配置文件web.xml:在WEB-INF路径下，在Servlet3开始，WEB-INF路径下的web.xml文件不在是必需的，但通常还是建议保留配置文件。

1)配置JSP

2)配置和管理Servlet

3)配置和管理Listener

4)配置和管理Filter

5)配置标签库

6)配置JSP属性

7)配置和管理JAAS授权认证

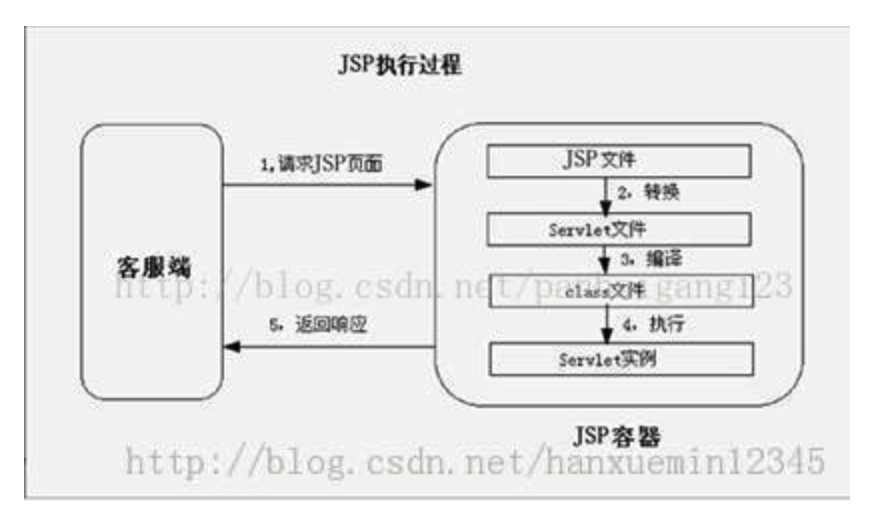
8)配置和管理资源引用。



JSP的本质是Servlet，当用户指定Servlet发送请求时，Servlet利用输出流动态生成HTML页面，包括每一个静态的HTML标签和所有在HTML页面中出现的内容。

JSP页面组成

1. 静态部分：标准的HTML标签，静态的页面内容，这些内容与静态HTMl页面相同。
2. 动态部分：受Java程序控制的内容，这些内容由Java脚本动态生成。



1. JSP文件必须在JSP服务器内运行
2. JSP文件必须生成Servlet才能执行。
3. 每个JSP页面的第一个访问者速度会有点慢，因为必须等待JSP编译成Servlet
4. JSP页面的访问者无需安装任何客户端，甚至不需要可以运行的Java环境，因为JSP页面输送到客户端的标准HTML页面。

JSP注释(JSP注释不会发送到客户端) HTML注释

<%-- 注释内容 --%> <!--注释内容-->

JSP用于声明变量和方法。 <%! 声明部分 %>

JSP不能使用abstract修饰声明部分的方法，因为抽象方法将导致JSP对应的Servlet变成抽象类，从而导致无法实例化。

JSP提供了一种输出表达式值的简单方法（输出表达式后面不能有分号）。 <%=表达式%>代替了out.println输出语句。

JSP的3个编译指令

1. page：该指令是针对当前页面的指令。
2. include：用于指定包含另一个界面
3. taglib：用于定义和访问自定义标签

语法格式

<%@编译指令名 属性名=“属性值” %>

page指令各属性的意义

lauguage:声明当前JSP页面使用的脚本语言的种类，因为页面是JSP页面，该属性的值通常都是java，该属性的默认值也是java，所以通常无需设置。

extends:指定JSP页面编译所产生的Java类所继承的父类，或所实现的接口。

import:用来导入包，下面几个包默认自动导入的，不需要显示导入，默认的包有java.lang.\*;javax.servlet.\*;java.servlet.jsp.\*;java.servlet.http\*;



session:设定这个JSP页面是否需要HTTP Session

buffer:指定缓冲区大小。输出缓冲区的jsp内部对象，out用于缓存JSP页面对浏览器的输出，默认值为8KB，可以设置为none，也可以设置为其他的值，单位为KB。

autoFlush:当缓冲区即将溢出时，是否需要强制输出缓冲区的内容，设置为true时正常输出：如果设置为false，则会在buffer溢出时产生一个异常。

info:设置该JSP程序的信息，也可以看做其说明，可以通过Servlet.getServletInfo()方法获取该值。如果在JSP页面中，可以直接调用getServlet()方法获取该值，因为JSP页面的本质是Servlet。

errorPage:指定错误处理页面。如果本页面产生了异常或者错误，而该jsp页面没有对应的处理代码，则会自动调用该属性所指定的JSP页面。

isErrorPage:设置本JSP页面是否为错误处理程序，如果该页面本身已是错误处理页面，则通常无需指定errorPage属性。

contentType:用于设定生成网页的文件格式和编码字符集，即MIME类型和页面字符集类型，默认的MIME类型是text/html;默认的字符集类型为ISO-8859-1.



pageEncoding:指定生成网页的编码字符集。

include指令：可以将一个外部文件嵌入到当前JSP文件中，同时解析这个页面中的JSP语句。

语法:<%@ include file=”” %> 如果嵌入的文件经常需要改变，建议使用<jsp:include>操作指令。

动作指令与编译指令不同，编译指令是通知Servlet引擎的处理消息，而动作指令只是运行时的动作，编译指令在将JSP编译成Servlet时起作用。而处理指令通常可替换成JSP小脚本，它只是JSP小脚本的标准化写法。

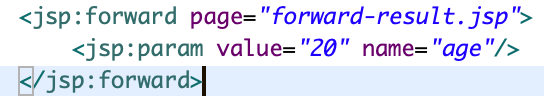
jsp:forward:执行页面转向，将请求的处理转发到下一个页面

语法：<jsp:forward page=”relativeURL|<%=expression%>”>

对于JSP规范1.0；<jsp:forward page=”relativeURL|<%=expression%>” >

<jsp:param.../>

</jsp:forward> 参数可以通过ruquest.getParamter();



jsp:param:用于传递参数，必须与其他支持参数的标签一起使用

jsp:include:用于动态包含一个jsp页面

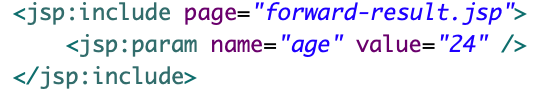
语法：<jsp:include page=”relativeURL|<%=expression%>” flush=”true”>

<jsp:param name=”” value=”” />

</jsp:include >

静态导入：是将导入页面的代码完全融入，两个页面融合成一个整体Servlet；而动态导入则在Servlet中使用include方法来引入被导入页面的内容。

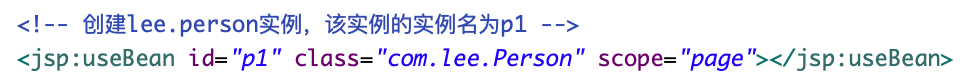
动态导入：可以包含额外的参数



jsp:plugin:用于下载JavaBean或Applet到客户端执行。

jsp:userBean:创建一个JavaBean实例

<jsp:userBean id=”name” class=”classname” scope=”page | request | session | application” /> id:属性是JavaBean的实例名 class属性确定JavaBean的实现类。 scope：属性用于指定JavaBean实例的作用范围。

对应Person p1 = new Person();

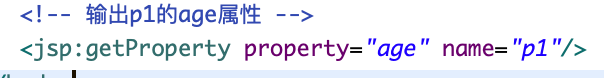
jsp:setProperty:设置JavaBean实例的属性值。

<jsp:setProperty name=”输出JavaBean的实例名” property=”属性确定需要设置的属性名” value=”属性需要设置的属性值” />

对应p1.setAge(24);

jsp:getProperty:输出JavaBean实例的属性值

<jsp:getProperty name=”输出JavaBean的实例名” property=”输出的属性名” />

对应 p1.getAge();

注意：执行forward时，被forward的页面将完全代替原有页面：而执行include时，被include的页面只是插入原有页面，简而言之，forward拿目标页面代替原有页面，而include则拿目标页面插入原有页面。

JSP脚本的内置对象（9个）

application:java.setvlet.ServletContext的实例，常用的方法getAttribute(String name);setAttribute(String name, String value); 和getInitParameter(String name)等

config:java.servlet.ServletConfig的实例，该实例代表该JSP的配置信息，getInitParameter(String name);getInitParameternames()等方法。事实上，JSP页面通常无需配置。

exception:java.lang.Throwable实例，该实例代表其他页面中的异常和错误，只有当页面是错误处理页面，即编译指令page的isErrorPage属性为true时，该对象才可以使用，常用的方法getMessage()和printStackTrace()等。

out:javax.servlet.jsp.JspWriter的实例，该实例代表JSP页面的输出流，用于输出内容，形成HTML页面。

page:代表该页面本身，通常没有太大用处，也就是Servlet中的this，其类型就是生成Servlet类，能用page的地方就可以用this。

pageContext:javax.servlet.jsp.PageContext的实例，该对象代表该JSP页面上下文，使用该对象可以访问页面中的共享数据。常用的方法有getServletContext()和getServletConfig();

request:javax.servlet.http.HttpServletRequest的实例，该对象封装了一次请求，客户端的请求参数都被封装在该对象里。这是一个常用的对象，获取客户端请求参数必须使用该对象。常用的方法有getParameter(String name);getParameterValues(String name);setAttribute(String name, Object value);getAttribute(String name);setCharacterEncoding(String env)等

response:javax.servlet.http.HttpServletResponse的实例，代表服务器对客户端的响应，通常很少使用该对象直接响应，而是使用out对象，除非需要生成非字符响应，而response对象常用于重定向，常用的方法有getOutputStream()，sendRedirect(java.lang.String.location)等。

session:javax.servlet.http.HttpSession的实例，该对象代表一次会话，当客户端浏览器与站点建立连接时，会话开始；当客户端关闭浏览器时，会话结束，常用的方法getAttribute(String name);setAttribute(String name, Object value);

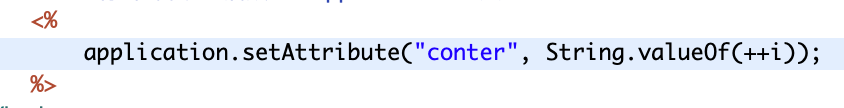
application对象：

浏览器完成三件事：1）向远程服务器发送请求 2）读取远程服务器返回的字符串 3）负责根据字符串数值渲染出一个丰富多彩的页面。

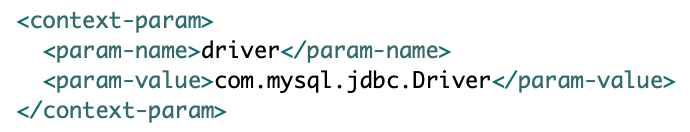
每次客户端请求，web服务器大致需要完成如下几个步骤：

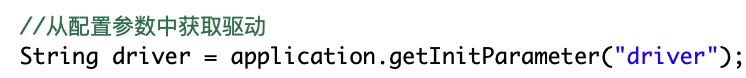
1. 启动单独线程
2. 使用I/O流读取用户请求的二进制数据
3. 从请求数据中解析参数
4. 处理用户请求
5. 生成响应数据
6. 使用I/O流向客户端发送请求数据

application:对于整个Web应用有效，一旦JSP，Servlet将数据放入application中，该数据将可以被应用下其他所有的JSP，Servlet访问（在整个Web应用的多个JSP，Servlet之间共享数据。访问Web应用的配置参数。）



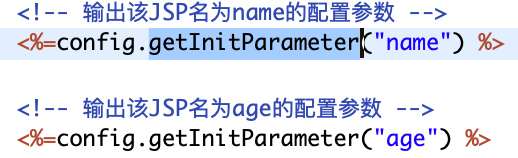






可以将一些配置信息放在web.xml文件中配置，避免使用硬编码方式写在代码中。

config:对象代表当前JSP配置信息，但JSP页面通常无需配置，因此也就不存在配置信息，因此JSP页面比较少用该对象。但是Servlet中则用处相对较大，因为Servlet需要在web.xml文件中进行配置，可以指定配置参数。





exception对象：是Throwable的实例，代表JSP脚本中产生的错误和异常，是JSP页面异常机制的一部分。如果该页面的page指令指定了errorPage属性，则将请求forward到errorPage属性指定的页面，否则使用系统页面来输出异常信息。

只有page指令的isErrorPage属性设置为true，才可以访问exception内置对象。

out对象代表一个页面输出流，通常用于在页面上输出变量值及常量。

pageContext对象：这个对象代表页面上下文，该对象主要用于JSP之间的共享数据。使用pageContext可以访问page，request，session，application范围变量，提供以下方法来访问范围变量

getAttribute(String name);取得page范围内的name属性。

getAttribute(String name, int scope);取得指定范围内的name属性，其中scope可以是如下四个值，

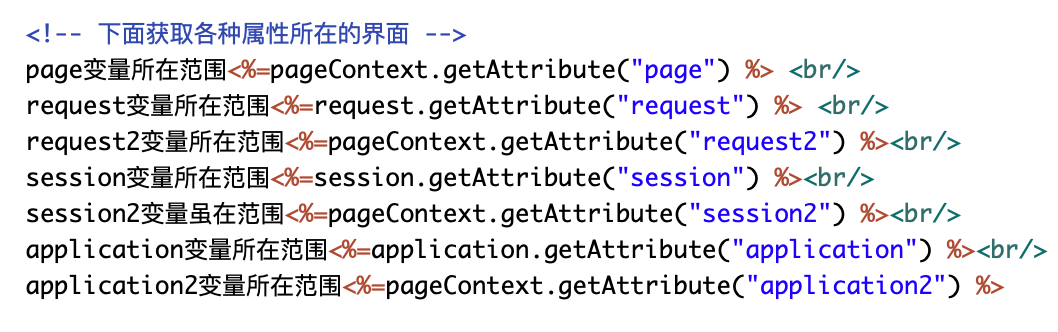
PageContext.PAGE\_SCOPE:对应page范围

PageContext.REQUEST\_SCOPE:对应于request范围

PageContext.SESSION\_SCOPE:对应于session范围

PageContext.APPLICATION\_SCOPE;对应于application范围





pageContext还可以获得其它内置对象

ServletRequest getRequest():获取request对象。

ServletResponse getResponse():获取response对象

ServletConfig getServletConfig():获取config对象

ServletContext getServletContext:获取application对象

HttpSession getSession():获取session对象。

request对象是JSP中重要的对象，每个request对象封装着一次用户请求，并且所有的请求参数都被封装在request对象中。

1）获取请求头/请求参数：请求参数发送给服务器，服务器端负责解析请求头/请求参数的就是JSP或Servlet，而JSP和Servlet取得请求参数的途径就是request。提供如下几个请求参数

String getParameter(String name);//获得name请求参数的值

Map getParameterMap();//获取所有的请求参数名和参数值所组成的Map对象。

Enumeration getParameterNames();//获取所有请求参数名所组成的Enumeration对象。

String[] getParameterValues(Strng name);//name请求参数的值，当该请求参数有多个值时，该方法将返回多个值所组成的数组。

String getHeader(String name);//更具指定请求头的值

java.util.Enumeration<String> getHeaderNames();//获取所有请求头名称

java.util.Enumeration<String> getHeaders();//获取指定请求头的多个值。

int getIntHeader(String name);//获取指定请求头的值，并将该值转为整数值。

请求参数通常分为两中情况：

1. GET方法的请求：直接在浏览器地址栏输入访问地址所发送的请求或提交表单发送请求时，该表单对象的form元素没有设置method属性，或设置method属性为get，这几种请求都是GET方式的请求，GET方式的请求会将请求参数的名和值转换成字符串，并附加在原URL之后，因此可以在地址栏中看到请求参数名和值。且GET请求传送的数据量较小，一般不能大于2kb。

写法：URL和参数之间以”?”分隔，而多个参数以“&”请求参数分隔。

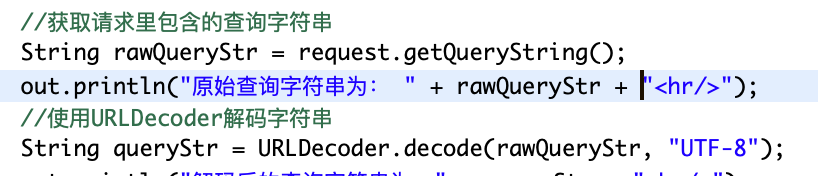
1. POST方式的请求：这种方式通常使用提交表单的方式来发送，且需要设置 form元素的method属性为post。POST方式传送的数据量较大，通常认为POST请求参数的大小不受限制，但往往取决于服务器的限制，POST请求传输的数据量总比GET传输的数据量大。而且POST方式发送的请求参数以及对应的值在HTML HEADER中传输，用户不能在地址栏中看到参数值，安全性相对较高。

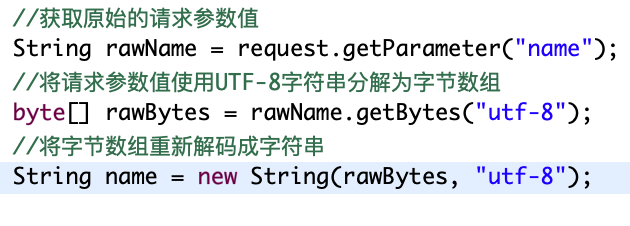
并不是每个表单域都会生成请求参数，而是有name属性的表单域才生成请求参数。

1）每个有name属性的表单域对应一个请求参数

1. 如果有多个表单域有相同的name属性，则多个表单域只生成一个请求参数，只是该参数有多个值。
2. 表单域的name属性指定请求参数名，value指定请求参数值。
3. 如果某个表单域设置了disabled=”disabled”属性，则该表单域不在生成请求参数。

中文乱码问题的解决方法

1）

2)

request:仅对本次请求有效，一旦JSP，Servlet将数据放入request中，该数据将可以被该次请求的其他JSP，Servlet访问。

操作request范围的属性

1）setAttribute(String attName, Object attValue);将attName设置成request范围的属性。

2）Object getAttribute(String name);获取request范围的属性。

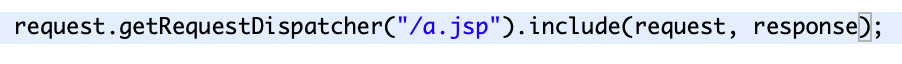
request还有一个功能就是执行forward和include也就代替JSP所提供的forward和include动作指令，前面需要forward时都是通过JSP提供的动作指令进行的。实际上request对象也可以执行forward。

HttpServletRequest类提供了一个getRequestDispatcher(String path)方法，其中path就是希望forward或者include的目标路径，该方法返回RequestDispatcher,该对象提供了如下两个方法。注意（path字符串必须以斜线开头）

forward(ServletRequest request, ServletResponse response);执行forward



include(ServletRequest request, ServletResponse response);执行include

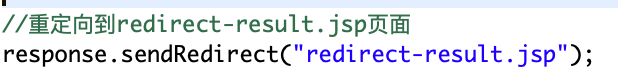


2.6.7 response对象

response代表服务器对客户端的响应。大部分时候，程序无须使用response来响应客户端请求，因为有个更简单的响应对象-out,它代表页面输出流，直接使用out生成响应给简单。

但out是JspWriter的实例，JspWriter是Writer的子类，Writer是字符流，无法输出非字符内容，假如需要在JSP页面中动态生成一幅位图，或者输出一个PDF文档，使用out作为作为响应对象将无法完成，此时必须使用response作为响应输出，

重定向是response的另外一个用处，与forward不同的是，重定向会丢失所有的请求参数和request范围的属性，因为重定向将生成第二次请求，与前一次请求不在同一个request范围内，所以发送一次请求的请求参数和request范围的属性全部丢失。HttpServletResponse提供了一个sendRedirect(String path)方法，该方法用于重定向到path资源，即重新向path资源发送请求。



|  |  |
| --- | --- |
| 转发（forward） | 重定向（redirect） |
| 执行forward后依然是上一次请求 | 执行redirect后生成了第二次请求 |
| forward的目标页面可以访问原请求的参数，因为依然是同一次请求，所有原请求的请求参数，request范围的属性全部存在。 | redirect的目标页面不能访问原请求的请求参数，因为是第二次请求了，所有的原请求的请求参数，request范围的属性全部丢失 |
| 地址栏里请求的URL不会改变 | 地址栏改为重定向的目标URL，相当于在浏览器地址栏里面输出新的URL后按回车键 |

增加Cookie:通常用于网站记录客户的某些信息（使用response内置对象完成的）

void addCookie(Cookie cookie);//增加Cookie

创建Cookie实例，Cookie的构造器为Cookie（String name, String value）;

设置Cookie的声明周期，即该Cookie在多长时间内有效

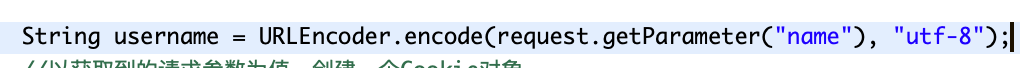
向客户端写Cookie

使用Cookie对象必须设置其生存期限，否则Cookie将会随浏览器的关闭而自动消失。





编码和解码的用法

request:

respon:

session：仅对一次会话有效，一旦JSP，Servlet将数据放入到Session中，该数据将可以被本次会话的其他所有的JSP，Servlet访问。这个对象代表一次用户会话，一次用户会话的含义是：从客户端浏览器连接服务，到客户端浏览器与服务器断开为止。这个过程就是一次会话。

session对象是HttpSession的实例，HttpSession有如下两个常用的方法

setAttribute(String attName, Object attvalue);//设置session范围内attribute属性值为attvalue。

getAttribute(String attName);//返回session范围内attribute属性的值。

page：仅对当前页面有效，一旦JSP，Servlet将数据放入page中，该数据只可以被当前页面的JSP脚本，声明部分访问。

2.7 Servlet介绍

JSP的本质就是Servlet，Servlet通常被称为服务器端小程序，是运行在服务器端的程序，用于处理及响应客户端的请求。

Servlet提供不同的方法用于响应客户端请求

1）doGet:用于响应客户端的GET请求。

2）doPost：用于响应客户端的POST请求。

3）doPut:用于响应客户端的PUT请求。

4）doDelete:用于响应客户端的DELETE请求。

事实上，客户端的请求通常只有GET和POST两种，Servlet为了响应这两种请求，必须重写doGet()和doPost()两个方法。

init(ServletConfig config):创建Servlet实例时，调用该方法的初始化Servlet资源。

destroy():销毁Servlet实例时，自动调用该方法的回收资源。

不用为Servlet类编写构造器，如果需要对Servlet执行初始化操作，应将初始化操作方法Servlet的init()方法中定义。如果重写了init(ServletConfig config)方法，则应该重写该方法的第一行调用super.init(config)。该方法将调用HttpServlet的init方法。

Servlet和JSP的区别在于

1)Servlet中没有内置对象，原来JSP中的内置对象都必须有程序显式创建。

2)对于静态的HTML标签，Servlet都必须使用页面输出流逐行输出。

Servlet3开始，配置Servlet有两种方式

1）在Servlet3类中使用@WebServlet注解进行配置。

|  |  |  |
| --- | --- | --- |
| 属性 | 是否必须 | 说明 |
| asyncSupported | 否 | 指定Servlet是否支持异步操作模式。 |
| displayName | 否 | 指定该Servlet的显示名。 |
| initParams | 否 | 用于为该Servlet配置参数 |
| loadOnStartup | 否 | 用于将该Servlet配置成load-startup的Servlet |
| name | 否 | 指定该Servlet的名称 |
| urlPatterns/value | 否 | 这两个属性的作用完全相同，都指定该Servlet处理的URL |

2）通过在web.xml文件中进行配置。

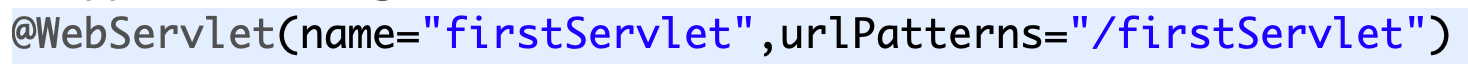
不要在web.xml文件的跟元素（<web-app.../>）中指定metadata-complete=”true”.

不要在web.xml文件中配置该Servlet。

如果打算使用web.xml文件来配置该Servlet，则需要配置如下两个部分

1）配置Servlet的名字：对应web.xml文件中的<servlet/>元素

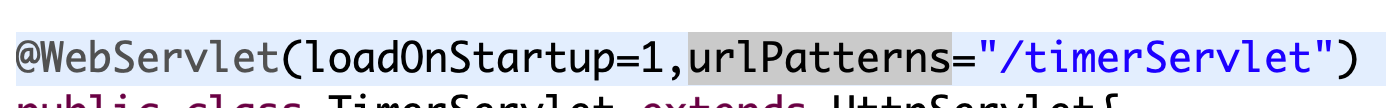
2）配置Servlet的url：对应的web.xml文件中的<servlet-mapping/>元素，这一步是可选的。但如果没有为Servlet配置URL，则该Servlet不能响应用户请求。





JSP/Servlet的生命周期：其实例的创建及销毁都是由web容器进行控制的

1. 客户端的第一次请求某个Servlet时，系统创建该Servlet的实例；大部分的Servlet都是这种Servlet
2. Web应用启动时创建Servlet实例，即load-on-startup Servlet.



每个Servlet的运行都遵循如下生命周期

1)创建Servlet实例

2)Web容器调用Servlet的init方法，对Servlet进行初始化。

3)Servlet初始化后，将一直存在于容器中，用于响应客户端请求，如果客户端发送GET请求，容器调用Servlet的doGet方法处理并响应请求；如果客户端发送POST请求，容器调用Servlet的doPost方法处理并响应请求，或者统一使用service()方法处理来响应用户请求。

1. Web容器决定销毁Servlet时，先调用Servlet时destroy方法，通常在关闭Web应用之时销毁Servlet。

创建Servlet实例有两个时机：用户请求之时或应用启动之时，应用启动时就创建Servlet，通常是用于某些后台服务的Servlet，或者需要拦截很多请求的Servlet；这种Servlet通常作为应用的基础Servlet使用，提供重要的后台服务。

配置load-on-startup的Servlet有两种方式。

1. 在web.xml文件中通过<servlet.../>元素<load-on-startup.../>子元素进行配置。
2. 通过@WebServlet注解的loadOnStartup属性指定。一定要加上urlPatterns/value否则的话这个注解不生效。

<load-on-startup.../>元素或loadOnStartup属性都只接收一个整型值，这个整型值越小，Servlet就越优先实例化。

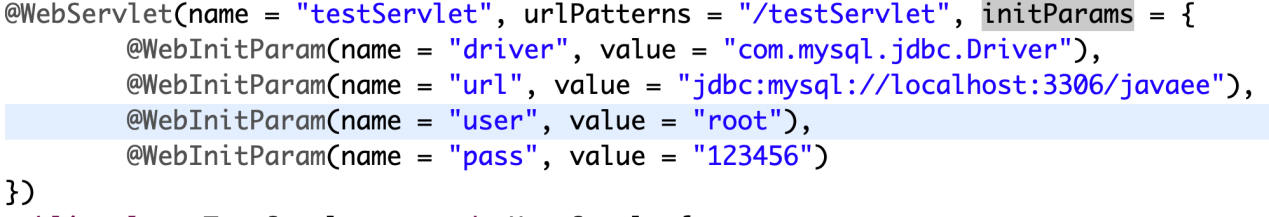
配置Servlet的参数

1）通过@WebService的initParams属性来指定

2）通过在web.xml文件的<servlet.../>元素添加<init-param.../>子元素来指定。

3）访问Servlet配置参数通过ServletConfig对象完成，ServletConfig提供如下方法：（对应于JSP的内置对象config）

java.lang.String.getInitParameter(String name):用于获取初始化参数。



Servlet作为表现层有如下三个劣势

1）开发效率低：所有的HTML标签都需要使用页面输出流完成

2）不利于团队协作开发，美工人员无法参与Servlet界面的开发

3）程序可维护性差：在标准的MVC模式中，Servlet仅作为控制器使用，JSP仅作为表现层（View）

4）负责收集用户请求参数

5）将应用的处理结果，状态数据呈现给用户。

MVC

M:Model,即模型，对应JavaBean。

V:View，即视图，对应JSP页面

C:Controller,即控制器，对应Servlet。

JSP2的自定义标签

自定义标签类应该继承一个父类：javax.servlet.jsp.tagext.SimpleTagSupport.

1）如果标签类包含属性，每个属性都应该有getter和setter方法。

1. 重写doTag()方法，这个方法负责生成页面内容。



建立TLD文件

TLD是Tag Library Definition的缩写，即标签库定义，文件的后缀是tld，每个TLD文件对应一个标签库，一个标签库可包含多个标签，TLD文件也称为标签库定义文件。自定义标签库，将该文件复制到WEB应用的WEB-INF路径，或者WEB—INF的任意子路径下即可。

标签库定义文件的跟元素是taglib,它可以包含多个tag子元素，每个tag子元素都定义一个标签。

taglib下有如下三个子元素

1. tlib-viersion:指定该标签库实现的版本，这是一个作为标识的内部版本号，对程序没有太大的作用。
2. short-name:该标签库的默认短名，该名称通常也没有太大的用处。
3. uri：这个属性非常重要，它指定该标签库的URI，相当于指定该标签库的唯一标识，如上面粗体字代码所示。JSP页面中使用标签库时就是根据URI属性来定位标签库的。

taglib元素可以包含多个tag元素，每个tag元素定义一个标签，tag元素下允许出现如下常用子元素。

1. name：指定标签名称，这个子元素也很重要，JSP页面中就是根据该名称来使用此标签。
2. tag-class：指定标签的处理类，毋庸置疑，这个子元素非常重要，它指定类标签由哪个标签处理类来处理。
3. body-content：这个子元素也很重要，它指定标签体内容，该子元素的值可以是如下几个。

tagdependent:指定标签处理类自己负责处理标签体。

empty:指定该标签只能作为空标签使用。

scriptless:指定该标签的标签体可以是静态HTML元素，表达式语言，但不允许出现JSP脚本。

JSP：指定该标签的标签体可以使用JSP脚本。

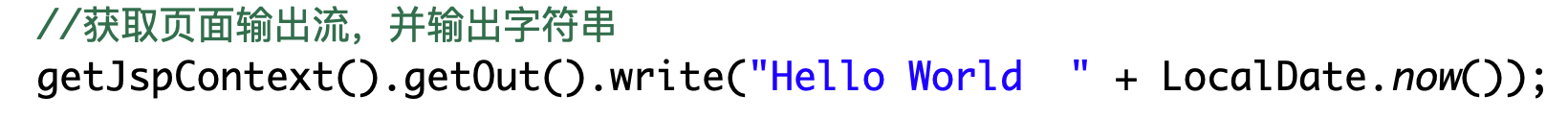
4）dynamic-attributes:指定该标签是否支持动态属性，只有当定义动态属性标签时才需要该子元素。

tag的语法格式如下：

<%@ taglib uri=”tagliburi” prefix=”tagPrefix” %>

1. uri指定标签库的uri，这个URI可以确定一个标签库。
2. prefix属性指定标签库前缀，即所有使用该前缀的标签将由此标签库处理。

带属性的标签：必须为每个属性提供对应的setter和getter方法



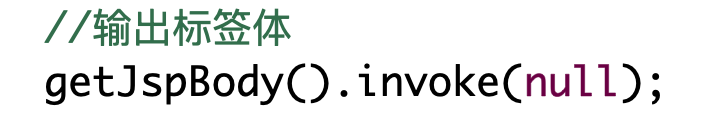
需要<tag.../>元素增加<attribute.../>子元素，每个attribute子元素定义一个标签属性，<attribute.../>子元素通常还需要指定如下几个子元素

1）name：设置属性名，子元素的值是字符串内容。

2）required：设置该属性是否为必须属性，该子元素的值是true或false。

3）fragment：设置该属性是否支持JSP脚本，表达式等动态内容，子元素的值是true或false。

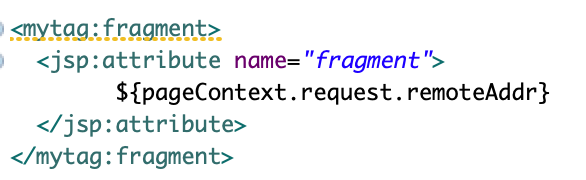
带标签体的标签：可以在标签内嵌入其他内容（包括静态的HTML内容和动态JSP内容），通常用于完成一些逻辑运算。



标签处理类中定义类型为JspFragment属性，该属性代表了页面片段

使用标签库时，通过<jsp:attribute.../>动作指令为标签的属性指定值。





动态变迁属性

1）标签处理类还需要实现DynamicAttributes接口

2）配置标签时通过<dynamic-attributes.../>子元素指定该标签支持动态属性。

实现DynaAttributesTag接口，就是动态属性标签处理类必须实现的接口，实现该接口必须重写setDynaAttribute()方法，该方法用于为该标签处理类动态地添加属性名和属性值。



Filter介绍

1）在HttpServletRequest到达Servlet之前，拦截客户的HttpServletRequest.

2) 根据需要检查HttpServletRequest,也可以修改HttpServletRequest头和数据

3）在HttpServletResponse到达客户端之前，拦截HttpServletResponse.

4）根据需要检查HttpServletResponse,也可以修改HttpServletResponse头和数据

Filter有如下几个种类

1）用户授权的Filter：Filter负责检查用户请求，根据请求过滤用户非法请求。

2）日志Filter：详细记录某些特殊的用户请求

3）负责解码的Filter：包括对非标准编码的请求解码

4）能改变XML内容的XML Filter等。

5）Filter可负责拦截多个请求或响应：一个请求或响应也可被多个Filter拦截。

创建一个Filter只需要两个步骤

1）创建Filter处理类

2）在web.xml文件中配置Filter或用注解配置Filter。

创建Filter类：必须实现javax.servlet.Filter接口

1）void init(FilterConfig config);用于完成Filter的初始化

2）void destroy();用于Filter销毁前，完成某些资源的回收。

3）void doFilter(ServletRequest request, ServletResponse response, FilterChain chain):实现过滤功能，该方法就是对每个请求及响应增加的额外处理。

配置Filter：

1）配置Filter名

2）配置Filter拦截URL模式

与Servlet的区别在于：Servlet通常只配置一个URL，而Filter可以同时拦截多个请求的URL，因此，在配置Filter的URL模式时通常会使用模式字符串，使得Filter可以拦截多个请求。与配置Servlet相似的是，配置Filter同样有两种方式。

1）在Filter类中通过注解进行配置

1. 在web.xml文件中通常配置文件进行配置。

@WebFilter修饰一个Filter类，用于对Filter进行配置。常用属性：

|  |  |  |
| --- | --- | --- |
| 属性 | 是否必须 | 说明 |
| asyncSupported | 否 | 指定该Filter是否支持异步操作模式。 |
| dispatcherTypes | 否 | 指定该Filter仅对哪种dispatcher模式的请求进行过滤，该属性支持ASYNC,ERROR,FORWARD,INCLUDE,REQUEST这5个值的任意组合。默认值为同时过滤5中模式的请求 |
| displayName | 否 | 指定该Filter显示名 |
| filterName |  | 指定该Filter的名称 |
| initParams | 否 | 用于为该Filter配置参数 |
| servletNames | 否 | 该属性可指定多个Servlet的名称，用于指定该Filter仅对这几个Servlet进行过滤。 |
| urlPatterns/value | 否 | 这两个属性的作用完全相同，都指定该Filter所拦截的URL。 |

web.xml的配置： /\*：代表拦截所有用户请求

Listener介绍

当web应用在Web容器中运行时，Web应用内部会不断地发生各种事件，如Web应用被启动，Web应用被停止，用户session开始，用户session结束，用户请求到达等，实际上，Servlet API提供了大量监听器来监听Web应用的内部事件

使用Listener的步骤

1）定义Listener实现类

2）通过注解或web.xml文件中配置Listener

常用的Web事件监听器接口

1）ServletContextListener:用于监听web应用的启动和关闭

2）ServletContextAttributeListener:用于监听ServletContext范围（application）内属性的改变

3）ServletRequestListener:用户监听用户请求

1. ServeltRequestAttributeListener:用于监听ServletRequest范围（request）内属性的变化
2. HttpSessionListener:用于监听用户session的开始和结束。
3. HttpSessionAttributeListener:用于监听HttpSession的开始和结束。

配置Listener

1. 使用@WebListener修饰Listener实现类即可
2. 在web.xml文档中使用<listener.../>元素进行配置。

使用@WebListener时通常无须指定任何属性，只要使用该注解修饰Listener实现类即可向web应用注册该监听器

在web.xml中使用<listener../>元素进行配置时只要配置如下子元素即可

1）listener-class:指定Listener实现类。

若将ServletContextListener配置在Web容器中，且Web容器（Servlet 2.3以上）支持Listener，则该ServletContextListener将可以监听Web应用的启动，关闭

<listener>

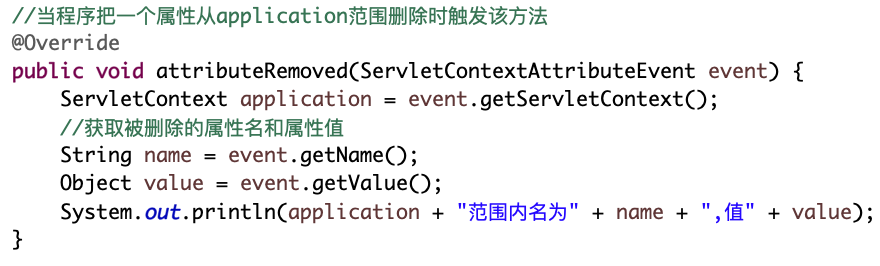
<!-- 指定Listener的实现类 -->

<listener-class>com.lee.实现类</listener-class>

</listener>

ServletContextAttributeListener:用于监听ServletContext(application)范围内属性的变化，实现该连接的监听器需要实现下面三个方法







ServletRequestListener:用于监听用户请求的到达，实现该接口的监听器需要实现下面方法

1）requestDestroyed(ServletRequestEvent arg0) :用户请求结束，被销毁时触发该方法。

2）requestInitialized(ServletRequestEvent arg0):用户请求到达，被初始化时触发该方法。

HttpSessionListener用于监听用户session的创建和销毁，实现该接口的监听器需要实现如下方法。

1）sessionCreated(HttpSessionEvent se):用户与服务器的会话开始，创建时触发该方法。

2）sessionDestroyed(HttpSessionEvent se):用户与服务器的会话断开，销毁时触发该方法。

通过HttpServletRequest的做法可以更精确地监控在线用户的状态

1）定义一个ServletRequestListener，这个监听负责监听每个用户请求，当用户请求到达时，系统将用户请求的sessionId,用户名，用户ip，正在访问的资源啊，访问时间记录下来。

2）启动一个后台线程，这条后台线程每隔一段时间检查上面的每条记录，如果某条在线记录的访问时间与当前时间相差超过指定值，将这条在线记录删除即可，这条后台线程随着web的启动而启动，可考虑使用ServletContextListener来完成。

JSP2特性

1）直接配置JSP属性

2）表达式语言

3）简化的自定义标签API

4）Tag文件语法

配置JSP属性：JSP属性定义使用<jsp-property-group/>元素配置；

1）是否允许使用表达式语言，使用<el-ignored/>元素确定，默认值为false，即允许使用表达式语言。

2）是否允许使用JSP小脚本，使用<scripting-invalid/>元素确定，默认值为false，即允许使用JSP小脚本。

1. 声明JSP页面的编码：使用<page-encoding/>元素确定，配置该元素后，可以代替每个页面里page指令contentType属性的charset部分。
2. 使用隐式包含：使用<include-prelude/>和<include-coda/>元素确定，可以代替在每个页面里使用include编译指令来包含其它页面。

表达式语法格式：${expression}

如果需要在支持表达式语言的页面中正常输出“$”符号，则在“$”符号前加转义字符”/”,否则系统以为”$”是表达式语言的特殊标记

表达式语言的内置对象

1）pageContext:代表该页面的pageContext对象，与JSP的pageContext内置对象相同。

2）pageScope:用于获取page范围属性值。

3）requestScope:用于获取request范围的属性值。

4）sessionScope:用于获取session范围的属性值。${sessionScope['user'] }

5）applicationScpoe:用于获取application范围的属性值。

6）param:用于获取请求的参数。${param.name }

7）paramValues:用于获取请求的参数值，与param的区别在于，该对象用于获取属性值为数组的属性值。

8）header:用于获取请求头的请求。${header.host } ${header['host'] }

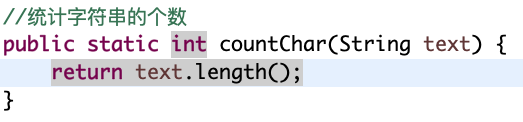
9）headerValues:用于获取请求头的属性值，与header的区别在于，该对象用于获取属性值为数组的属性值。

10）initParam:用于获取请求Web应用的初始化参数。

11）cookie:用于指定的Cookie值。${cookie["name"].value }

自定义函数

1）开发函数处理类，函数处理类就是普通类，这个普通类中包含若干个静态方法，每个静态方法都可定义成一个函数，实际上这个步骤也是可省略的——完全可以直接使用JDK或其他项目提供的类，只要这个类包含静态方法即可。



1. 使用标签库定义函数：定义函数的方法与定义标签的方法大致相似。这<taglib.../>元素下增加<tag.../>元素用于自定义标签:增加<function.../>元素则用于定义自定义函数。function中的参数

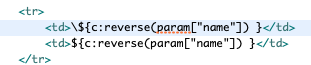
1）name:指定自定义函数的函数名。

2）function-class:指定自定义函数的处理类。

3）function-signature:指定自定义函数对应的方法。



3)在JSP页面的EL中使用函数：一样需要先导入标签库，然后在使用函数。



Tag File是自定义标签的简化用法，使用Tag File可以无须定义标签处理类。

1. 建立Tag文件，在JSP所支持Tag File规范下，Tag File代理了标签处理类，它的格式类似JSP文件

标签指令：1）taglib：作用与JSP文件中的taglib指令效果相同，用于导入其他标签库

2）include：作用与JSP文件中include指令效果相同，用于导入其他JSP或静态页面。

3）tag：作用类似于JSP文件中的page指令，有pageEncoding，body-content等属性，用于设置页面编码等属性。

4）attribute：用于设置自定义标签的属性，类似于自定义标签处理类中的标签属性。

5）variable：用于设置自定义标签的变量，这些变量将传给JSP页面使用。

Servlet 3 新特性

@WebServlet:用于修饰一个Servlet类，用于部署Servlet类。

@WebInitParam:用于与@WebServlet或@WebFilter一起使用，为Servlet，Filter配置参数。

@WebListener:用于修饰Listener类，用于部署Listener类。

@WebFilter:用于修饰Filter类，用于部署Filter类。

@MultipartConfig:用于修饰Servlet，指定该Servlet将会负责处理myltipart/form-data类型的请求（主要用于文件的上传）

@ServletSecurity:这是一个与JAAS有关的注解，修饰Servlet指定该Servlet的安全与授权控制。

@HttpConstraint:用于与@ServletSecurity一起使用，用于指定该Servlet的安全与授权控制。

@HttpMethodConstraint:用于与@ServletSecurity一起使用，用于指定该Servlet的安全与授权控制。

Servlet3的异步处理是通过AsyncContext类来处理的，Servlet可通过ServletRequest的如下两个方法开启异步调用，创建AsyncContext对象。

1）AsyncContext startAsync();

2）AsyncContext startAsync(ServletRequest, ServletResponse)

AsyncContext对象代表异步处理的上下文，它提供了一些工具，可完成设置异步调用的超时时长，dispatch用于请求，启动后台线程，获取request，response对象等功能。

为Servlet开启异步调用两种方式。

1）在@WebServlet指定asyncSupported=true;

2）在web.xml文件的<servlet.../>元素中增加<async-supported.../>子元素。

<servlet>

<servlet-name>async</servlet-name>

<servlet-class>类路径</servlet-class>

<!-- 开启异步调用支持 -->

<async-supported>true</async-supported>

</servlet>

异步监听器需要实现AsyncListener接口，实现该接口的监听器需要实现如下4个方法。

1）onStartAsync(AsyncEvent event):当异步调用开始时触发该方法。

2）onComplete(AsyncEvent event):当异步调用完成时触发该方法。

3）onError(AsyncEvent event):当异步调用出错时触发该方法。

4）onTimeout(AsyncEvent event):当异步调用超类式触发该方法。

注册监听器

AsyncContext actx = request.startAsync();

//为该异步调用注册监听器

actx.addListener(new 实现AsyncListener的类)

HttpServletReqeust增加了对文件上传的支持。

1）Part getPart(String name):根据名称来获取文件上传域。

2）Collection<Part> getParts():获取所有的文件上传域。

part，每个part对象对应一个文件上传域，该对象提供了大量写方法来访问上传文件的文件类型，大小，输出流等，并提供了一个write(String file)方法将上传文件写入服务器磁盘中。上传文件表单域设置enctype属性。

enctype属性：

1）application/x-www-form-urlencoded:这是默认的编码方法，它只处理表单域里的value属性值，采用这种编码方式的表单会将表单域的值处理成URL编码方法。

2）multipart/form-data:这种编码方式会以二进制流的方式来处理表单数据，这种编码方式会把文件域指定文件的内容也封装到请求参数里。

3）text/plain:这种编码方式当表单的action属性为mailto:URL的形式比较方便，这种方式主要适用于直接通过表单发送邮件的方式。

上传语法：

1）jsp：

2）注解：@MultipartConfig修饰， 配置：可以在<serlvet.../>元素中添加<multipart-config.../>元素来达到相同效果。

3）Servlet写法：

1:获取该文件类型

Part part = request.getPart(“file”);

2:获取文件的类型

part.getContentType();

3:获取文件的大小

part.getSize();

4:获取该文件上传域的Header Name

Collection<String> headerNames = part.getHeaderNames();

5:获取包含原始文件的字符串

String fileNameInfo = part.getHeader(“content-disposition”);

6:提取上传文件的原始文件名

String fileName = fileNameInfo.subString(fileNameInfo.indexOf("filename=\"") + 10, fileNameInfo.length() - 1);

7:将上传的文件写入服务器

part.write(getServletContext().getRealPath(“/uploadFiles”) + “/” + fileName);

ServletContext:允许通过编程的方式动态注册Servlet，Filter。并允许动态设置Web应用参数

1. 多个重载的addServlet()方法，动态地注册Servlet。
2. 多个重载的addFilter()方法，动态地注册Filter。
3. 多个重载的addListener()方法，动态地注册Listener.
4. setInitParameter(String name, String value)方法，为Web应用设置初始化参数。

Servlet 3.1新增的非阻塞式IO

Servlet底层的IO是通过如下两个IO流来支持的

1）ServletInputStream:Servlet用于读取数据的输入流。

2）ServletOutputStream:Servlet用于输出数据的输出流。

在Servlet3.1，ServletInputStream新增一个setReadListener(ReaderListener readListener)方法，该方法允许以非阻塞IO读取数据，实现ReaderListener监听器需要实现如下三个方法。 1）onAllDataRead():当所有数据读取完成时触发该方法

2）onDataAvailable():当有数据可用时激发该方法。

3）onError(Throwable t):读取数据出现错误激发该方法。

类似的ServletOutputStream:也提供了setWriterListener(WriteListener writeListener)方法，通过这种方法，以非阻塞IO进行输出。

在Servlet中使用非阻塞IO非常简单，步骤如下。

1）调用ServletRequest的startAsync()方法开启异步模式。

AsyncContext context = request.startAsync();

2）通过ServletRequest获取ServletInputStream，并为ServletInputStream设置监听器(Readlistener的实现类)。

ServletInputStream input = request.getInputStream();

3）实现ReadListener接口来实现监听器，在该监听器的方法中以非阻塞方法读取数据。

Tomcat 8.5 的WebSocket支持:只要服务器端有了新数据，服务器端就可以直接将数据推送给浏览器，让浏览器显示最新的状态。

1）使用注解方式开发，@ServerEndpoint修饰的java类即可作为WebSocket服务器。

2）继承Endpoint基类实现WebSocket服务端。

开发@ServerEndpoint修饰的Java类之后，，该类还可以定义如下方法。

1）被@OnOpen修饰的方法：当客户端与该WebSocket服务端建立连接时激发该方法。

2）被@OnClose修饰的方法，当客户端与该WebSocket服务端断开连接时激发该方法。

3）被@OnMessage修饰的方法，当WebSocket服务端收到客户端消息时激发该方法。

4）被@OnError修饰的方法，当客户端与该WebSocket服务端连接出现错误时激发该方法。

WebSocket是HTML5规范新增的类：



1）onopen：当WebSocket客户端与服务端建立连接时自动激发该事件处理函数。

2）onclose：当WebSocket客户端与服务端关闭连接时自动激发该事件处理函数。

3）onmessage:当WebSocket客户端收到服务端消息时自动激发该事件处理函数。