# Servlet容器

Servlet容器是作为一个独立发展的标准化产品，目前种类很多，我们常用的Servlet容器有：Tomcat和Jetty，它们各有各自的优缺点。

Servlet容器负责Servlet和客户端的通信以及调用Servlet方法。



## Servlet和Servlet容器的关系

Servlet和Servlet容器是通过标准化接口来相互协作。

Tomcat容器分为四个等级，真正管理Servlet的容器是Context容器，一个Context对应一个Web工程。

## Servlet容器启动

在Tomcat7中增加了一个启动类org.apache.catalina.startup.Tomcat。创建一个实例对象并调用start方法就可以很容易地启动Tomcat。

添加一个Web应用时将会创建一个StandardContext容器，并且给这个Context容器设置必要的参数，url和path分别代表这个应用在Tomcat中的访问路径和这个应用实际的物理路径，这两个参数与Tomcat配置中的两个参数是一致的。

ContextConfig将会负责整个Web应用配置的解析工作。

## Web应用的初始化

Web应用的初始化工作是在ContextConfig的configureStart方法中实现的，应用的初始化主要是解析web.xml文件，这个文件描述了一个web应用的关键信息，也是一个Web应用的人口。

在Tomcat中Servlet被包装成Context容器中的StandardWrapper。

# JSP

## 内置对象

**jsp内置对象：**

request：请求对象

response:响应对象

pageContext:页面上下文

session:回话对象

application:应用程序对象

out:输出对象

config:配置对象

page：页面对象

exception:异常对象

从本质上讲，jsp内置对象是特定的Java类所产生的，在服务器自动生成的。

|  |  |  |
| --- | --- | --- |
| 对象名 | 类型 | 作用域 |
| request | Javax.servlet.ServletRequest | Request |
| response |  | Page |
| pageContext |  | Page |
| session |  | Session |
| application | Java.servlet.servletContext | Application |
| out | Javax.servlet.jsp.JspWrite | Page |
| config | Javax.servlet.ServletConfig | Page |
| page | Java.lang.Object | Page |
| exception | Java.lang.Throwable | Page |

## 指示对象

JSP指示对象的主要目的，在于指示容器将JSP转译为Servlet源码时，一些必须遵守的信息。

语法：<%@ 指示类型 [属性=”值”]\* %>

在JSP中有三种常用的指示类型：page,include与taglib。

* page:指示类型告知容器如何转译目前JSP网页。

**属性**：import：告知容器转译JSP时，必须在源代码中包括的import陈述。

contentType:告知容器转译JSP时，必须使用HttpServeltRequest的setContentType().

pageEncoding：告知容器这个JSP网页的文字编码，以及内容类型附加的charset设置。

info：设置目前JSP页面的基本信息，这个信息最后会转换为Servlet程序中使用getServeltInfo()所取得。

autoFlush：用于设置输出串流是否要自动清除，默认为true，如果设置为false，而缓冲区满了却还没有调用flush()将数据送出到客户端，则会产生异常。

buffer：用于设置至客户端的输出串流缓冲区大小，设置时必须指定单位，buffer=20kb

errorPage：用于设置当JSP执行错误而发生异常时，该转发哪一个页面处理异常。

extends：用于设置当JSP转译呈Servlet时，该继承哪一个类。

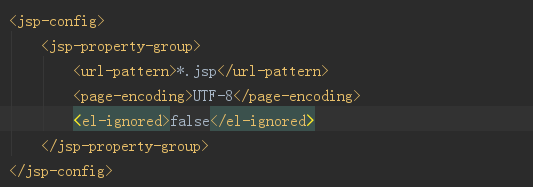
isErrorPage：设置JSP是否为处理异常的页面，这个属相要与errorPage配合使用

language：指定容器使用哪种语言的语法转译JSP网页，实际上目前只支持Java

session：设置是否在转译后Servlet源代码中具有创建HttpSession对象的语句，默认为true。

isELIgnored：设置JSP网页中是否忽略表达式语言，默认为false

isThreadSafe：告知容器编写JSP时是否注意到线程安全问题，默认为true，如果设置为false，则转译之后的Servlet会实现SingleThreadModel接口，每次请求都会将创建一个Servlet实例来服务请求，虽然可以避免线程安全问题，但是这会引起性能问题，极度不建议设置为false。



* include:指示类型告知容器将别的jsp页面包括进来进行转译。
* taglib:告知容器如何转译这个页面中的标签库

## EL表达式

在EL中，可以使用EL隐式对象指定范围来存取属性。若不指定属性的存在范围，则默认是以page,request,session,application的顺序来寻找EL中所指定的属性。

如果EL访问的对象是个数组对象，则可以使用[]运算符来指定索引以存取数组中的元素。不仅数组对象可以在[]中指定索引来访问元素，List类型的对象也可以使用[]运算符指定索引来进行访问元素。

|  |
| --- |
| **<%** String[] names={"li","sai","hao"};  request.setAttribute("array",names); **%>** <p>**${**array[0]**}**</p><p>**${**array[1]**}**</p><p>**${**array[2]**}**</p> |

使用点运算符（.）的场合也可以使用[]运算符：

* 如果使用点（.）运算符，则左边可以是JavaBean或Map。
* 如果使用[]运算符，则左边可以是JavaBean，Map或List对象。

建议：如果左边是Map对象时，建议使用[]运算符，因为如果设置Map时的键名称有空白或点字符时，这是可以正确取得值的。

|  |
| --- |
| **<%** Map<String ,String> map = new HashMap<>();  map.put("user name","hao");  map.put("local.role","admin");  request.setAttribute("login",map); **%>** <p>**${**login["user name"]**}**</p><p>**${**login["local.role"]**}**</p> |

### EL隐式对象

* pageContext隐式对象：对应于PageContext类型，PageContext本身就是一个JavaBean，只要是getXxx()方法，都可以使用${pageContext.xxx}来取得。
* 请求参数相关隐式对象：与请求参数相关的EL隐式对象有：param和paramValues。

例如：${param.user}相当于<%= request.getParameter(“user”)%>

${paramValues.user}相当于<%= request.getParameterValues().user%>

* 属相范围相关隐式对象：与属性范围相关的EL隐式对象有：

|  |  |
| --- | --- |
| 隐式对象 | 对应的JSP隐式对象 |
| pageScope | pageContext |
| requestScope | request |
| sessionScope | session |
| applicationScope | application |

EL隐式对象仅仅代表作用范围。

* 标头相关隐式对象：如果要取得用户请求头的表头数据，可以使用header或headerValues隐式对象。
* Cookie隐式对象：cookie隐式对象可以用来取得用户的Cookie设置值，如果在Cookie中设置了username属性，、则可以使用${cookie.username}来取值。
* 初始参数隐式对象：initParam可以用来取得web.xml中设置的ServletContext初始参数，也就是在<context-param>中设置的初始参数。

### EL运算符

使用EL可以直接进行一些算术上的运算，逻辑运算与关系运算。

EL运算符的执行优先级与java运算符对应，也可以使用 括号()来自行决定先后顺序。

### 自定义EL函数

第一步：编写功能类，并且功能类中的方法必须是公共的（publish）和静态的（static）

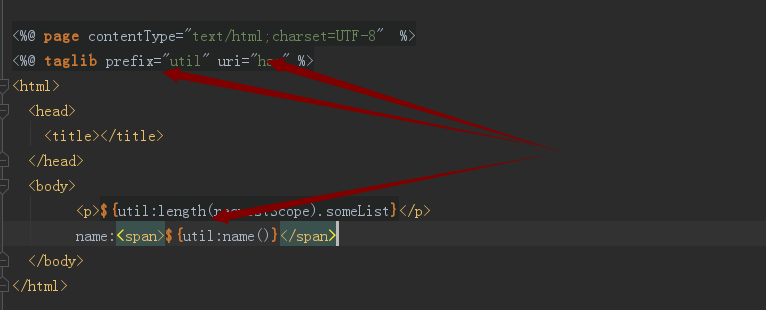
|  |
| --- |
| public class Util {  public static int length(Collection collection)  {  return collection.size() ;  }  public static String getName()  {  return "admin" ;  } } |

第二步：编写标签程序描述文件，即tld文件，这个文件是XML文件，但后缀是以.tld结尾，

该配置文件可以放在WEB-INF文件下，这样容器会自动找到这个TLD文件并载入，。也可以放在jar文件中。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <taglib xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee [url]http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd[/url]"  version="2.0">  <tlib-version>1.0</tlib-version>  <short-name>openhome</short-name>  <uri>hao</uri><!--设置url对应名称-->  <function>  <name>length</name><!--自定义的EL函数名称-->  <function-class>code.lsh.util.Util</function-class><!--对应的类-->  <function-signature>int length(java.util.Collection)</function-signature>  </function>  <function>  <name>name</name>  <function-class>code.lsh.util.Util</function-class>  <function-signature>java.lang.String getName()</function-signature>  </function> </taglib> |

第三步：页面引入：



# JSTL

JSTL不仅提供了条件判断的逻辑标签，还提供了对应的JSP标准标签的扩展以及更多的功能标签。JSTL提供的标签库分为五个大类：

* 核心标签库：提供条件判断，属性访问，URL处理以及错误处理等标签
* I18n兼容格式标签库：提供数字，日期等的格式化功能，以及区域信（Locale）息，编码处理等国际化功能的标签。
* SQL标签库：提供基本的数据库查询，更新，设置数据源等功能的标签。
* XML标签库：提供XML解析，流程控制，转换等功能的标签。
* 函数标签库：提供常用子串处理的自定义EL函数标签库。

JSTL是另一个标准规范，并非在JSP的规范中，所以必须另外下载：

下载地址：

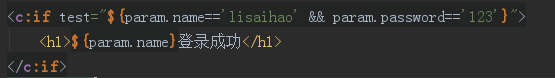
## 核心标准库

JSTL核心标准库主要包括：流程处理标签可处理页面呈现逻辑，错误处理标签可捕获异常，重定向标签，属相处理标签，输出处理标签，URL处理标签等。

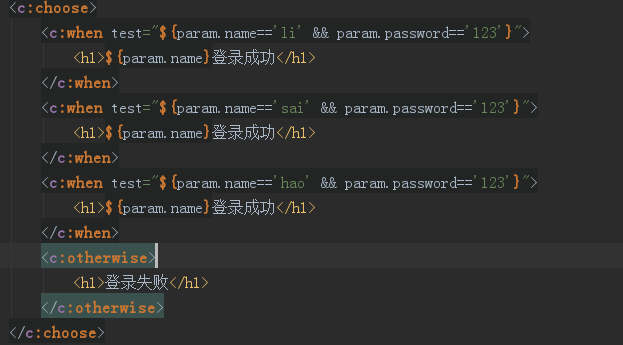


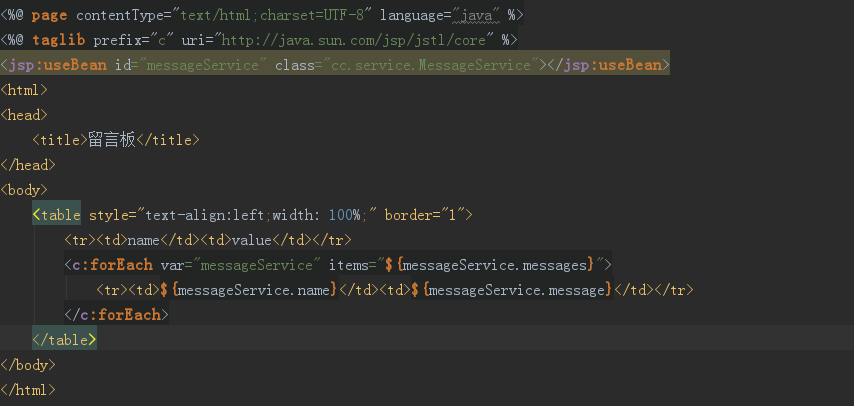
前置名称设置了这个标签库再次JSP网页中的名称空间，以避免与其他标签库的标签名发生冲突，惯例使用”c”作为前置名称，url引用则告诉容器，如何引用JSTL标签库实现。

### 流程处理标签



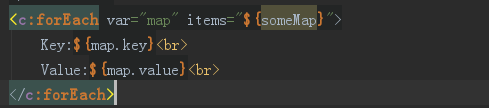
**说明：**<c:if>标签只有在test的结果为true时显示Body内容，不过并没有与之对应的<c:else>标签。

**说明：** <c:when>以及<c:otherwise>必须放在<c:chose>中，当<c:when>的test的结果为true时，会输出<c:when>的Body内容，而不会执行<c:otherwise>中的内容，<c:chose>中可以有多个<c:when>标签，此时会从上到下进行测试，如果有个<c:when>的test结果为true就输出其Body中的内容，之后的<c:when>就不会做测试，如果所有的<c:when>测试都不成立，则会输出<c:otherwise>的内容。

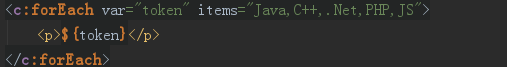


**说明：**<c:forEach var=”” item=””> items属性可以是数组，Collection，Iteration，Enumeration，Map和字符串，每次会依次从item指定的对象中取出一个元素，并指定给var属性设置的变量，之后就可以在<c:forEach>标签的Body中使用var属性所设置的变量来取得该元素。

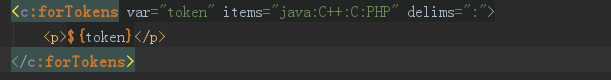
如果items是Map，则设置给var的对象会是Map.Entry，这个对象有getKey()和getValue()方法：



如果items是字符串，则必须是个以逗号区隔的值：

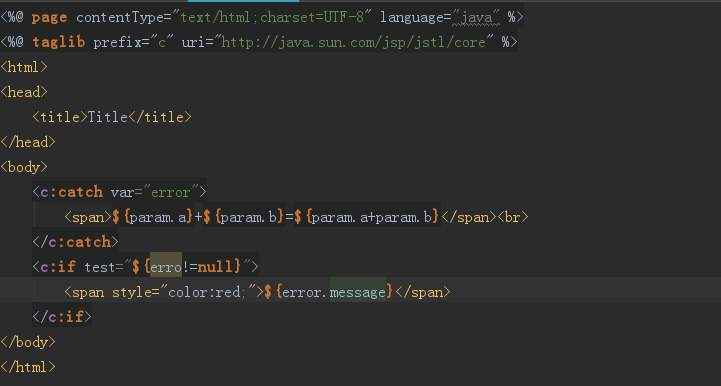


如果希望自行指定切割依据，则可以使用<c:forTokens>:



### 错误处理标签

使用JSTL中的<c:catch>标签进行一场捕获：



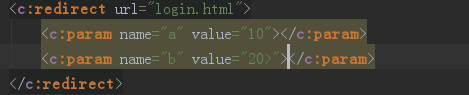
如果发生异常的网页直接捕获异常对象，可以使用<c:catch>将可能产生异常的网页段落包起来。如果异常发生，则这个异常对象会设置给var属性所指定的名称。

### 网页导入标签



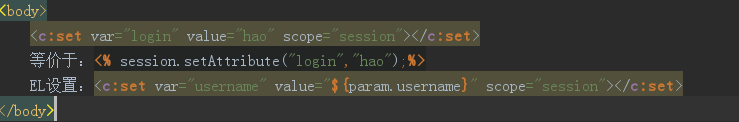
使用<c:import>导入网页时可以带有参数，并且可以导入目前Web应用程序中的网页之外，其中charEncoding属性指定要导入网页的编码，防止产生乱码情况。

### 重定向标签



### 属性处理标签

如果想要在page，request，session，application等范围内设置一个属性，或者想要设置Map对象的键和值，我们可以使用<c:sets>标签



如果<c:set>中没有设置scope属性，则会以page,request,session,application的范围寻找属性名称，如果在某个范围找到属性名称，则在该范围设置属性。如果在所有范围内都没有找到属性名称，则会在page范围中新增属性。

如果要移除某个属性，则可以使用<c:remove>标签：

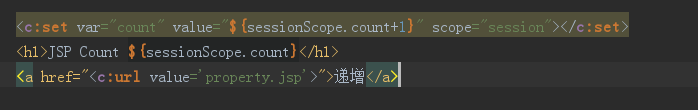


<c:set>也可以设置JavaBean的属性或Map对象的键/值，要设置JavaBean或Map对象，必须使用target属性进行设置：



### URL处理标签

当用户关闭Cookie功能时，我们可以使用JSTL中的<c:url>标签，该标签会自动用SessionID作URL重写。



### 输出标签

<c:out value=””>会自动将角括号，单引号，双引号等字符用替代字符取代。

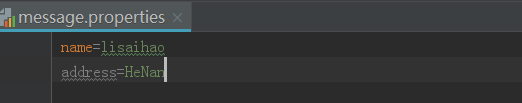
## I18N兼容格式标签库

应用程序可以根据不同国家的用户，呈现不同的语言，数字格式，日期格式等

**国际化：**地区（Locale）信息，资源包（Resource bundle）和基础名称（Base name）

**地区信息**：代表了特定的地理，政治或文化区，地区信息可由一个**语言编码**与可选的**地区编码**来指定。其中语言编码时ISO-639定义，有两个小写字母代码，如：“zh”表示中国，“ca”表示加拿大。地区编码有两个大写字母表示，定义在ISO-3166,例如：“UK”表示英国，“CN”表示中国。

**资源包**：包括了特定地区的相关信息，代表同一组信息但不同地区的各个资源包共享相同的**基础名称**。



例如：ResourceBundle对象就是JVM中资源包的代表对象，使用ResourceBundle的getBundle()时指定的名称，就是在指定基础名称。若只给出一个参数，则用默认的Locale（有Locale.getDefault()取得的值）取得.properties文件，如果默认的Locale代表zh\_CN，则ResourceBundle的getBundle(“message”)会尝试取得message\_zh\_CN.properties文件中的信息，若找不到，再尝试找message.properties文件中的信息。

对于中文的.properties文件，我们可以使用jdk自带的native2ascii工具进行编译



同样也可以将Unicode编码表示的.properties转回中文：



### 信息标签

要使用JSTL的I18N兼容格式标签库，必须在JSP网页上使用taglib指示元素定义前置和uri引用：



最基本：<fmt:bundle>,<fmt:message>

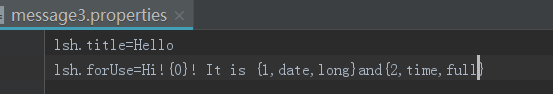
定义message.properties配置文件到/WEB-INF/classes中。



使用：<fmt:setBundle>标签设置basename属性，设置的作用域默认是整个页面都有作用，如果有额外的<fmt:bundle>设置，则会以<fmt:bundle>为主：



如果信息有些部分必须动态决定，则可以使用占位符先代替：



如果想设置占位符的真正内容，则使用<fmt:param>标签

### 地区标签

JSTL的I18N兼容型标签，会尝试从属性范围中取得javax.servlet.jsp.jstl.fmt.LocalizationContext

对象，借以决定资源包与地区信息。

决定信息文件顺序：（1）使用指定的Locale对象取得信息文件。

（2）根据浏览器Accept-Language标头指定的偏好地区（Prefered locale）顺序。

（3）根据后备地区（fallback locale）信息取得信息文件

（4）使用基础名称取得信息文件

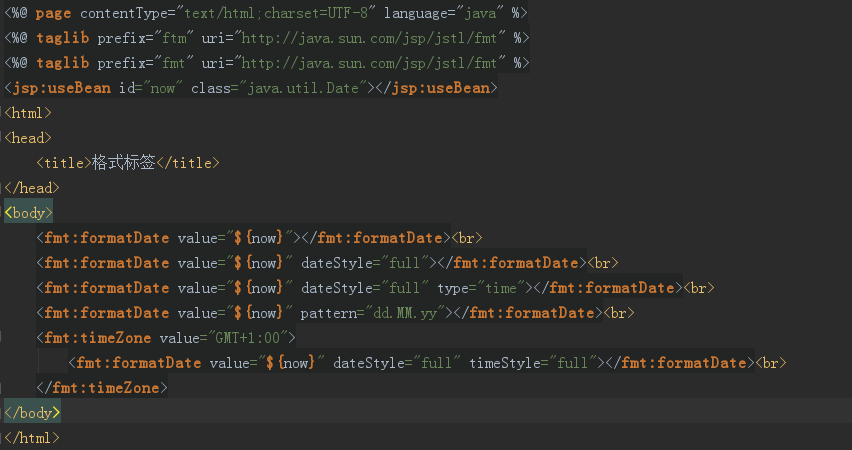
<fmt:message>标签bundle属性，可以用来指定LocalizationContext对象，可以在创建LocalizationContext对象时指定ResourceBundle与Locale对象。

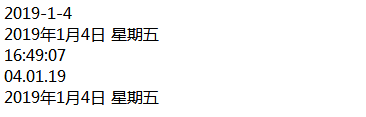
使用<fmt:setLocale>共享Locale信息，value属性上指定地区信息。

### 格式标签

JSTL的格式标签可以对数字，日期与时间，搭配地区设置或指定的格式来进行格式化，也可以进行数字，日期与时间的解析。

**日期格式化输出：**





<fmt:formatDate>默认用来格式化日期，可根据不同的地区设置来呈现不同的格式。

dateStyle：指定日期的详细程度，可设置的值有：default,short,medium,long,full。

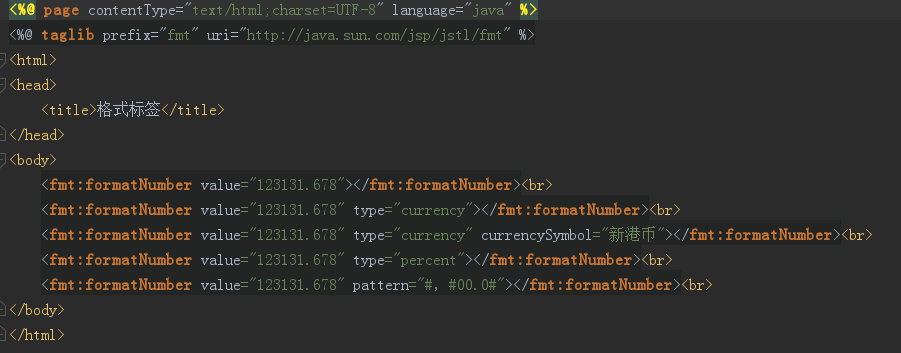
type：显示时间，可设置的值有：time ,both，默认是date。

timeStyle：指定时间的详细程度，可设置的值有：default,short,medium,long,full。

pattern：自定义格式，格式的指定方式与java.text.SimpleDateFormat的指定方式相同。

<fmt:timeZone>：指定时区，可使用字符串或java.util.TimeZone对象指定。

**数字格式化输出：**



<fmt:formatNumber>格式化输出数字，可根据不同的地区呈现不同的格式。

type：可设置的值：number（默认），currency，percent。

currency：将数字按货币进行格式化。

percent：将数字按百分比进行格式化。

currencySymbol：指定货币符号

pattern：将数字一指定的格式进行格式化，与java.text.DecimalFormat设置相同。

<fmt:parseDate>和<fmt:parseNumber>：在value属相指定要解析的数值，pattern指定解析的格式。

## SQL标签库

## XML标签库

## 函数标签库



JSTL提供的字符串处理相关函数：

* 改变字符串大小写：toLowerCase，toUpperCase
* 取得子字符串：substring，substringAfter，substringBefore
* 剪切字符串前后空白：trim
* 字符串取代：replace
* 检查是否包括子字符串：startsWith，endsWith，contains,containsIgnoreCase

# Servlet

Servlet功能：（1）创建并放回基于客户端请求的动态HTML页面

（2）创建可嵌入到现有的HTML页面中的部分HTML页面

（3）与其他服务器资源进行通信（如：数据库或java的应用程序）。

Servlet容器装载Servlet：

（1）Servlet容器启动时装载某些Servlet。

（2）Servlet启动后，客户端第一次向Servlet发出请求。

（3）Servlet文件被更改后，重新加载Servlet。

Servlet装载后，Servlet容器创建一个Servlet实例并调用Servlet的init()方法进行初始化，在Servlet整个生命周期内，init()方法之后调用一次。

Servlet初始化：（1）

## Cookie

Cookie是浏览器存储信息的一种方式，服务器可以响应浏览器set-cookie标头，浏览器收到这个标头与数值后，会将它以文件的形式存储到计算机上，这个文件就是Cookie

## Session

当一个Session开始时，Servlet容器将会创建一个HttpSession对象，在这个对象中可以存放客户的信息。

Servlet容器会为HttpSession分配一个唯一标识符，称为SessionID。Servlet容器会把SessionID作为Cookie保存到用户客户端中。这中Cookie存放于浏览器进程中，随着浏览器进程的结束而消失。

当浏览器请求应用程序时，会将Cookie中存放的SessionID一并发送给应用程序，Web容器会根据SessionID来找到对应的HttpSession对象，这样就可以取得各浏览器个别的会话数据。

默认关闭浏览器会马上失效的是浏览器上的Cookie，不是HttpSession。因为Cookie失效了，就无法通过Cookie来发送SessionID，所以尝试getSession()时，容器会产生新的HttpSession。

每次客户端发出HTTP请求时，Servlet可以从HttpServletRequest对象中读取SessionID，然后根据SessionID找到对应的HttpSession对象，从而获取用户的状态信息。

HttpSession并非线程安全，所以必须注意属性设定时共享存取问题。

Javax.servlet.SessionCookieConfig可以设定存储SessionID的Cookie相关信息，注意设定SessionCookieConfig必须在ServletContext初始化之前，所以在web.xml设置。

### HttpSession与URL重写

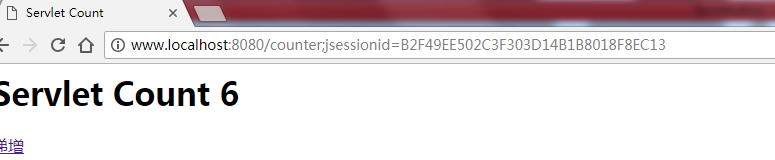
HttpSession默认使用Cookie存储SessionID，如果用户禁用Cookie，就无法使用Cookie在浏览器中存储SessionID，在这种情况下，如果我们扔打算使用HttpSession来进行会话管理，那么可以搭配URL重写，向浏览器响应一段超链接，超链接URL后附加SessionID，当用户点击超链接，将SessionID以Get请求发送给Web应用程序。

URL重写可以使用HttpServeltResponse的encodeURL()方法来协助所需要的URL重写，若能从HTTP请求中取得所需的SessionID的Cookie，encodeURL()会将传入的URL原封不动的输出，否则encodeURL()会自动产生带有SessionID的URL重写。

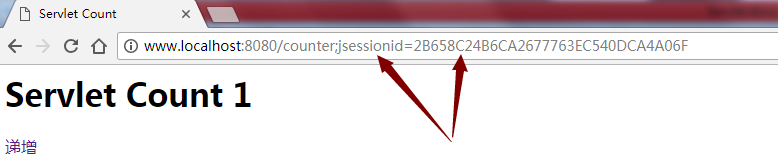
例如：

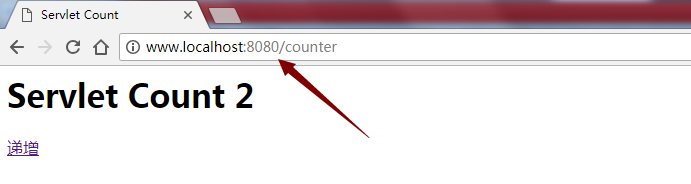
|  |
| --- |
| @WebServlet("/counter") public class Counter extends HttpServlet {  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException  {  resp.setContentType("text/html; charset=UTF-8");  PrintWriter out = resp.getWriter();  int count = 0;  HttpSession session = req.getSession();  if (session.getAttribute("count") != null) {  Integer c = (Integer) session.getAttribute("count");  count = c + 1;  }  session.setAttribute("count", count);  out.println("<html>");  out.println("<head>");  out.println("<title>Servlet Count</title>");  out.println("</head>");  out.println("<body>");  out.println("<h1>Servlet Count " + count + "</h1>");  out.println("<a href='" + resp.encodeURL("counter") + "'>递增</a>");  out.println("</body>");  out.println("</html>");  out.close();  } } |

**浏览器禁用Cookie运行结果：**



**不禁用Cookie情况下：**





从结果可以看出，在浏览器第一次请求时，容器并不知道浏览器是否禁用Cookie，所以容器的做法是Cookie与URL重写的方式，即使用encodeURL(url)都会显示SessionID的URL.当再次请求时，如果 浏览器没有禁用Cookie，则容器可以从Cookie取得SessionID，则encodeURL()就不会编码URL，否则SessionID显示在URL。

HttpServletResponse的encodeRedirectURL()方法，则可以在要求浏览器重定向时，在URL上显示SessionID.

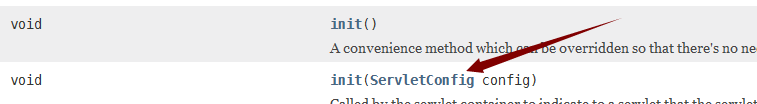


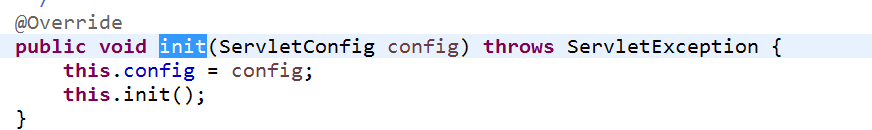
## ServletConfig

在web容器启动后，会读取Servlet设置信息，将Servlet类加载并实例化，并为每个Servlet设置信息产生一个ServletConfig对象，而后调用Servlet接口的init()方法，并将产生的ServletConfig对象当作参数出入。

ServletConfig即每个Servlet设置的代表对象，容器会为每个Servlet设置信息产生一个Servlet及ServletConfig实例。GenericServlet同时实现了Servlet和ServletConfig.

GenericServlet的主要目的，就是初始Servlet调用init()方法传入的ServletConfig封装起来。





GenericServlet在实现Servlet的init方法时，会调用另一个无参的init方法。

如果想要使用ServletConfig来作一些事情，则不能在构造器中定义，因为实例化Servlet时，容器还没有调用init方法传入ServletConfig，所以不会有ServletConfig实例。

## ServletContext

ServletContext接口定义了运行Servlet的应用程序环境 的一些行为与观点，可以使用ServletContext实现对象来取得所请求资源的URL，设置与存储属性，应用程序初始化参数，甚至动态设置Servlet实例。

在Web应用程序被加载到Web容器后，容器会生成一个ServletContext对象作为整个应用程序的代表，并设置给ServletConfig。

1. getRequestDispatcher()

用来取得RequestDispatcher实例，使用时路径的指定必须以”/”作为开头，这个斜杠代表应用程序环境根目录。

以”/”开头有时称为环境相对(Context-relative)路径，没有以”/”开头则称为请求相对(Request-relative)路径。实际上HttpServeltRequest的getRequestDispather()方法的实现时，若是环境相对路径，则直接委托给ServletContext的getRequestDispather()；若是请求相对路径，则转换成环境相对路径，再委托给ServletContext的getRequestDispather()。

1. getResourceAsStream()

ServletContext中getResourceAsStream()方法是获取Web应用程序中某个文件内容，使用时指定路径必须以”/”作为开头，拜师相对于应用程序环境根目录，或者相对是/WEB-INF/lib中jar文件里META-INF/resources的路径。

绝对路径是指文件在服务器上的真实路径，相对路径不是相对于Web应用程序根目录，而是相对于启动Web容器时的命令执行目录。

# 过滤器（Filter）

介于Servlet之前，可拦截过滤浏览器对Servlet的请求，也可以改变Servlet对浏览器的响应。

如：性能评测，用户校验，字符替换，编码设置等要求，都可以使用过滤器实现。

我们可以视需求抽换过滤器或调整过滤器的顺序，也可以根据对不同的URL应用设置不同

的过滤器。

## 请求封装

## 响应封装

# 异步处理

Web容器为每一请求分配一个线程，默认情况下，响应完成前，该线程占用的资源都不会被释放，若有些请求需要长时间处理，就会长时间占用线程所需资源，若这类请求很多，许多线程资源都被长时间占用，会对系统的性能造成负担。

## AsyncContext

该类代表一个可执行异步操作的上下文。

我们可以通过调用ServletRequest.startAsync()或者[ServletRequest.startAsync(ServletRequest, ServletResponse)](mk:@MSITStore:D:\jar\javaee-web.CHM::/javax/servlet/ServletRequest.html#startAsync-javax.servlet.ServletRequest-javax.servlet.ServletResponse-)方法创建并初始化一个AsyncContext对象，如果我们重复调用该方法，将会返回同一个AsyncContext对象，并适当的初始化。

创建AsyncContext对象之后，此次请求的响应会被延后，并释放容器分配的线程。此次对客户端的响应将暂存至调用AsyncContext的complete()或dispatch()方法为止，前者表示响应完成，后者表示将调派指定的URL进行响应。

当异步操作过时后，容器必须执行以下操作：

1. 调用onTimeout()方法，并且所有的AsyncListener实例都注册在初始化异步操作的ServletRequest上。
2. 如果没有一个监听者调用complete()方法或dispatch()方法，将会转发到一个错误页面和

HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR的状态码。

1. 如果匹配到错误页面，或错误页面没有调用complete()或dispatch()方法，则complete()方法将会调用。

**我们可以设置@WebServlet中的asyncSupported=true属性，告知容器此Servlet支持异步处理，或者配置web.xml中<async-supported>true</async-supported>。**

**如果Servlet将会异步处理，若其前端有过滤器，则过滤器也需要标识其支持异步处理，我们可以设置@WebFilter中asyncSupported=true属性，或者web.xml配置文件中:**

**<filter><async-supported>true</async-supported></filter>**

### 模拟服务器推播

# JDBC

JDBC全程Java DataBase Connectiveity，是Java数据库连接的标准规范。它定义一组标准类和接口。应用程序需要连接数据库时就调用这组标准API，而标准API中的接口会由数据库厂商实现，通常称为JDBC驱动程序。

## 连接数据库和操作数据库

### 连接数据库

第一步：注册Driver实现对象。

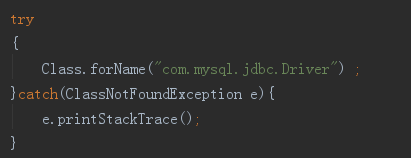
实现Driver接口的对象时JDBC进行数据库访问的起点，com.mysql.jdbc.Driver类实现了java.sql.Driver接口，管理Driver实现对象的类时java.sql.DriverManager。

使用JDBC时，加载.class文件有四种方式：

1. 使用Class.forName()。
2. 自行创建Driver接口实现类的实例
3. 启动JVN时指定jdbc.drivers属性

Java. –D jdbc.drivers=com.mysql.jdbc.Driver; 主类

1. 设置JAR中，/services/java.sql.Driver文件。



第二步：取得Connection对象。

Connection接口的实现对象，是数据库连接代表对象。

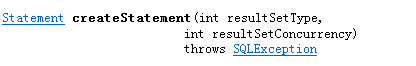


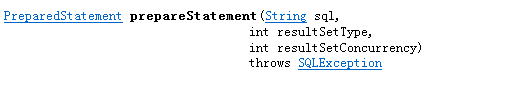
第三步：关闭Connection对象。

### 操作数据库

#### 使用ResultSet

在使用Connection的createStatement()或prepareStatement()方法创建Statement或PrepareStatement实例时，可以指定结果集类型和并行方式：





结果集类型：ResultSet.TYPE.FORWARD\_ONLY（默认）：ResultSet只能前进光标

ResultSet.TYPE.SCROLL\_INSENSITIVE：ResultSet不仅可以前后移动光标，而且不会反应数据库中的数据修改。

ResultSet.TYPE.SCROLL\_SENSITIVE：ResultSet不仅可以前后移动光标，而且会反应数据库中的数据修改。

更新设置：ResultSet.CONCUR\_READ\_ONLY（默认）：只能用ResultSet进行数据读取，无法进行更新。

ResultSet.CONCUR\_UPDATABLE：可以使用ResultSet进行数据更新。

如果要使用ResultSet进行数据修改，则有些条件限制：

* 必须选择单一表格
* 必须选择主键
* 必须选择所有NOT NULL的值

#### 批次更新

## DataSource

## 常用接口

### Metadata

Metadate即“关于数据的数据”，在JDBC中，可以通过Connection的getMetaData()方法获取DatabaseMetadata对象，通过这个对象可以获取数据库整体信息。

### ResultSetMetadata

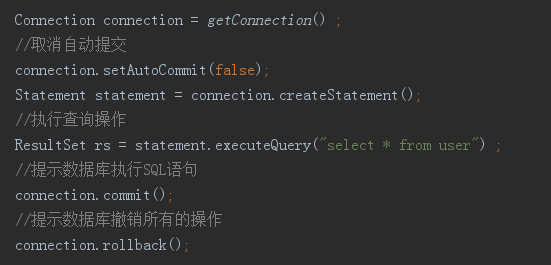
### RowSet

RowSet接口用以代表数据的列集合，这里的数据并不一定是数据库中的数据，可以是试算表数据，XML数据或任何具有行集合概念的数据源。

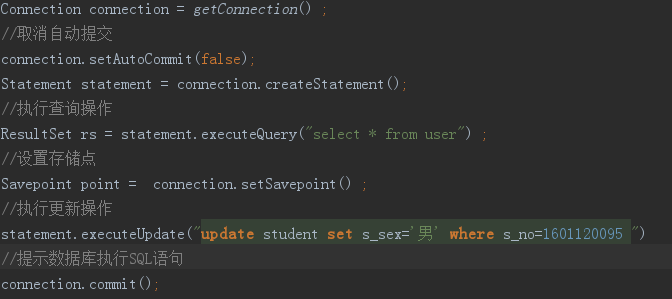
## 事物简介

事物的基本特性：原子性，一致性，隔离性，持续性。

在原子操作上，JDBC可以操作Connection中的setAutoCommit()方法，给它false自变量，提示数据库启动事物，在SQL语句执行之后，自行调用commit()方法，提示数据库确认操作。如果中间有错误，则调用rollback()，提示数据库撤销所有操作。

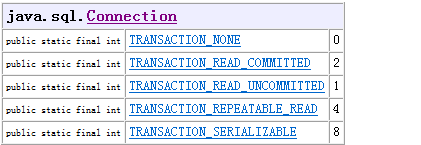


如果在事物管理时，仅想撤回某一个SQL执行点，则可以设置存储点（Save Point）



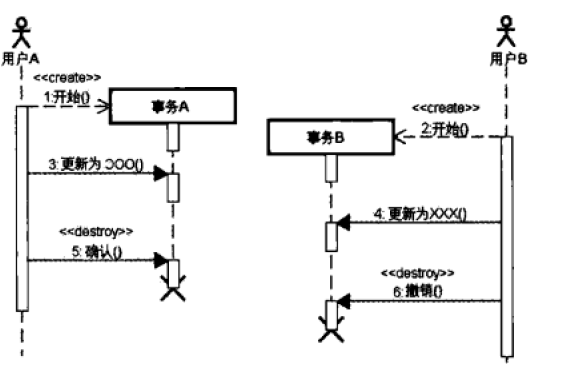
### 事物隔离级别

在隔离性上，JDBC可以通过Connection中getTransactionIsolation()方法取得数据库目前的隔离行为设置，可以设置常数是定义在Connection上的，如下：



#### 更新遗失（Lost update）

若某个字段的数据为ZZZ，用户A，B分别在不同时间点对同一个字段进行更新事物：



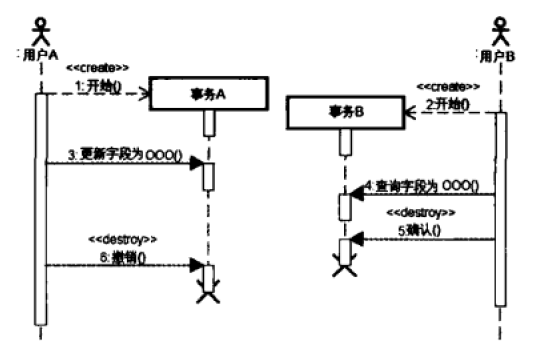
在完全没有隔离两者事物的情况下，用户B撤销操作时间在用户A确认之后，因此最后字段结果会是ZZZ，用户A看不到他更新确认的OOO结果。

如果避免这种更新遗失问题，可以设置隔离层级为“可读取未确认”（Read uncommitted），也就是A事物已更新但为确认的数据，B事物仅可作读取操作。

**TRANCATION\_READ\_UNCOMMITTED**： 允许脏读，不可重复读和幻读，禁止更新丢失。

#### 脏读（Dirty read）

两个事物同时进行，其中一个事物更新数据但未确认，另一个事物就读取数据，就可能发生脏读问题：



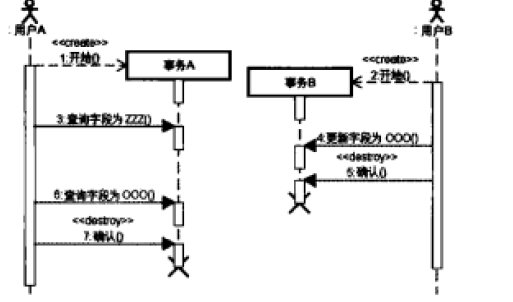
用户B在A事物撤销前读取了字段数据为OOO，如果A事物撤销了事物，那么用户B读取的数据就是不正确的。

**TRANCATION\_READ\_COMMITED：**禁止脏读，允许不可重读读和幻读。

数据库对此隔离行为的基本做法是：读取的事物不会阻止其他事物，未确认的更新事物会阻止其他事物。

#### 无法重复的读取（Unrepeatable read）

某个事物两次读取同一个字段的数据并不一致。如：事物A在事物B更新前后进行数据的读取，则A事物会得到不同的结果。



避免无法重复的读取问题，可以设置隔离层级为“可重复读取”，

**TRANCATION\_REPEATABLE\_READ：**禁止脏读和不可重复读，单独运行幻读。

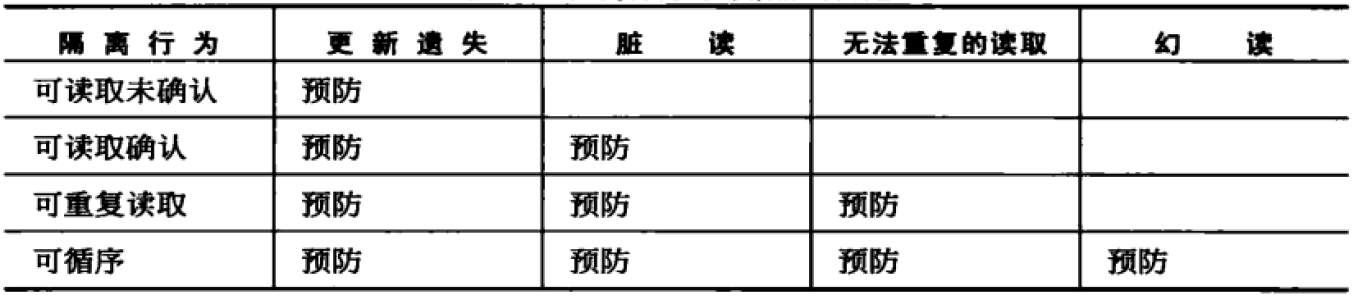
#### 幻读（Phantom read）

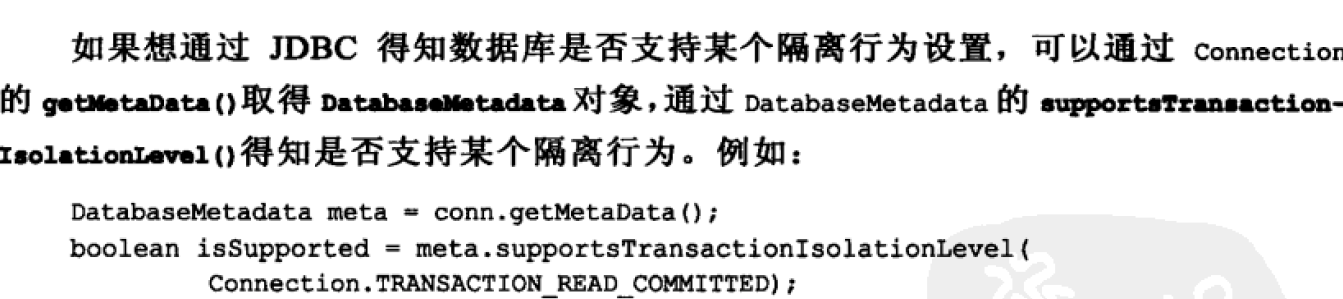
如果同一事物期间，读取到的数据笔数不一致。如：事物A第一次读取得到五笔数据，此时事物B新增了一笔数据，导致事物B再次读取得到的六笔数据。

如果隔离行为设置为可重复读取，但发生幻读现象，可以设置隔离层次为“可循序”，也就是在有事物时若有数据不一致的疑虑，事物必须可以按照顺序逐一进行。

**TRANSACTION\_SERIALIZABLE**：禁止脏读，不可重复读和幻读。

**TRANSACTION\_NONE：**表示对事物不设置隔离行为，仅适用于没有事物功能，以只读功能为主，不会发生同时修改字段的数据库。





# Web容器安全管理

