取表单数据有如下方法，get和post通用：

* **getParameter()：**可以通过调用 request.getParameter() 方法来获取表单参数的值。
* **getParameterValues()：**如果参数出现一次以上，则调用该方法，并返回多个值，例如复选框。
* **getParameterNames()：**如果您想要得到当前请求中的所有参数的完整列表，则调用该方法。该方法返回一个String组成的枚举，包含所有具有name属性的表单元素值

注意:获取的表单元素都是字符串类型

String username = request.getParameter("username");

String password = request.getParameter("password");

**int** sex = Integer.*parseInt*(request.getParameter("sex"));

String[] likes = request.getParameterValues("like");

Enumeration<String> ss=request.getParameterNames();

## 2.4HTTP

### 2.4.1Http请求(rquest)

HttpServletReques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，开发人员通过这个对象的方法，可以获得客户这些信息，该对象是doGet()和doPost()方法的参数，可以直接在doGet()和doPost()方法中使用

常用方法：

eg：[www.baidu.com](http://www.baidu.com):8088/zhidao/question?username=root&password=123&sex=1&like=1

**getRequestURL**方法返回客户端发出请求时的完整URL。

（返回”http://[www.baidu.com](http://www.baidu.com):8088/zhidao/question”）

**getRequestURI**方法返回请求行中的资源名部分。

（返回”/zhidao/question”）

**getQueryString** 方法返回请求行中的参数部分。

（返回” username=root&password=123&sex=1&like=1”）

**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。

比如

<servlet-mapping>  
<servlet-name>TestServlet</servlet-name>  
<url-pattern>/servlet/test/\*</url-pattern>  
</servlet-mapping>

访问http://localhost:8080/JavaProject/servlet/test/hello/a.html，这里的\*可以是任何东西

（返回” /hello/a.html”）

**getCookies()**返回一个cookie数组，包含客户端发送该请求的所有的 Cookie 对象。

**getSession()**返回与该请求关联的当前 session对象，或者如果请求没有 session 会话，则创建一个。需要注意的是，由于session的使用需要session ID，而session id是存储在浏览器端的cookie中的，**如果客户端禁用cookie，可能造成session无法正常使用**

**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

### 2.4.2Http响应(response)

HttpServletResponse是服务器端对客户端请求发出的响应，响应通常包括一个状态行、一些响应报头、一个空行和文档

常用方法：

**sendError(int sc,String msg)**方法手动设置HTTP状态码(sc)，还会向客户发出一条错误信息。并且会清除缓冲区

**addCookie(Cookie cookie)**方法把指定的 cookie 添加到响应

**flushBuffer()**强制任何在缓冲区中的内容被写入到客户端。

**sendRedirect(String location)**使用指定的重定向位置 URL 发送临时重定向响应到客户端。

**setContentType**方法设置响应头，指定本次响应数据的文本类型(具体类型百度参照contentType表)

**addHeader(String name,String value)**将一个名称为name，值为value的响应报头添加到响应中

**encodeURL(String url)**对包含 session ID 的指定 URL 进行编码，或者如果编码不是必需的，则返回原本的URL。

## 2.5 重定向和请求转发

在servlet内部，我们常用两种方法来处理请求，一种是**请求转发(dispatcher)**，另一种是**重定向(redirect)**。

**请求转发(dispatcher)**是request请求在到达servlet后**在服务器内部**被转发给另一个servlet或jsp进行处理的过程

**重定向(redirect)** 是request在到达servlet后，url被重新定位，离开服务器到浏览器，然后浏览器重定向到指定的url上的过程

java提供了request.getRequestDispatcher()和response.sendRedirect()两个方法来帮助我们处理这两种情况，从方法的调用上就可以看出**请求转发(dispatcher)**是对request进行转发，而**重定向(redirect)** 实际上是已经完成处理了，要返回response了，只不过这个response是要求客户端跳转url的响应

区别：

**request.getRequestDispatcher()**是请求转发，前后两阶段共享同一个request ;

**response.sendRedirect()**是重新定向，前后两阶段不是一个request。

**需要注意的是：**

request.getRequestDispather();返回的是一个RequestDispatcher对象。

### 2.5.1请求转发

1.利用ServletRequest接口中的**getRequestDispatcher（String path）**方法。参数是下一个要处理的servlet的url

eg：

request.getRequestDispatcher("/third?name=jay");

2.ServletContext接口中**getNamedDispatcher（String path）**

不同的是request.getRequestDispatcher(url)的url可以是相对路径也可以是绝对路径。

而this.getServletContext().getRequestDispatcher(url)的url只能是绝对路径。

RequestDispatcher接口中定义了二种方法用于请求转发：

**1.forward（ServletRequest，ServletResponse）**：

将请求转发给服务器上另外一个Servlet，JSP页面，或者HTML文件，这个方法必须在响应response被提交给客户端之前调用，否则抛出异常。

方法调用后在响应中的没有提交的内容被自动消除。

**2.include（ServletRequest，ServletResponse）**：

用于在响应中包含其他资源(Servlet，JSP页面或HTML文件)的内容。

即请求转发后，原先的Servlet还可以继续输出响应信息，转发到的Servlet对请求做出的响应将并入原先Servlet的响应对象中。

**3.forward方法和include方法的区别：**

forward方法调用后在响应中的没有提交的内容被自动消除。

include方法使原先的Servlet和转发到的Servlet都可以输出响应信息。

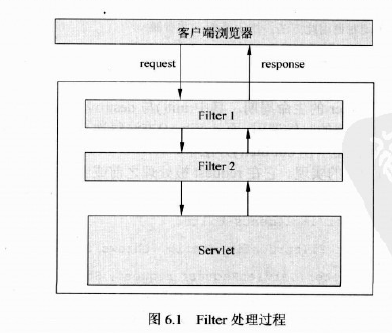
### 2.5.2重定向

## 2.6 servlet过滤器(Filter)

有时候，我们不想接收或发送一些不符合预期的HTTP请求或响应，这时，我们可以使用servlet过滤器(Filter)，他可以实现：

**在客户端的请求访问后端资源之前，拦截这些请求。**

**在服务器的响应发送回客户端之前，处理这些响应。**



常用的过滤器有：

* 身份验证过滤器（Authentication Filters）。
* 数据压缩过滤器（Data compression Filters）。
* 加密过滤器（Encryption Filters）。
* 触发资源访问事件过滤器。
* 图像转换过滤器（Image Conversion Filters）。
* 日志记录和审核过滤器（Logging and Auditing Filters）。
* MIME-TYPE 链过滤器（MIME-TYPE Chain Filters）。
* 标记化过滤器（Tokenizing Filters）。
* XSL/T 过滤器（XSL/T Filters），转换 XML 内容。

过滤器(Filter)是一个实现了 javax.servlet.Filter 接口的 Java 类java类，在web.xml中声明，当启动web应用程序时，它会为我们在web.xml中声明的每一个过滤器(Filter)创建一个过滤器对象

一个实现了 javax.servlet.Filter 接口的对象应当实现三个方法：

**1.public void doFilter (ServletRequest, ServletResponse, FilterChain)**

该方法完成实际的过滤操作，当客户端请求方法与过滤器设置匹配的URL时，Servlet容器将先调用过滤器的doFilter方法。FilterChain用户访问后续过滤器。这里要注意，一个一次完整的HTTP过程(一个request和一个response)对同一个Filter要访问两次，第一次是request进入服务器，执行FilterChain前面的代码，第二次是response离开服务器，执行FilterChain后面的代码(责任链设计模式)，FilterChain这一行代码起允许此次请求进入的作用，在响应出来之前，代码将阻塞在这一行。

**2.public void init(FilterConfig filterConfig)**

web 应用程序启动时，web 服务器将创建Filter 的实例对象，并调用其init方法，读取web.xml配置，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（filter对象只会创建一次，init方法也只会执行一次）。开发人员通过init方法的参数，可获得代表当前filter配置信息的FilterConfig对象。

**3.public void destroy()**

Servlet容器在销毁过滤器实例前调用该方法，在该方法中释放Servlet过滤器占用的资源。

web.xml中的配置

<filter>

<filter-name>TestFilter</filter-name>

<filter-class>com.runoob.test.myTestFilter</filter-class>

<init-param>

<param-name>Site</param-name>

<param-value> 我是网站名</param-value>

</init-param>

</filter>

可以在init方法中读取<init-param>的name-value键值对，eg：

**public** **class** myTestFilter **implements** Filter {

**public** **void** init(FilterConfig config) **throws** ServletException {

// 获取初始化参数

String site = config.getInitParameter("Site");

// 输出初始化参数

System.*out*.println("网站名称:" + site); // 输出 网站名称:我是网站名

}

**public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** java.io.IOException, ServletException {

HttpSession session = req.getSession();

//登录后才能进入下一步处理，否则直接进入错误提示页面

**if** (session.getAttribute("username") != **null**) {

context.log("身份认证通过，进入下一步处理 ");

chain.doFilter(request, response); //允许请求进入，阻塞doFilter，等待响应出来再向下执行

} **else** {

context.log("身份认证失败，直接返回");

res.sendRedirect("../failure.jsp");

}

}

**public** **void** destroy( ){

/\* 在 Filter 实例被 Web 容器从服务移除之前调用该方法 \*/

}

}

## 2.7 servlet用作异常处理

一个servlet就像mvc中的控制器一样，对应一个url，因此，我们可以定义一个处理Error的servlet，然后在web.xml中配置它，所有出现异常的servlet都会将异常交由专门处理Error的servlet进行处理

eg：

<!-- servlet 定义 -->

<servlet>

<servlet-name>ErrorHandler</servlet-name>

<servlet-class>ErrorHandler</servlet-class>

</servlet>

<!-- servlet 映射 -->

<servlet-mapping>

<servlet-name>ErrorHandler</servlet-name>

<url-pattern>/error</url-pattern>

</servlet-mapping>

<!-- error-code 相关的错误页面 -->

<error-page>

<error-code>404</error-code>//定义状态码，所有404都会重定向到/error

<location>/error</location>

</error-page>

<error-page>

<error-code>403</error-code>

<location>/error </location>

</error-page>

<!-- exception-type 相关的错误页面 -->

<error-page>

<exception-type>

javax.servlet.ServletException//定义异常类型，所有ServletExceptio类型的异常都会重定向到/error

</exception-type >

<location>/error</location>

</error-page>

<error-page>

<exception-type>java.io.IOException</exception-type >

<location>/error</location>

</error-page>

**当然，异常处理的servlet也可以有多个**

## 2.8 cookie

Cookie 是存储在客户端计算机上的文本文件，用于保存客户端的各种跟踪信息，一个cookie其实就是一组键值对，cookie的整个使用流程如下：服务器向浏览器发送一组指定了时效的cookie，浏览器将其保存在本地，在下一次访问服务器时，浏览器会将这些cookie发送给服务器使用

Servlet提供了一个javax.servlet.http.cookie类供我们使用，我们可以通过request. getCookies()方法获取浏览器发送过来的整个cookie数组，也可以通过response.addCookie方法向HTTP响应中添加cookie到浏览器

Servlet Cookie 处理需要对中文进行编码与解码，eg：

java.net.URLEncoder.encode("中文"，"UTF-8");

java.net.URLDecoder.decode("编码后的字符串","UTF-8");

**cookie常用的方法：**

**1.public void setDomain(String pattern) public String getDomain()**

这两个方法用于设置/获取 cookie 适用的域，指定cookie对哪些域名生效，例如 w3cschool.cc。

**2.public void setMaxAge(int expiry)**

该方法设置 cookie 过期的时间（以秒为单位）。如果不这样设置，cookie 只会在当前 session 会话中持续有效。

**3.public int getMaxAge()**

该方法返回 cookie 的最大生存周期（以秒为单位），默认情况下，-1 表示 cookie 将持续下去，直到浏览器关闭。

**4.public String getName()**

该方法返回 cookie 的名称。名称在创建后不能改变。

**5.public void setValue(String newValue) public String getValue()**

这两个方法设置/获取与 cookie 对应的值。

eg：

Cookie cookie=**new** Cookie("mycookie",Integer.*toString*(i));

cookie.setMaxAge(30);

response.addCookie(cookie);

## 2.9 session

cookie是浏览器端缓存，而session是服务器端缓存，session存储在服务器上，用于跟踪客户端信息。servlet提供了**HttpServletRequest.getSession(true)**方法来创建一个session，里面的参数true意为如果此次request的来源———客户端在服务器上已经有session文件了，那么就自动获取session id找到对应的session文件并返回，如果该客户端还没有session文件，则创建一个新的session

session是存放在服务器的内存中的，因此session文件不宜存储过多信息，服务器会定时清除长时间未活动的session，tomcat默认的session失效时间是20分钟

一台服务器上同时会有多个用户的session文件，一个session文件有一个session ID唯一标识，而由于HTTP是无状态协议，因此，我们必须有一个方法来分辨session文件和客户端的对应关系，这里提供了三种方法

1. **cookie**

服务器将session ID通过cookie储存在浏览器中，然后浏览器下次请求服务器时将会带上session id，通过session id找到对应的session文件。弊端是一旦用户禁用cookie，这种机制就失效了

1. **隐藏表单**

<input type="hidden" name="sessionid" value="12345">

可以在页面中嵌入一个隐藏的表单，每次当该表单被提交的时候，session ID都会被存储在get或post数据中。但是这可能会导致一个问题，并不是每次请求数据都会提交表单，比如ajax请求，比如<a>超链接跳转

1. **url重写**

我们可以通过在url后面加上session id的方式来访问session文件，java为我们提供了一个方法response.encodeURL(String url)，该方法会判断浏览器是否禁用cookie(只要发现浏览器发送了任何Cookie过来,就认为当前客户端没有禁用Cookie)，如果没有禁用，会输出原url，如果禁用，则会在url后面加上sessionID，如

http://localhost:8080/xxx/index.jsp;jsessionid=091A82E070579D3FA881FE901E532422?uid=192

jsessionid是tomcat对于sessionid的称呼(其他的servlet容器就不一定叫jsessionid了)

**常用的session方法：**

**1.public Object getAttribute(String name)**

该方法返回在该 session 会话中具有指定名称的对象，如果没有指定名称的对象，则返回 null。

**2.public void removeAttribute(String name)**

该方法将从该 session 会话移除指定名称的对象。

**3.public void setAttribute(String name, Object value)**

该方法使用指定的名称绑定一个对象到该 session 会话。

**4.public Enumeration getAttributeNames()**

该方法返回 String 对象的枚举，String 对象包含所有绑定到该 session 会话的对象的名称

**5.public String getId()**

该方法返回该 session 会话的session ID。

**6.public void invalidate()**

调用该方法将主动销毁session文件，并解除绑定到它上面的任何对象。

## 2.10 文件上传

Servlet可以和HTML表单元素配合来完成中小型文件的上传功能，在处理文件上传的前端页面时，需要注意：

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

**表单 enctype 属性应该设置为 multipart/form-data，这意味着将以二进制形式提交表单**

**表单 action 属性应该设置为在后端服务器上处理文件上传的 Servlet 文件对应的url**

**上传单个文件，应该使用单个带有属性 type="file" 的 <input .../> 标签。如果要允许多个文件上传，请使用包含多个 name 属性值不同的 input 标签。输入标签具有不同的名称属性的值。浏览器会为每个 input 标签关联一个浏览按钮。当然，前端插件也是可以使用的**

eg：

<form method="post" action="/TomcatTest/UploadServlet" enctype="multipart/form-data">

选择一个文件:

<input type="file" name="uploadFile" />

<br/><br/>

<input type="submit" value="上传" />

</form>

<http://www.runoob.com/servlet/servlet-file-uploading.html>