JavaBeans

Table of Contents

[Scoped Parameters and Attributes 2](#_Toc100576621)

[ServletConfig Params 2](#_Toc100576622)

[ServletContext Params 3](#_Toc100576623)

[Session Attributes 4](#_Toc100576624)

[Request Attributes 4](#_Toc100576625)

[Page Scope Attributes 5](#_Toc100576626)

[JavaBeans 5](#_Toc100576627)

[JSP Actions 6](#_Toc100576628)

## Scoped Parameters and Attributes

We have examined several types of parameters and attributes so far. Each of these have a specific scope, which we will examine now.

### ServletConfig Params

*ServletConfig* is an instance in the Servlet API and is used to initialize a servlet.

<web-app>  
 <servlet>  
 <servlet-name>ServletConfigTest</servlet-name>  
 <servlet-class>ServletConfigTest</servlet-class>  
 <init-param>  
 <param-name>topicName</param-name>  
 <param-value>ServletConfigParams</param-value>  
 </init-param>  
 </servlet>  
</web-app>

XML

The parameters defined in the web.xml file are accessible by the servlet through the *ServletConfig* object. Since these parameters are only available to a specific servlet, their scope is limited to the **servlet**.

### ServletContext Params

Unlike *ServletConfig* objects, there is only a **single *ServletContext* object** per web application. All servlets use this same object. It is used to hold information about the whole web application or the execution environment. Thus, its scope is **application wide**.

<web-app>  
 <context-param>  
 <param-name>globalVariable</param-name>  
 <param-value>javacodegeek.com</param-value>  
 </context-param>  
</web-app>

XML

String value = getServletContext().getInitParameter("globalVariable");

JAVA

Another difference between *ServletConfig* and *ServletContext* objects is that *ServletContext* objects allow **attributes** as well as parameters. Parameters are defined in the web.xml file, are read-only and can only be strings. Attributes on the other hand, can be written and modified in code as shown below, and can hold any object type:

ArrayList<String> studentNames = new ArrayList<>();  
getServletContext().setAttribute("Students", studentNames);  
ArrayList<String> students = (ArrayList<String>) getServletContext().getAttribute("Students");

JAVA

### Session Attributes

We have previously used *HttpSession* objects. These objects allow us to set attributes which are limited to the **current session**. The attributes are destroyed as soon as the session ends.

ArrayList shoppingCart = new ArrayList();  
*HttpSession* session = request.getSession();  
session.setAttribute("CartObject", shoppingCart);  
ArrayList shoppingCard = (ArrayList) session.getAttribute("CartObject");

JAVA

### Request Attributes

The **request objects** which come from client requests have both parameters and attributes. The parameters contain form information submitted by the client which we cannot change. The attributes can be set by us. These attributes have a scope limited to the **lifetime of the request**. Once we send back a response to the client, these attributes are lost.

ArrayList shoppingCart = new ArrayList();  
request.setAttribute("CartObject", shoppingCart);  
ArrayList shoppingCart = (ArrayList) request.getAttribute("CartObject");

JAVA

### Page Scope Attributes

We can also have attributes that have scopes limited to a **JSP page**. These attributes are discarded when the request for the JSP page is completed. Page scope attributes are defined using **JavaBeans**, which we will explore next.

## JavaBeans

A **JavaBean** is actually a standard. Classes that follow the JavaBean standard must:

* Provide a **public, no-argument constructor**
* Implement the **java.io.*Serializable*** interface, which is used to convert objects to streams of bytes to be stored in files and convert streams of bytes from files back into objects
* Optionally have getter and setter methods for private properties

public class CounterBean implements *Serializable* {  
 private int counter;  
 CounterBean() {  
 counter = 0;  
 }  
 int getCounter() {  
 return counter;  
 }  
 void setCounter(int c) {  
 counter = c;  
 }  
}

JAVA

JavaBean classes such as the one shown above can be used for one of two purposes, either to execute business logic, or to carry data between layers.

### JSP Actions

If we want to use a JavaBean class inside a **scriplet**, it would look like this:

<%  
 CounterBean counterBean = new CounterBean();  
 counterBean.setCounter(5);  
 int count = counterBean.getCounter();  
%>

JSP

However, this can quickly become quite a lot of code. Additionally, combining lots of Java and HTML code together **decreases readability**. Instead, it is better to use **JSP Actions**.

We can use JSP Actions with JavaBean classes as follows:

<%@page import="com.beans.PersonBean"%>  
<html>  
<body>  
<jsp:useBean id="date" class="java.util.Date" scope="page" type="java.util.Date"/>  
<h1>Current Date: <%=date%></h1>  
<jsp:useBean id="person" class="com.beans.PersonBean" scope="page"/>  
<jsp:setProperty name="person" property="nid" value="256"/>  
<jsp:setProperty name="person" property="firstName" value="Adam"/>  
<h3>First name and NID of user:  
<jsp:getProperty name="person" property="firstName"/>  
and <jsp:getProperty name="person" property="nid"/></h3>  
</body>  
</html>

JAVA