# JSP内置对象概览

在JSP中为了方便开发，提供了9个内置对象。这些内置对象由容器自动实例化，直接 拿来使用即可，不需要像Java中那样通过new关键字进行实例化对象后才可以使用。

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 内置对象 | 类型 | 描述 |
| 1 | pageContext | javax.servlet.jsp.PageContext | JSP的页面容器 |
| 2 | request | javax.servlet.http.HttpServleRequest | 用户的请求信息 |
| 3 | response | javax.servlet.http.HttpServletResponse | 服务器向客户端的回应信息 |
| 4 | session | javax.servlet.http.HttpSession | 表示一次会话 |
| 5 | application | javax.servlet.ServletContext | 表示所有用户的共享信息 |
| 6 | config | javax.servlet.ServletConfig | 服务器配置，可以取得初始化参数信息 |
| 7 | out | javax.servlet.jsp.JspWriter | 页面输出 |
| 8 | page | java.lang.Object | 表示从页面中表示出来的一个Servlet实例 |
| 9 | exception | java.lang.Throwable | 表示JSP页面所发生的异常，在错误页中才起作用 |

1. 九个内置对象中比较成员的是pageContext、request、response、session、application

这5个内置对象。

1. 除了把对象名称记住，还需要记住对象所对应的类型。因为在开发中，一定会使用

doc文档进行类或方法的查询，这时只能通过类型找到方法。例如，要想知道request

对象有哪些方法，一定要查询javax.servlet.http.HttpServletRequest才知道。

# **四种属性范围**

属性保存范围指的是一个内置对象可以在多个页面中保存并继续使用。在JSP中存 在4 种属性范围：

1. page：只能在一个页面中保存属性，跳转之后无效（使用pageContext对象表示）
2. request：可以在一个页面中保存，并且服务器端跳转后的页面依然有效（使用request

对象表示）。

1. session：在一次会话范围中保存，无论何种跳转都可以使用，但是在新打开一个浏 览器后后无法继续使用（使用session对象表示）。
2. application：在整个服务器上保存，所有用户、页面、跳转都可以使用，新开浏览器 也可以使用，但服务器重启后无效（使用application对象表示）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **范围（对象）** | **本页** | **服务端跳转后** | **客户端跳转后** | **说明** |
| page | Y | N | N |  |
| request | Y | Y | N |  |
| session | Y | Y | Y | 重新打开一个浏览器无效 |
| application | Y | Y | Y | 重新打开任意个浏览器依然有效；但重启服务器后无效 |

这四个内置对象都支持如下所示的属性操作方法：

**·public void setAttribute( String name , Object o )：设置属性的名称及内容**

**·public Object getAttribute( String name )：根据属性名获取属性内容**

**·public void removeAttribute( String name )：删除指定的属性**

注意：使用getAttribute( )方法获取属性内容时须进行向下转型，因为它返回的Object类型对象。

## page范围

page范围的属性（使用pageContext对象设置）表示将属性设置在本页上，只能在本页中有效、跳转之后无法取得。

## request范围

request范围的属性（使用request对象设置）在本页中有效，最关键的是request范围的属性在服务器端跳转后依然有效，但客户端跳转后无效。

## session范围

1. 如果希望一个属性在设置后，可以在任何一个与设置页面相关的页面中取得，则可以使用session属性范围。
2. session范围的属性（使用session对象设置）在本页中有效、且不管是服务器端跳转还是客户端跳转，均有效。但打开一个新的浏览器则无效，因为session属于一次会话。
3. 每一个新的浏览器连接上服务器后就是一个新的session。每一个浏览器连接到服务

器后，实际上都表示各自的session，表示每一位不同的上 网者都有各自的属性 ，所以新的浏览器打开后无法取得其他session对象设置的属性。

## application范围

1. 如果希望设置一个属性，可以让所有用户（每一个session）看得见，则可以将属性的范围设置成application，这样属性即可保存在服务器上。但需要注意，aplication范围的属性设置过多会影响服务器性能，因为application范围的属性是将属性设置到一个服务器中，如果设置过多的话，则肯定会影响服务器性能。
2. application范围的属性（使用applicationContext对象设置）在整个服务器中有效。

但重启服务器后，所设置的属性将全部消失。

## 深入研究page属性范围

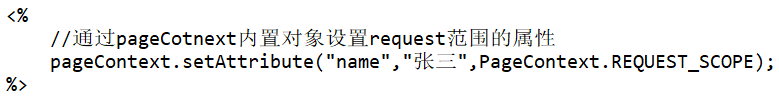
page范围的属性可以通过pageContext内置对象设置，但是从javax.servlet.jsp.PageContext类中可以发现，它有这么一个设置属性的方法：

**public void set Attribute( String name , Object value, int scope )**

与之前所使用的setAttribute( )方法不同，可以发现它有一个int的整型变量，此变量可以指定一个属性的保存范围。而在pageContext类中也同样存放存在4个表示属性范围的整型常量，可以直接通过这些整型常量指定scope的内容。如下：

|  |  |
| --- | --- |
| 属性名 | 描述 |
| public static final int PAGE\_SCOPE | 表示page范围（默认） |
| public static final int REQUEST\_SCOPE | 表示request范围 |
| public static final int SESSION\_SCOPE | 表示session范围 |
| public static final int APPLICATION\_SCOPE | 表示application范围 |

例如：



可以发现pageContext对象可以设置任意范围的属性，而其他操作也只是对这一功能的再包装而已。但一般还是习惯于使用pageContext对象设置保存在一页范围的属性，很少使用pageContext对象设置其他范围的属性。

# request对象

1. request对象是使用最多的一个对象，其主要作用是接受客户端发送而来的请求信息，如请求的参数、发送的头信息等都属于客户端发送来的信息。
2. request对象是javax.servlet.http.HttpServletRequest接口的实例化对象，此接口的定义如下：

**public interface HttpServletRequest extends ServletRequest**

可以发现HttpServletRequest接口是ServletRequest接口的子接口，所以在查找request对象的方法时除了要查询HttpServletRequest接口，也要查询ServletRequest接口。

3）request内置对象的常用方法有：

|  |  |  |
| --- | --- | --- |
| No | 方法 | 说明 |
| 1 | **public String getParameter( String name )** | **接收客户端发来的请求参数内容** |
| 2 | **public String[] getParameterValues( String name )** | **取得客户端发来的一组请求参数内容** |
| 3 | **public Enumeration getParameterNames( )** | **取得全部请求参数的名称** |
| 4 | **public String getRemoteAddr( )** | **得到客户端的IP地址** |
| 5 | **void setCharacterEncoding( String env )throws UnsupportedEncodingException** | **设置该页面统一的请求编码** |
| 6 | public boolean isUserInRole ( String role ) | 进行用户身份的验证 |
| 7 | **public Httpsession getSession( )** | **取得当前的session对象** |
| 8 | **public StringBuffer getRequestURL( )** | **返回正在请求的路径，一个完整的http请求路径** |
| 9 | public Enumeration getHeaderNames( ) | 取得全部请求头信息的名称 |
| 10 | public String getHeader( String name ) | 根据头信息名称取得对于的值 |
| 11 | **public String getMethod( )** | **取得用户的提交方式** |
| 12 | **public String getServletPath( )** | **取得访问的路径** |
| 13 | **public String getContextPath( )** | **取得上下文资源路径** |

## 解决乱码

在Web开发中，使用request对象调用getParameter( )方法接收请求餐宿是最常见的操作，但是，在进行参数提交时，提交的中文会存在乱码问题，所以request对象在接收请求参数前，往往**需要调用setCharacterEncoding( )方法设置统一的编码。**



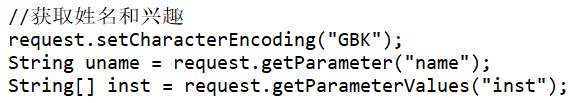
除了可以这样解决乱码，还可以通过过滤器进行编码的设置。

## 接收请求参数

|  |  |
| --- | --- |
| String getParameter( String name ) | 接收一个参数值 |
| String[] getParameterValues( String name ) | 接收一组参数值（如复选框） |

1. 使用request对象中的getParameter( )方法可以接收一个表单文本框中输入的内容。 而实际上getParameter( )是根据参数名取得参数的内容，也就是说文本框的name属 性值就是参数名，而文本框中输入的内容就是该参数的值。但是这种方式只适合每 次接收一个参数，如果有一组参数（同名参数）传递，则必须使用getParameterValues( ) 方法。
2. 在表单控件中，像文本框（text）、单选按钮（radio）、密码框（password）、隐藏域 （hidden）等，一般都使用getParameter( )方法进行接收，因为这些控件在使用时参 数的值都只有一个；而像复选框（checkbox），参数值可以有多个，所以只能使用

getParameterValues( ) 方法接收，如果不小心使用了getParameter( ) 方法，则只会接收第一个选中的内容。使用getParameterValues( )方法接收一组参数值时，将以字符串数组的形式返回。在使用循环将值取出的时候，在取出前最好先判断其字符串数组是否为null，避免产生空指向异常（NullPointException）。返回的内容如果是null，此时如果调用length，就会产生NullPointerException，所以在使用前最好先判断接收来的参数是否为null。

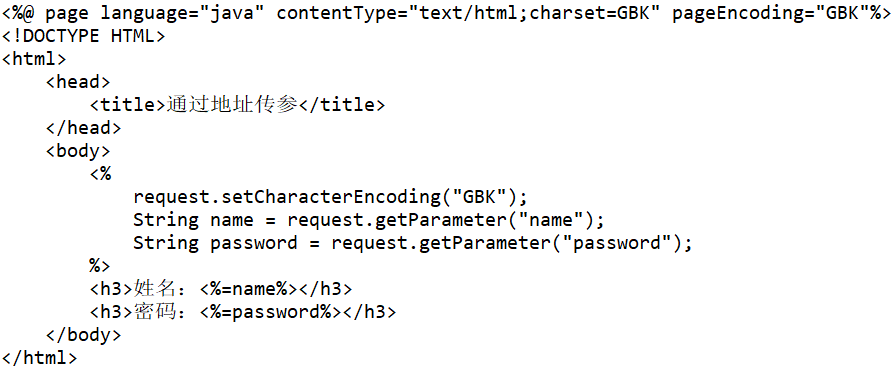


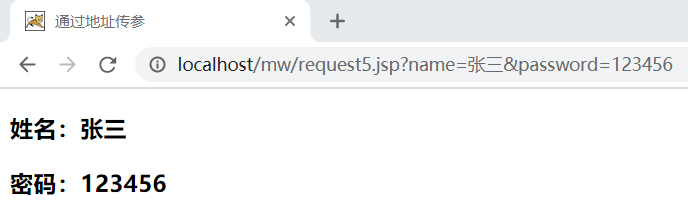
1. 在Web开发中，所有参数不一定非要由表单传递过来，也可以使用地址重写的方式

进行传递。地址重写的格式如下：

**动态页面地址?参数名称1=参数值1&参数名称2=参数值2&…**

所有的参数与之前的地址之间使用“？”分离，然后按照“参数名称=参数值”的格式传递参数，多个参数使用“&”分离。





1. 在request内置对象中还有一个灵活的方法，即getParameterNames( )，此方法可以

返回所有请求参数的名称，但是此方法返回值的类型是Enumeration，接口实例所以需要使用hasMoreElements( )方法判断是否有内容以及使用nextElement( )方法取出内容。（代码示例请参考教材书）

## get和post区别

在HTML中的<from>标签中有get和post两种表单提交方式，它们的区别如下：

1. 使用get提交，提交的内容会显示在地址栏中；而使用post提交，提交的内容不会 显示在地址栏中。直接输入地址栏访问页面属于get提交方式，而post只是应用在

表单上的操作。

1. get请求需要在地址栏上显示信息，所有信息的长度有所限制，一般大小时4~5KB 的数据；而post不会显示在地址栏，可以提交更多的内容，如果表单中有一些大文 本或者图像数据，则只能使用post提交方式。
2. post提交比get提交安全。

## 获取头信息

Java的Web开发使用的是HTTP协议，主要操作就是基于请求和回应，但是在请求和回 应的同时也会包含一些其他信息（如客户端的IP、Cookie、语言等），这些额外的信 息称为头信息。

可以通过request内置对象的getHeaderNames( )方法取得所有头信息的名称，结果返回一个Enumeration接口对象，然后进行遍历，取出每个头信息的名称，进而使用request对象调用getHeader( )根据头信息的名称取出对应的值。

## 角色验证

安装Tomcat时，提醒过用户输入一个管理的用户名和密码，实际上就相当于建立了一个管理员的账户。如果现在某些JSP需要输入特定的管理员的账号才能访问，那么就需要进行角色验证，而要进行角色验证，可以使用request对象的isUserInRole( )方法完成。

## 其他操作

在request中也可以取得客户端的IP地址、访问的路径信息，提交的方式等。





request内置对象在实际的开发中使用最多，读者一定要记住的是，当服务端需要得到请求客户端的相关信息时就会使用request对象完成。

# response对象

1. response对象的主要作用是对客户端的请求进行回应，将Web服务器处理后的结果 发回给客户端。
2. response对象属于javax.servlet.http.HttpServletResponse接口的实例，其接口的定义

如下：

**public interface HttpServletResponse extends ServletResponse**

从接口定义可以发现，此接口是ServletResponse接口的子接口，所以在查找reponse对象的方法时除了要查HttpServletResponse接口，也要查ServletResponse接口。

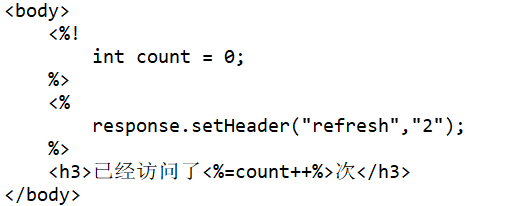
1. response对象的常用方法如下：

|  |  |
| --- | --- |
| **方法** | **描述** |
| public void setHeader( String name , String value ) | 设置回应的头信息 |
| public void sendRedirect ( String location )throws IOException | 页面跳转 |
| public void addCookie( Cookie cookie ) | 向客户端增加Cookie |

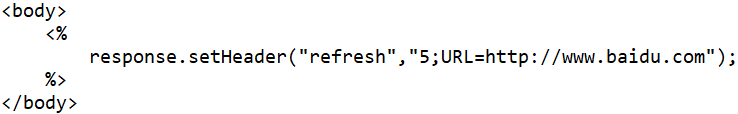
## 设置头信息

之前讲解果，客户端在进行请求时会发送额外的信息，这些就是头信息。服务器端也可以根据需要向客户端设置头信息，在所有头信息的设置中，定义刷新页面的头信息使用的最多，用户可以直接使用response对象的setHeader( )方法将头信息名称设置为refresh，实现页面的定时刷新或定时跳转。 但需要注意两点：

1. 通过response对象设置的定时刷新或定时跳转属于客户端类型跳转。
2. 在HTML文件中如果想执行定时跳转可以使用meta标签实现页面的定时跳转。
3. 设置定时以秒为单位，如果设置为0，则立即执行刷新或跳转操作。
4. 定时刷新

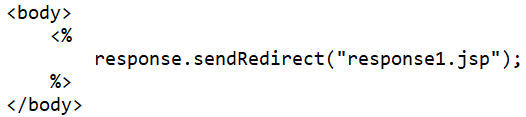


2、定时跳转



## 页面跳转

response对象除了可以通过设置头信息的方式完成跳转外，还可以使用response对象的sendRedirect( )方法直接完成页面跳转。但这种跳转和设置头信息的跳转一样，都属于客户端跳转。



还可以通过地址的方式传参：



## response跳转与<jsp:forward>跳转的对比

使用<jsp:forward>标签可以完成服务器端跳转，，使用response对象的sendRedirect( )方法可以完成客户端跳转。那这两种跳转有什么区别呢？

1）<jsp:forward>属于服务器端跳转，跳转之后地址栏的信息不会有任何的改变；而

response.sendRedirect( )属于客户端跳转，跳转之后地址栏是会改变的，变为跳转之后的页面地址。而且在使用request范围的属性时，只有服务器端跳转才能够将request范围的属性保存到跳转页；而如果是客户端跳转，则无法进行request范围的属性的传递。

1. 另外，如果使用的是<jsp:forward>标签完成服务器端跳转，则执行到<jsp:forward>标签时会立即进行跳转，后面的代码不再执行；如果使用的是response.sendRedirect( )完成客户端跳转，则是在整个页面执行完之后才执行response.sendRedirect( )方法进行跳转。由于这两种跳转存在的差异，所以在以后的代码开发中，尤其是在使用了JDBC的操作中，一定要在<jsp:forward>语句执行之前关闭数据库的连接，否则数据库连接将再也无法关闭。而如果数据库连接没有关闭，当达到一定程度时将出现“数据库连接已经达到最大的异常”，此时只能重新启动服务器了。
2. 在传参方面，如果使用了<jsp:forward>，可以通过<jsp:param>进行参数的传递；而如果使用了request.sendRedirect( )传递参数，则只能通过地址重写的方式完成。

在实际开发中，服务端跳转要比客户端跳转更常用，这一点要结合MVC设计模式才能做到深入的掌握。

## 操作Cookie

Cookie是浏览器提供的一种技术，这种技术让服务器端的程序能将一些只须保存在客户端、或者在客户端进行处理的数据，放在本地使用的计算机中，不须通过网络的传输，因而提高网页处理的效率，同时也能够减少服务器端的负载，并且使用Cookie保存信息可以减少客户端的部分操作。但是由于Cookie是服务器端保存在客户端的信息，所以其安全性也是很差的。如网站的记住密码功能，但如果没有及时地清理掉使用的账户，就有可能出现安全隐患。

1）在JSP中专门提供了javax.servlet.http.Cookie操作类，其常用的方法有：

|  |  |
| --- | --- |
| 方法 | 描述 |
| public Cookie( String name , String value ) | 实例化Cookie对象，同时设置名称和内容 |
| public String getName( ) | 取得Cookie的名称 |
| public String getValue( ) | 取得Cookie的内容 |
| public void setMaxAge( int expiry ) | 设置Cookie在客户端的保存时间，以秒为单位 |

在声明Cookie对象设置名称和内容后，记得设置Cookie的保存时间。否则这个 Cookie只针对一次打开浏览器有效，当浏览器重新启动后，之前所设置的全部 Cookie就不在了。所以，如果要想真正的将Cookie保存在客户端上，就必须设置 Cookie的保存时间。使用Cookie类提供的setMaxAge( )即可。

1. 所有的Cookie是由服务器端设置到客户端上，所以要想客户端增加Cookie则必须使用response对象的addCookie( )方法：

|  |  |
| --- | --- |
| 方法 | 描述 |
| public void addCookie ( Cookie cookie ) | 向客户端设置Cookie |

3）要想取得客户端设置的Cookie，可以通过request对象的getCookies( )方法完成：

|  |  |
| --- | --- |
| 方法 | 描述 |
| public Cookie[ ] getCookies ( ) | 取得客户端设置的全部Cookie |

此方法返回一个Cookie对象数组，然后通过循环遍历的方式取得Cookie的名称和内容。request对象之所以可以取得设置的全部Cookie，是因为客户端每次向服务器端发送请求时都会将之前设置的Cookie随着头信息一起发送到服务器上。

在取出Cookie时会存在一个名称为JSESSIONID的Cookie以及对应的值，这是系统自动设置的。因为，在每一个客户端访问服务器时，服务器为了明确区分每个客户端，都会自动设置一个JSESSIONID的Cookie，用于表示用户的唯一身份标识。

1. 设置Cookie时，如果没有设置Cookie的保存时间，则设置的Cookie只在本次会话有效，重新打开浏览器开启一个新会话，将无法获取，还有可能出现NullPointerException异常。此时，如果要先真正将Cookie保存在客户端，就必须设置Cookie的保存时间，使用Cookie类提供的setMaxAge( )方法即可，以秒为单位。虽然Cookie中可以保存信息，但是并不能无限制地保存，一般一个客户端最多只能保存300个Cookie。

除了使用response对象的addCookie( )向客户端添加Cookie外，还可以使用它的setHeader( )方法通过设置头信息的方式向客户端添加Cookie。

需要注意的是，Cookie是设置在浏览器上的，不能跨浏览器。

# session对象

在实际开发中，session对象最主要的用处就是完成用户登录（login）、注销（logout）等常见功能，每一个session对象都表示不同的访问用户，session对象是javax.servlet.http.HttpSession接口的实例化对象，表示的是一次会话。其常用的操作有：

1. 取得session id
2. 完成登录与注销的操作
3. 判断用户是否是新用户（是否是第一次访问这个页面）
4. 取得用户操作的时间。

session最主要的是可以完成用户登录的合法性验证。

## 取得session id

1. 当一个用户连接到服务器后（用户第一次访问服务器动态资源时），服务器会自动为此session分配一个不会重复的session id，并自动将其添加到客户端Cookie中，之后客户端每次向服务器发送请求时，会将Cookie（包括了session id）作为头信息发送到服务器，服务器依靠不同的session id来区分每一个不同的用户。
2. 可以使用HttpSession接口中的getId( )方法取得session Id。
3. 对于每一个已连接到服务器的用户，如果重启服务器，则这些用户再次发出请求实际上表示的都是一个新连接的用户，服务器后为每个用户重新分配一个新的session id。如果想让session在服务器重启后依然可以继续使用，可以进行session的持久化。

## session持久化

在实际操作中，如果服务器重新启动，则已经分配的session id就会消失，已经连接的用户再次发出请求实际上表示的就是一个新连接的用户，服务器会为每个用户重新分配一个新的session id。那么有没有一种处理机制可以让这些session继续保存，使得等待服务器重新启动后可以继续使用，可以通过序列化的方式保存session继续使用，在Tomcat中可以通过配置server.xml文件，将每一个用户的session在服务器关闭时序列化到存储介质（可以是文件或数据）上保存，这样即使服务器重新启动，也可以通过反序列化的方式，从指定的存储介质上反序列化每一个session对象。（详细操作请参考教材书）。

## 登录注销操作

在各系统中几乎都会包括用户登录验证及注销功能，此功能可以使用session实现。具体思路是：当用户登录成功后，设置一个session范围的属性，然后在其他需要验证的页面中判断是否存在此session范围的属性，如果存在，则表示已经是登录状态的合法用户；如果属性不存在，则给出提示，提示用户登录，并跳转到登录页。用户登录后可以进行注销的操作，注销调用session对象的invalidate( )方法销毁此session的全部信息，其中就包括了设置的属性。

## 会话跟踪技术

使用session完成登录注销的操作属于会话跟踪技术中的一种。在Web开发中，一共存在4种会话跟踪技术：

1. 通过session对象提供的方法保存
2. 使用Cookie保存信息
3. 通过表单的隐藏域保存信息
4. 通过地址重写的方式保存信息

根据环境的不同，可以使用的操作不同。例如，如果没有严格要求，直接使用session对象即可；如果客户端禁用了Cookie操作，则只能通过隐藏域或地址重写的方式进行会话跟踪。例如，可以将一个session id设置在地址之后。格式如下：



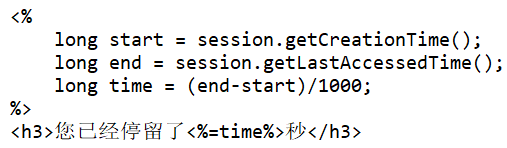
但是这种做法操作会存在一些麻烦，所以在开发中一般使用session属性范围保存客户信息，而由于客户端禁用Cookie所造成的问题并不是程序开发中所应考虑的。

## 判断新用户

在session对象中可以使用isNew( )方法判断一个用户在本次会话中是否是第一次访问页面。其判断的原理是通过Cookie的方式进行的，通过判断用户的session id是否在服务器上保存，如果保存了，则是老用户，否则是新用户。

## 取得用户的操作时间

在session对象中，可以通过getCreation Time( )方法取得一个session的创建时间（用户通访问服务器的任意动态资源时，session即创建），也可以使用getLastAccessedTime( )方法取得一个session的最后一次操作时间。



# application对象

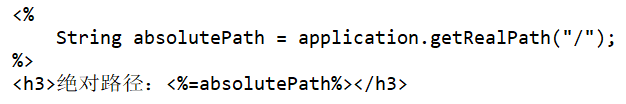
application内置对象是javax.servlet.ServeltContext接口的实例化对象，表示Servlet的 上下文，代表了整个容器的操作。常用的方法如下：

|  |  |
| --- | --- |
| 方法 | 描述 |
| public String getRealPath( String path) | 得到真实路径 |
| public Enumeration getAttributeNames( ) | 得到所有属性名称 |
| public String getContextPath( ) | 取得当前虚拟路径名称 |

其常用的操作有：取得真实路径完成IO操作、网站计数器、取得全部属性。

## 取得绝对路径

使用application对象的getRealPath( )方法可以取得当前网站的绝对路径。



## 代替application对象

对于application对象而言，可以使用getServletContext( )方法代替。



这里的方法表示的是由容器调用，实际上ServletContext本身就表示整个容器。

## 完成IO操作

既然可以使用application对象取得绝对路径，那么就可以进行文件IO操作。

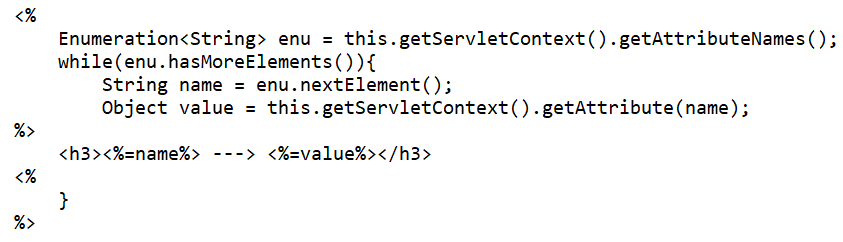
## 网站计数器

网站计数器可以计算网站的访问量。

1. 网站的来访人数可能会有很多，有可能超过20位整数，那么只靠基本数据类型将难以保存，所以必须使用大整数类BigInteger完成。
2. 用户每次在第一次访问时才需要进行计数的操作，而重复刷新页面时则不应该重复计数，所以在执行计算之前必须使用isNew( )方法判断用户是否是第一次访问。
3. Web开发属于多线程凑在哦，所以在进行更改、保存时需要进行同步操作。

## 查看application范围的属性

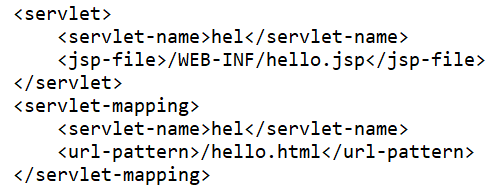
在application对象中提供了getAttributeNames( )方法，可以取得全部属性的名称，进而还可以取得对应的值。



# Web安全性及config内置对象

## Web安全性

每个Web项目中一般都存在一个WEB-INF目录，WEB-INF目录的安全性是最高的，不会被外界所访问到，（注意WEB-INF目录名必须大写），任何文件放在WEB-INF目录中都会受到很好的保护。例如将一个JSP文件放在该目录中，则此文件也会受到很好的保护，但是这样一来外界就无法访问该文件，而此时我们可以修改WEB-INF目录中的web.xml文件，为该文件加入一个映射操作，这样就可以通过映射路径的方式访问该文件。修改WEB-INF目录中的web.xml文件，添加配置：



<servlet-name>节点的主要功能是连接<servlet>节点和<servlet-mapping>节点，此节点的名称只在配置文件的内部其作用，并且多个配置不能重名。

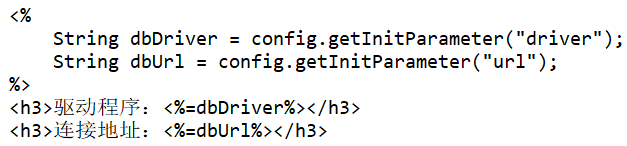
## config对象

config对象时javax.servlet.ServletConfig接口的实例化对象，主要的功能是取得一些初始化的配置信息。常用方法如下：

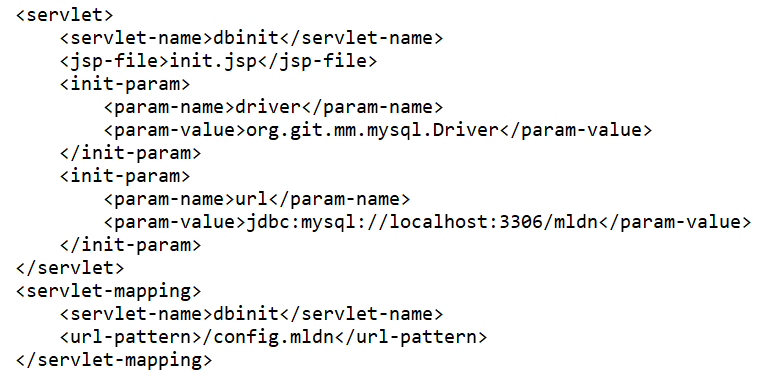
|  |  |
| --- | --- |
| 方法 | 描述 |
| public String getInitParameter( String name ) | 取得参数名对应的参数值 |
| public Enumeration getInitParameterNames( ) | 取得全部的初始化参数名称 |

所有的初始化参数是在web.xml文件中配置的。

示例，编写init.jsp文件：



例在web.xml中映射init.jsp并给它配置初始化参数（配置完成后记得重启服务器）：



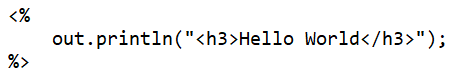
# out内置对象

out内置对象是javax.servlet.jsp.JspWriter类的实例化对象，主要功能是使用print( )或println( )方法向页面输出信息，但是从在实际开发中，一般使用表达式完成输出操作。另外，还可以操作获取缓冲区的信息。常用方法如下：

|  |  |
| --- | --- |
| 方法 | 描述 |
| print( )或println( ) | 向页面输出信息 |
| public int getBufferSize( ) | 返回JSP中缓冲区的大小 |
| public int getRemaining( ) | 返回JSP中未使用缓冲区的大小 |

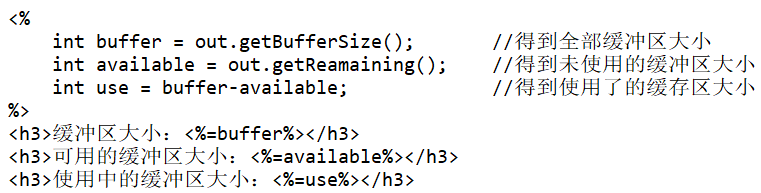
## 页面输出

使用out对象的print( )或println( )方法可以向页面输出内容。



## 获取缓冲区信息

使用out对象还可以获取缓冲区的信息，如查看缓冲区的大小，可用缓冲区等。



# **pageContext对象**

pageContext内置对象是javax.servlet.jsp.PageContext抽象类的实例化对象，主要表示的是一个JSP页面的上下文。除了之前讲解果的操作属性的方法外，其他常用的方法如下：

|  |  |
| --- | --- |
| 方法 | 描述 |
| public abstract void forward ( String relativeUrlPath ) throws ServletException , IOException | 页面跳转（服务器端类型） |
| public void include( String relatieUrlPath ) throws  ServletException , IOException | 页面包含（动态包含类型） |
| public ServletConfig getServletConfig ( ) | 取得config对象 |
| public ServletContext getServletContext ( ) | 取得ServletContext对象 |
| public ServletRequest getRequest ( ) | 取得ServletRequest对象 |
| public ServletResponse getResponse ( ) | 取得ServletResponse对象 |
| public HttpSession getSession ( ) | 取得HttpSession对象 |

* 1. 使用pageContext对象的forward( )方法可以完成服务器端页面跳转的功能，与

<jsp:forward>跳转标签的功能相同，且两者都属于服务器端跳转。但不同的是，使用<jps:forward>跳转指令是通过<jsp:param>向跳转的页面传递参数，而pageContext对象的forward( )方法是通过地址重写的方式向跳转的页面传递参数。

* 1. 使用pageContext对象的include( )方法可以完成页面包含的功能，与<jsp:include>动态包含指令功能相同。但不同的是，使用<jsp:include>动态包含指令是通过 <jsp:param>标签向被包含的页面中传递参数，而pageContext对象的include( )方法是通过地址重写的方式向被包含的页面中传递参数。

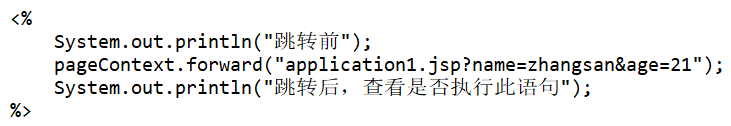
1. 可以使用pageContext对象的getRequset( )、getResponse( )、getSession( ) 、 getServletContext( )、getConfig( )可以取得其他内置对象。但特别需要注意的是，在

使用getReuqest( )和getResponse( )方法时并不能完全取得 request和response内置对象，取得的只是request和response对象所属接口的父 接口实例。所以，通过pageContext取得这两个内置对象后，这两个内置对象只能调用其所属接口的父接口中的方法。如果想让其变为真正的request，response对象，可以进行向下转型。

1. pageContext对象在一般的开发中很少直接使用，但是在标签编程中比较经常使用。

## 页面跳转

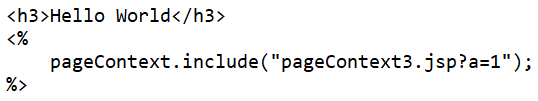
pageContext对象可以使用forward( )方法实现服务器端的页面跳转（与<jsp:forward>标签功能相同），通过地址传参的方式向跳转的页面传递参数。



通过观察运行结果可以发现，在执行跳转语句后，后面的语句依然会执行。

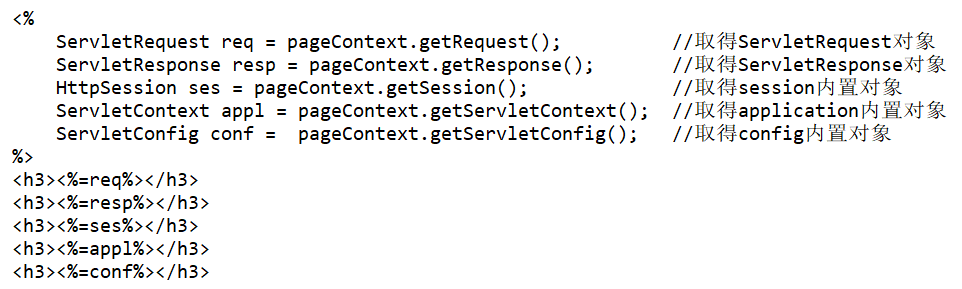
## 页面包含

pageConext对象也可以实现动态类型的页面包含功能（与<jsp:include>标签功能相同），如果要向被包含的页面传递参数，可以通过地址传参的方式向被包含的页面传递参数。



## 取得其他内置对象

pageContext对象可以取得request、response、session、application、config等内置对象。



但需要注意的是，在取得request、reponse内置对象时，返回的并不是真正的内置对象，而是request、response所属接口的父接口对象，此时，它们只能调用其父接口中的方法。如果想要获得真正的request、response内置对象，需要进行向下转型：

