SERVLETS

ServletConfig

- **ServletConfig** interface is used to assign properties to a Servlet at the time of Servlet object creation itself (known as Servlet Initialization Parameters) by the container.
- Exactly the way you use constructor in case of applications
- **web.xml** is known as deployment descriptor, where in, the deployment particulars (that also include **initialization parameters**) of a Servlet are written by the Programmer.
- The particulars are declared in <**init-param**> tag in **web.xml** file.

- An object of **ServletConfig** is created by the Container itself at the time it creates a Servlet object.

- **For every object of Servlet, the container creates, also an object of ServletConfig.**

- This ServletConfig object is used to communicate with the Servlet under execution.

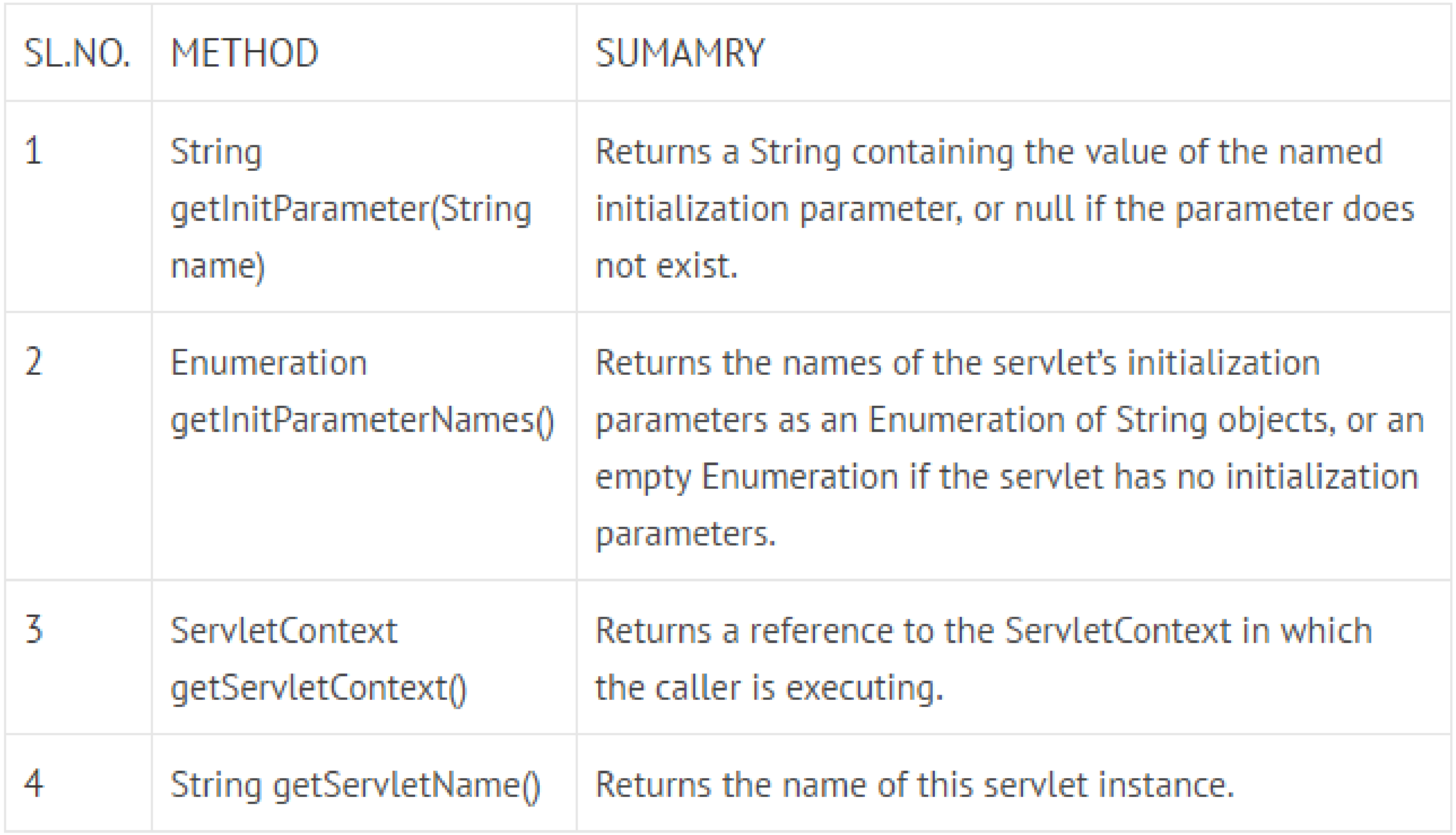
- Eventhough, the contianer creates the ServletConfig object implicitly for its purpose, the Programmer can make use of the object in his code to read **initialization parameters** into the Servlet from **web.xml** file.

- If the parameter values change, the **web.xml** file need not be compiled.
- At the time reading, whatever values exists in the **web.xml** file, with those values or parameters only the Servlet is initialized

Some points to be known with ServletConfig interface

- ServletConfig is an interface from **javax.servlet** package.
- The **getServletConfig()** method of **GenericServlet** returns an object of ServletConfig.
- The **ServletConfig** object created by the Web container for a specific Servlet cannot read other servlet's **init-param** data.
- ServletConfig object is specific for a Servlet. Other way to say, if 100 servlet objects are being executed in the container, 100 ServletConfig objects are also created implicitly used by the container to communicate with the Servlet.
- When the container destroys the Servlet object, along with it its corresponding **ServletConfig** object also destroyed.
- Programmer can make use ServletConfig object to read tags of the Servlet.
- <init-param> tag data is known as **initialization parameters** and are used to initialize the Servlet with some special properties.
- The method of ServletConfig object used to read is "String getInitParameter(String)".
- The data, to be read by ServletConfig, is written within <servlet> tag.

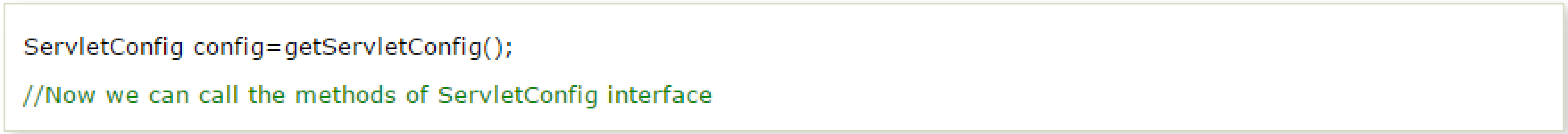
ServletConfig methods



SL.NO.	METHOD	SUMAMRY
1	String getInitParameter(String name)	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
2	Enumeration getInitParameterNames()	Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
3	ServletContext getServletContext()	Returns a reference to the ServletContext in which the caller is executing.
4	String getServletName()	Returns the name of this servlet instance.

How to get the object of ServletConfig

- **getServletConfig() method** of Servlet interface returns the object of ServletConfig.



```
ServletConfig config=getServletConfig();
//Now we can call the methods of ServletConfig interface
```

Accessing the Web Context

- The context in which Web components execute is an object that implements the ServletContext interface.
- Every web application has one and only one ServletContext and is accessible to all the active resources of that application.
- It is also used by the servlets to share data with one another.

ServletContext : A Closer Look

- This is an interface.
- It is used to communicate with ServletContainer.
- Due to this there is only one servletContext for the whole project/application.
- It is used to
  - set the attribues for application level
  - and get the initialization parameter values from web.xml

Context init parameters

- Like servletconfig paramter there is servlet context parameter .
- This is called as contextParameter.
- These parameters are defined in the web.xml.

Configuration in the web.xml

Outside all the <servlet> tag

- Add the <context-param>
  - Inside <context-param> tag add the
    <param-name>
    <param-value>

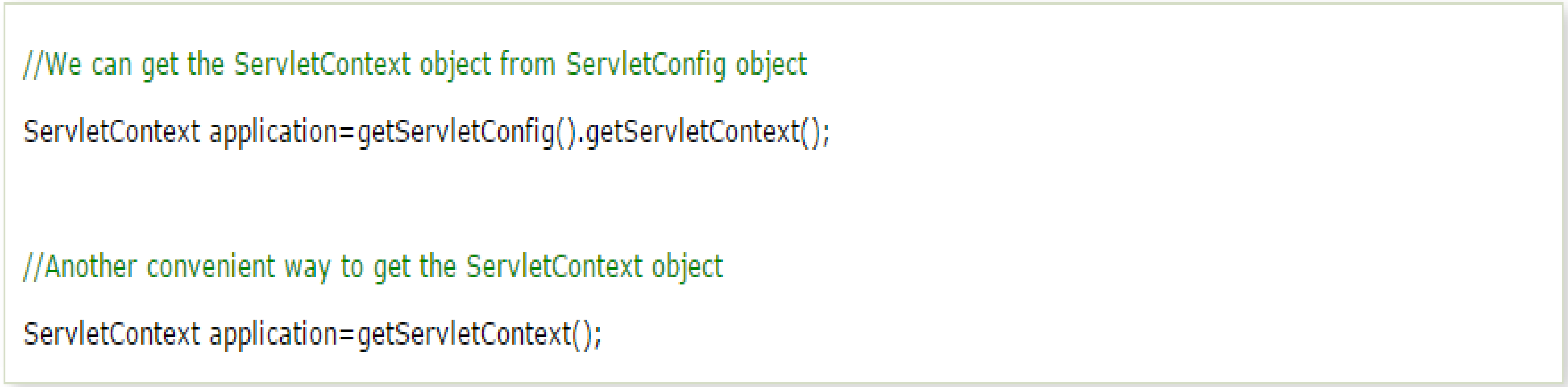
ServletContext methods

Method	Description
public String getInitParameter(String name)	Returns the parameter value for the specified parameter name.
public Enumeration getInitParameterNames()	Returns the names of the context's initialization parameters.
public void setAttribute(String name,Object object)	sets the given object in the application scope.
public Object getAttribute(String name)	Returns the attribute for the specified name.
public void removeAttribute(String name)	Removes the attribute with the given name from the servlet context.

How to get the object of ServletContext interface

- **getServletContext() method** of ServletConfig interface returns the object of ServletContext.

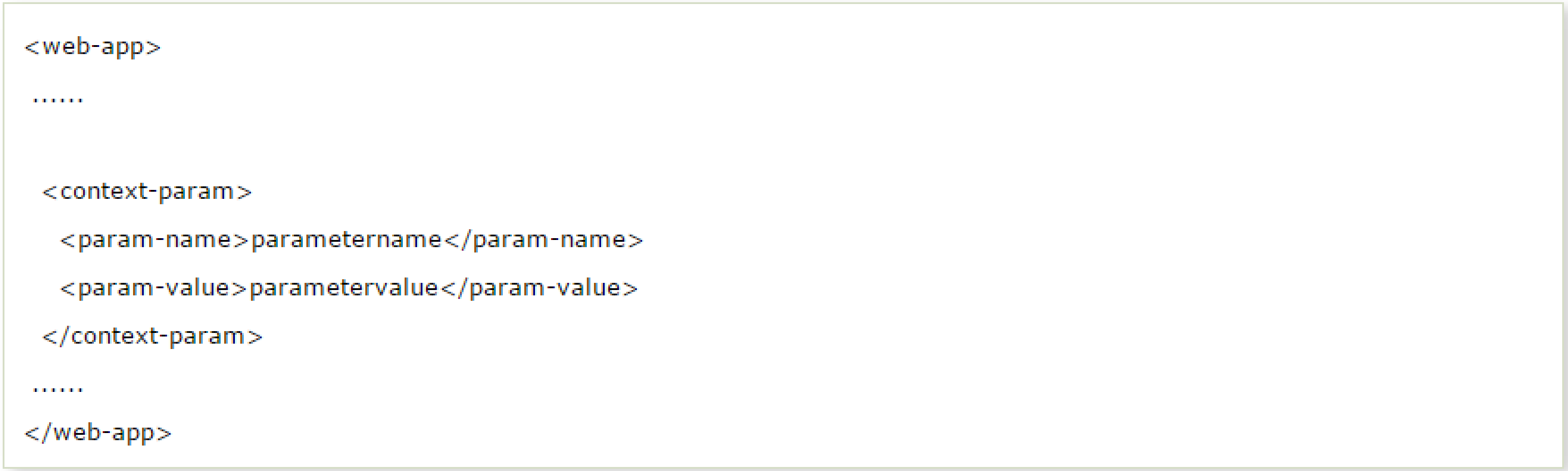
- **getServletContext() method** of GenericServlet class returns the object of ServletContext.



```
//We can get the ServletContext object from ServletConfig object
ServletContext application=getServletConfig().getServletContext();


//Another convenient way to get the ServletContext object
ServletContext application=getServletContext();
```

initialization parameter in Context scope

- The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope.
- The param-name and param-value are the sub-elements of the context-param.
- The param-name element defines parameter name and and param-value defines its value.



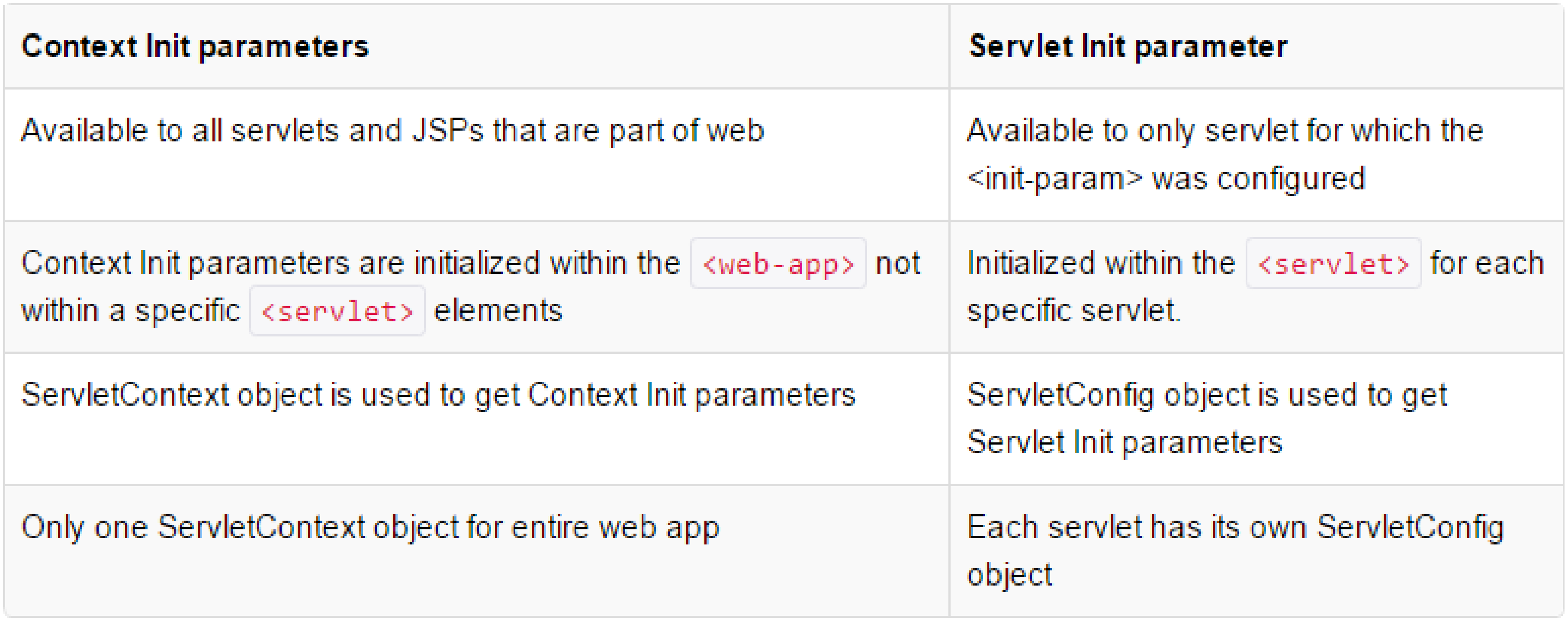
```
<web-app>
 ......


  <context-param>
    <param-name>parametername</param-name>
    <param-value>parametervalue</param-value>
  </context-param>
 ......
</web-app>
```

Difference between servlet init parameters and context parameters.

- **Inside the deployment descriptor**
  - **The context init parameters are included within the <web-app> element.**
  - **The servlet init parameters are included within the <servlet> element for each specific servlet.**

- **The servlet code is**
  - **getServletContext().getInitParameter(“foo");**
  - **getServletConfig().getInitParameter(“foo");**

Difference between Context Init Parameters and Servlet Init Parameter



Context Init parameters	Servlet Init parameter
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <init-param> was configured
Context Init parameters are initialized within the `<web-app>` not within a specific `<servlet>` elements	Initialized within the `<servlet>` for each specific servlet.
ServletContext object is used to get Context Init parameters	ServletConfig object is used to get Servlet Init parameters
Only one ServletContext object for entire web app	Each servlet has its own ServletConfig object

Managing State in Servlets

- We all know that **HTTP** is a stateless protocol.
- All requests and responses are independent.
- But sometimes you need to keep track of client's activity across multiple requests.

- For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

State Management in Servlets

- **State Management** is a mechanism used by the **Web container** to store state information for a particular user. There are four different techniques used by Servlet application for state management. They are as follows:

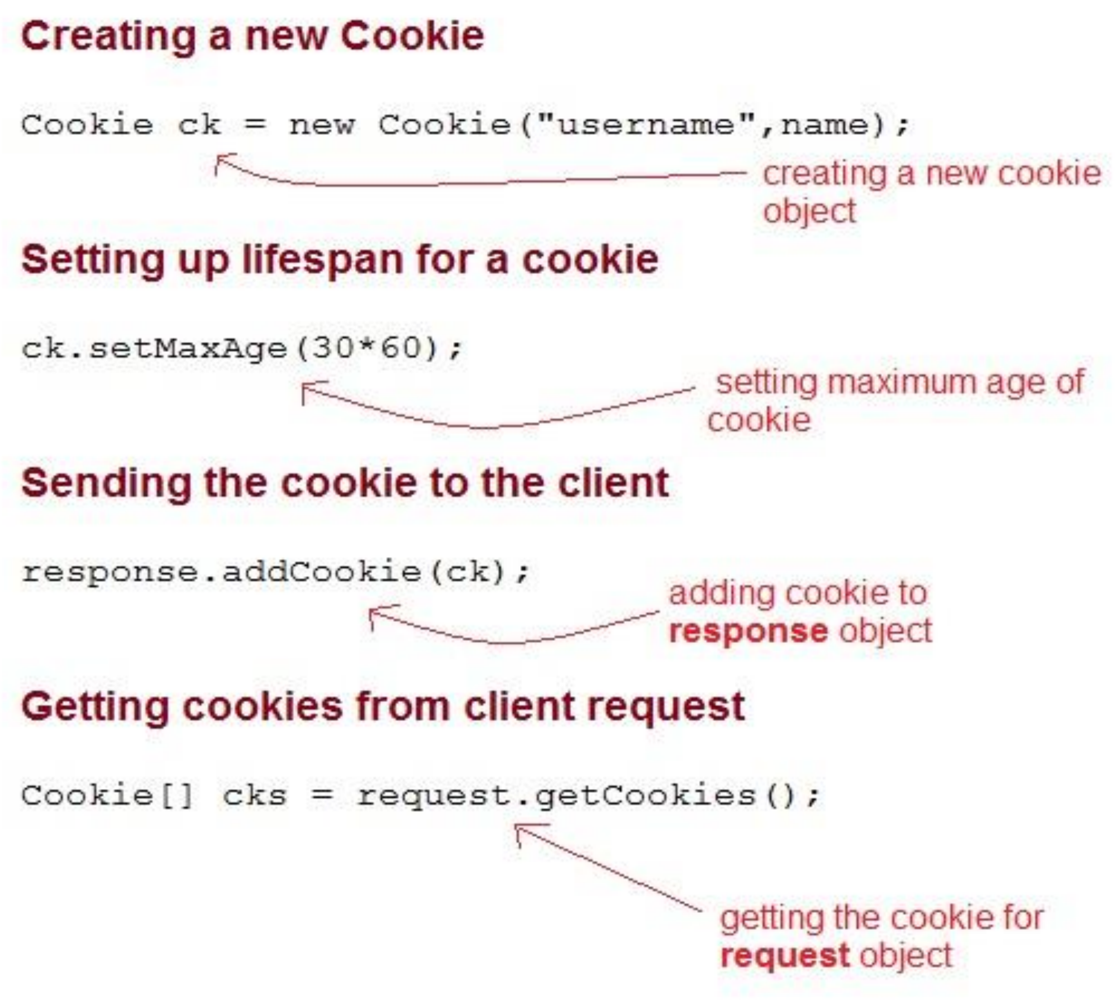
- **Cookies**
- **Database**
- **URL Rewriting**
- **HttpSession**
- **Hidden Form field**

Using Cookies for Session Management

- **Cookies** are small pieces of information that are sent in response from the web server to the client.
- **Cookies**are the simplest technique used for storing client state.
- **Cookies** are stored on client's computer.
- They have a lifespan and are destroyed by the client browser at the end of that lifespan.
- Using Cookies for storing client state has one shortcoming though, if the client has turned of Cookie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

Cookies API

- Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **response** object using the addCookie() method.
- This method sends cookie information over the HTTP response stream.
- getCookies() method is used to access the cookies that are added to response object.



Creating a new Cookie

```
Cookie ck = new Cookie("username",name);
```

creating a new cookie object

Setting up lifespan for a cookie

```
ck.setMaxAge(30*60);
```

setting maximum age of cookie

Sending the cookie to the client

```
response.addCookie(ck);
```

adding cookie to **response** object

Getting cookies from client request

```
Cookie[] cks = request.getCookies();
```

getting the cookie for **request** object

Hidden Form Field

- Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieved from another servlet.
- **Advantages :**
- Does not have to depend on browser whether the cookie is disabled or not.
- Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.
- **Disadvantage :**
- Extra form submission is required on every page. This is a big overhead.

index.html

```
<form method="post" action=“First">
  Name:<input type="text"  name="user" /><br/>
  Password:<input type="text"  name="pass" ><br/>
  <input type="submit" value="submit">
</form>
```

web.xml

```
<web-app...>
   <servlet>
      <servlet-name>First</servlet-name>
      <servlet-class>First</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>First</servlet-name>
      <url-pattern>/First</url-pattern>
   </servlet-mapping>
      <servlet>
      <servlet-name>Second</servlet-name>
      <servlet-class>Second</servlet-class>
   </servlet>
 <servlet-mapping>
      <servlet-
      name>Second</servlet-name>
      <url-pattern>/Second</url-
      pattern>
   </servlet-mapping>

   <welcome-file-list>
      <welcome-
      file>index.html</welcome-file>
   </welcome-file-list>

</web-app>
```

First.java

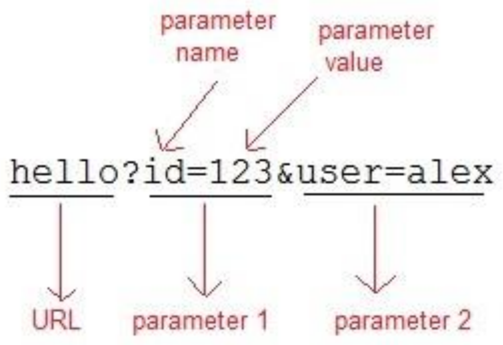
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class First extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
     //getting value submitted in form from HTML file
      String user = request.getParameter("user");
      //creating a new hidden form field
      out.println("<form action='Second'>");
      out.println("<input type='hidden' name='user' value='"+user+"'>");
      out.println("<input type='submit' value='submit' >");
      out.println("</form>");
   }
}
```

Second.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Second extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
            //getting parameter from the hidden field
        String user = request.getParameter("user");
        out.println("Welcome "+user);
    }
}
```

URL Rewriting

- If the client has disabled cookies in the browser then session management using cookie wont work.
- In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.
- In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.



- When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL.
- The **Web Container** will fetch the extra part of the requested URL and use it for session management.
- The getParameter() method is used to get the parameter value at the server side.

index.html

```
<form method="post" action="validate">
  Name:<input type="text"  name="user" /><br/>
  Password:<input type="text"  name="pass" ><br/>
  <input type="submit" value="submit">
</form>
```

web.xml

```
<web-app...>
   <servlet>
      <servlet-name>validate</servlet-name>
      <servlet-class>MyServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>validate</servlet-name>
      <url-pattern>/validate</url-pattern>
   </servlet-mapping>
      <servlet>
      <servlet-name>First</servlet-name>
      <servlet-class>First</servlet-class>
   </servlet>
```

```
<servlet-mapping>
     <servlet-name>First</servlet-name>
     <url-pattern>/First</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
     <welcome-file>index.html
    </welcome-file>
  </welcome-file-list>

</web-app>
```

MySerlet.java

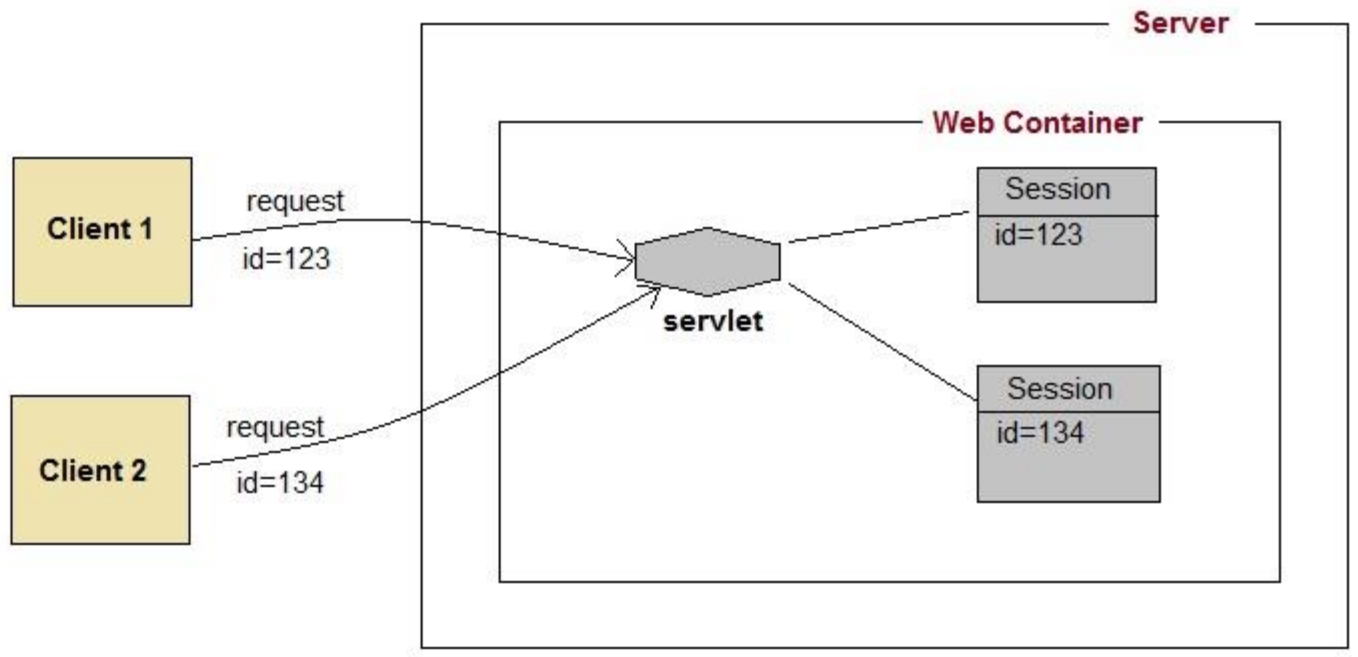
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet extends HttpServlet {
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      String name = request.getParameter("user");
      String pass = request.getParameter("pass");
      if(pass.equals("1234"))
      {
          response.sendRedirect("First?user_name="+name+"");
      }
   }
}
```

First.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class First extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse
      response)
          throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String user = request.getParameter("user_name");
        out.println("Welcome "+user);
    }
}
```

How Session Works

- Session is used to store everything that we can get from the client from all the requests the client makes.

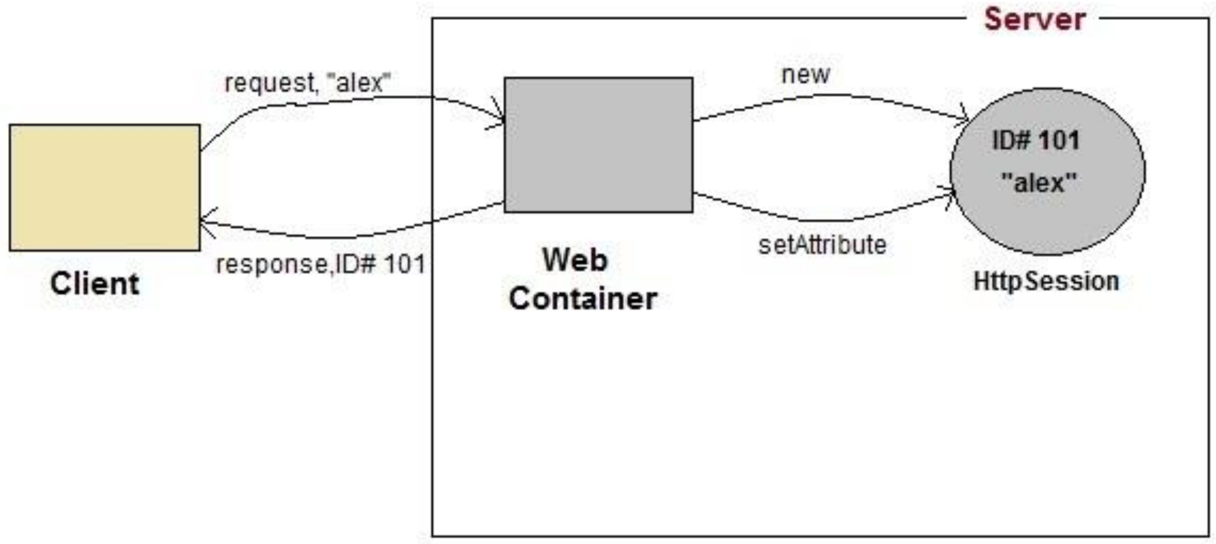


- The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed.
- So wherever the user goes, we will always have his information and we can always manage which user is doing what.
- Whenever a user wants to exit from your application, destroy the object with his information.

What is HttpSession?

- **HttpSession** object is used to store entire session with a specific client.
- We can store, retrieve and remove attribute from **HttpSession** object.
- Any servlet can have access to **HttpSession** object throughout the **getSession()** method of the **HttpServletRequest** object.

How HttpSession works

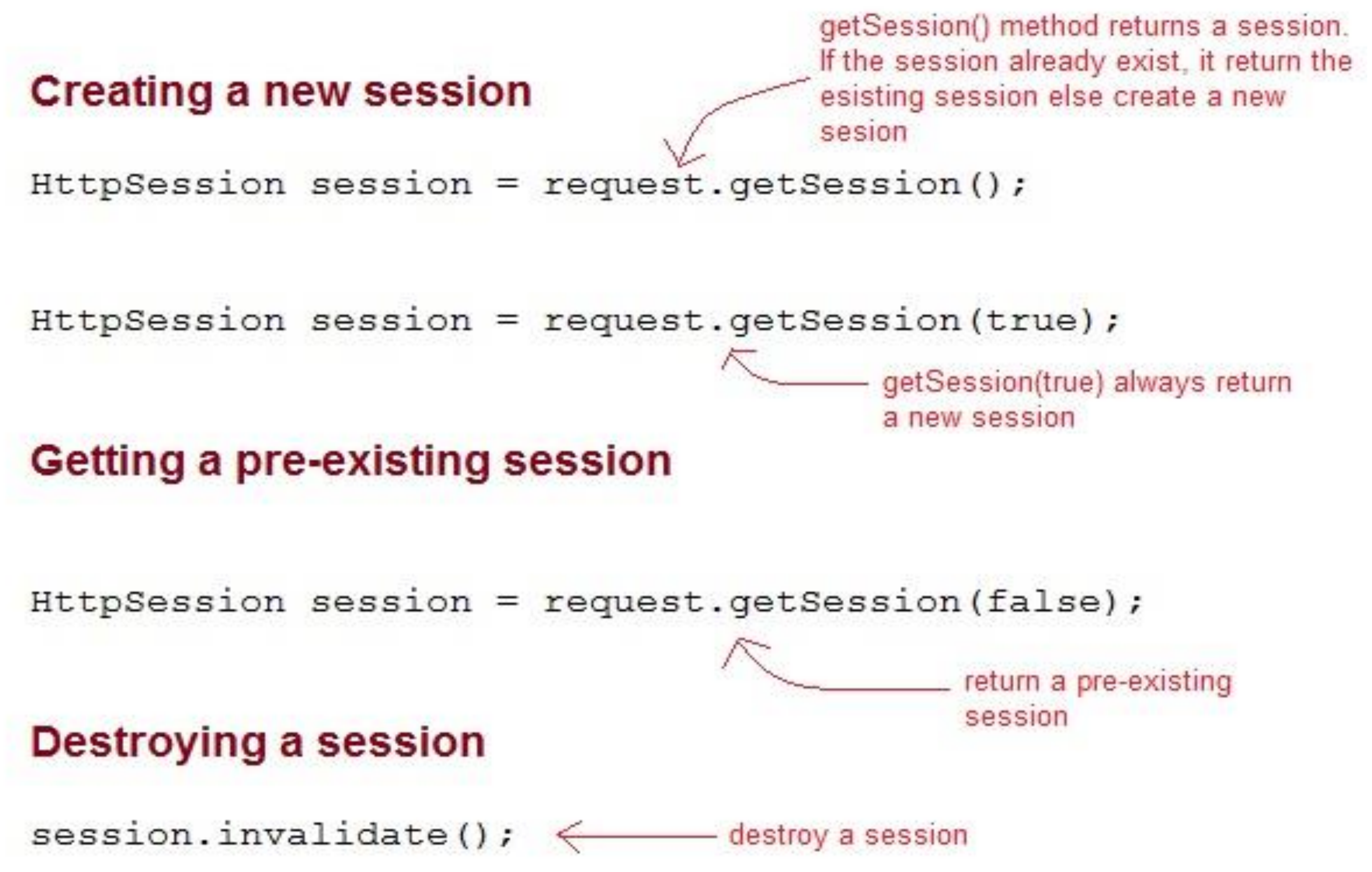


On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.

The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.

The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

HttpSession Interface



Creating a new session

getSession() method returns a session. If the session already exist, it return the esisting session else create a new sesion

```
HttpSession session = request.getSession();
```

```
HttpSession session = request.getSession(true);
```

getSession(true) always return a new session

Getting a pre-existing session

```
HttpSession session = request.getSession(false);
```

return a pre-existing session

Destroying a session

```
session.invalidate();
```

destroy a session

Index.html

```
<html>
  <head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <form action="s1" method="post">
      <input type="submit" value="submit"/>
    </form>
  </body>
</html>
```

S1.java

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
      HttpSession session = request.getSession();
      session.setAttribute("user","CDAC");
      response.sendRedirect("s2");
    }
  }
```

S2.java

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
  response.setContentType("text/html;charset=UTF-8");
  try (PrintWriter out = response.getWriter()) {
      /* TODO output your page here. You may use following sample
    code. */
      HttpSession session = request.getSession(false);
      out.println(session.getAttribute("user"));
  }
}
```

Thank You