JAVA SERVLETS

By
Sudha Agarwal

INDEX

- Servlet Introduction

SENDREDIRECT( ) METHOD

❖sendRedirect() method redirects the response to another resource. This method actually makes the client(browser) to create a new request to get to the resource. The client can see the new url in the browser.

❖sendRedirect() accepts relative URL, so it can go for resources inside or outside the server.

SENDREDIRECT() VS. REQUEST DISPATCHER

❖The main difference between a redirection and a request dispatching is that, redirection makes the client(browser) create a new request to get to the resource, the user can see the new URL while request dispatch get the resource in same request and URL does not changes.

❖Also, another very important difference is that, sendRedirect() works on response object while request dispatch work on request object.

SERVLETCONFIG INTERFACE

❖When the Web Container initializes a servlet, it creates a ServletConfig object for the servlet.

❖ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from web.xml(Deployment Descriptor).

METHODS OF SERVLETCONFIG

❖**String getInitParameter(String name):** returns a String value initialized parameter, or NULL if the parameter does not exist.

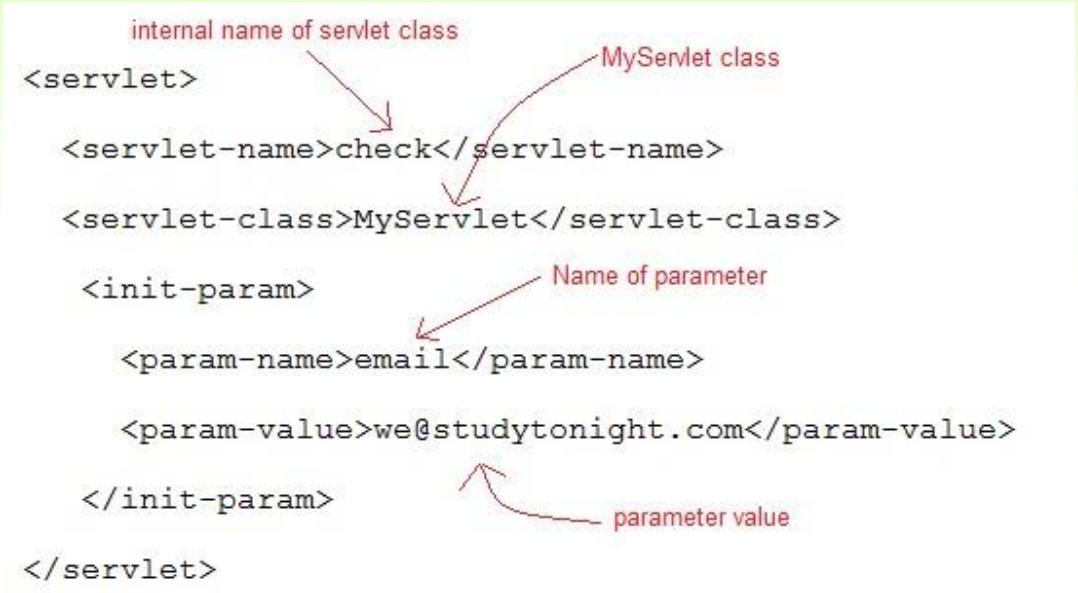
❖**Enumeration getInitParameterNames():** returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.

❖**ServletContext getServletContext():** returns a reference to the ServletContext

❖**String getServletName():** returns the name of the servlet instance

SERVLETCONFIG

❖In the Deployment Descriptor(web.xml) file,




```
<servlet>
  <servlet-name>check</servlet-name>
  <servlet-class>MyServlet</servlet-class>
   <init-param>
     <param-name>email</param-name>
     <param-value>we@studytonight.com</param-value>
   </init-param>
</servlet>
```

❖Inside the Servlet class, using following code,

```
ServletConfig sc = getServletConfig();
out.println(sc.getInitParameter("email"));
```

SERVLETCONTEXT INTERFACE

❖For every Web application a ServletContext object is created by the web container. ServletContext object is used to get configuration information from Deployment Descriptor(web.xml) which will be available to any servlet or JSPs that are part of the web app.

METHOD OF SERVLETCONTEXT

❖**Object getAttribute(String name):** returns the container attribute with the given name, or NULL if there is no attribute by that name.

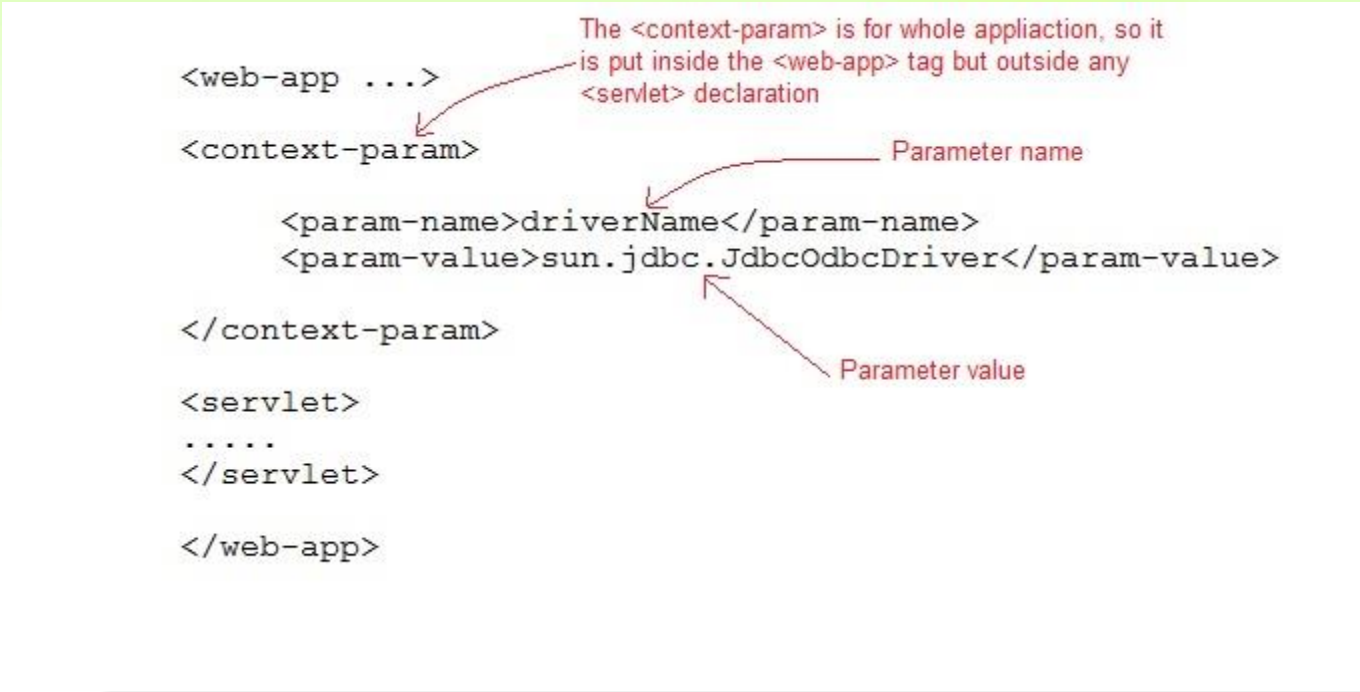
❖**String getInitParameter(String name):** returns parameter value for the specified parameter name, or NULL if the parameter does not exist

❖**Enumeration getInitParameterNames() :** returns the names of the context's initialization parameters as an Enumeration of String objects

❖**void setAttribute(String name,Object obj) :** set an object with the given attribute name in the application scope

❖**void removeAttribute(String name):** removes the attribute with the specified name from the application context

CONTEXT PARAMETER IN WEB.XML




```
<web-app ...>

<context-param>

     <param-name>driverName</param-name>
     <param-value>sun.jdbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet>
.....
</servlet>

</web-app>
```

```
ServletContext app = getServletContext();
OR
ServletContext app = getServletConfig().getServletContext();
```

Advantages of ServletContext

1. Provides communication between servlets
2. Available to all servlets and JSPs that are part of the web app
3. Used to get configuration information from web.xml

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 SESSION 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.

❖**Session Management** is a mechanism used by the Web container to store session information for a particular user.

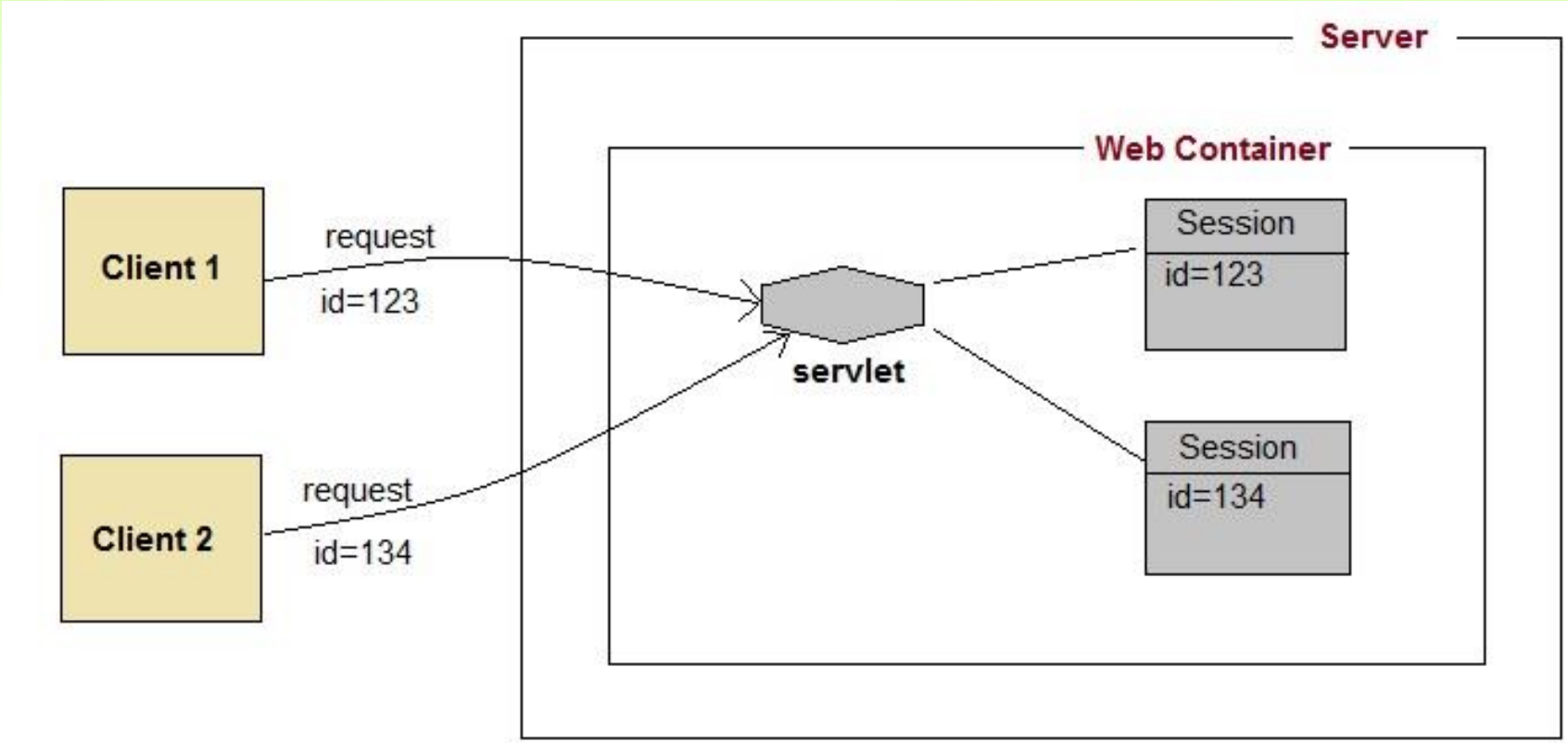
MANAGING SESSION IN SERVLETS

❖There are four different techniques used by Servlet application for session management. They are as follows:

1. Cookies
2. Hidden form field
3. URL Rewriting
4. HttpSession

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

How Session Works



How Session Works

❖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.

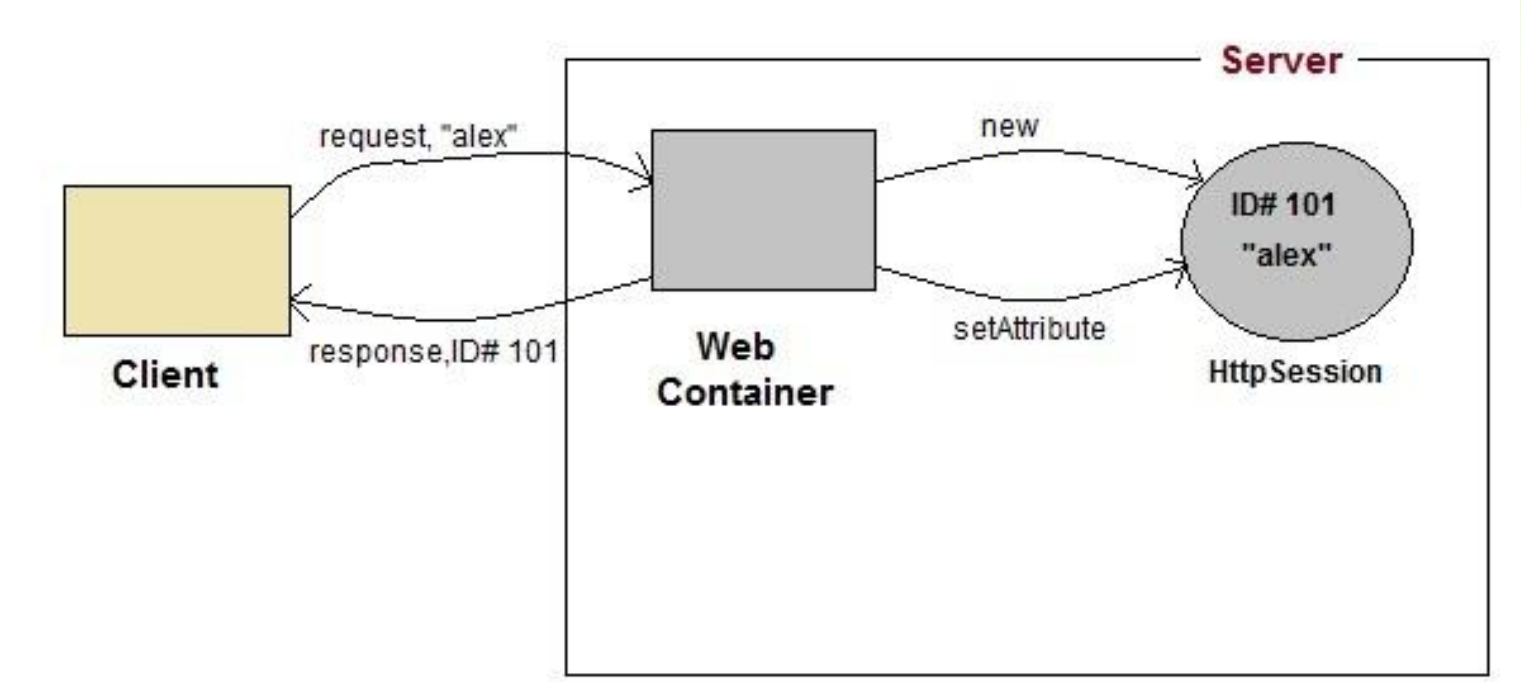
❖So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

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



How HttpSession Works

1. 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.

2. 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.

3. 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

METHODS OF HTTPSESSION

- **long getCreationTime()** returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.

- **String getId()** returns a string containing the unique identifier assigned to the session.

- **long getLastAccessedTime()** returns the last time the client sent a request associated with the session

- **int getMaxInactiveInterval()** returns the maximum time interval, in seconds.

- **void invalidate()** destroy the session

- **boolean isNew()** returns true if the session is new else false

- **void setMaxInactiveInterval(int interval)** Specifies the time, in seconds,after servlet container will invalidate the session.

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.

COOKIES API

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);
```

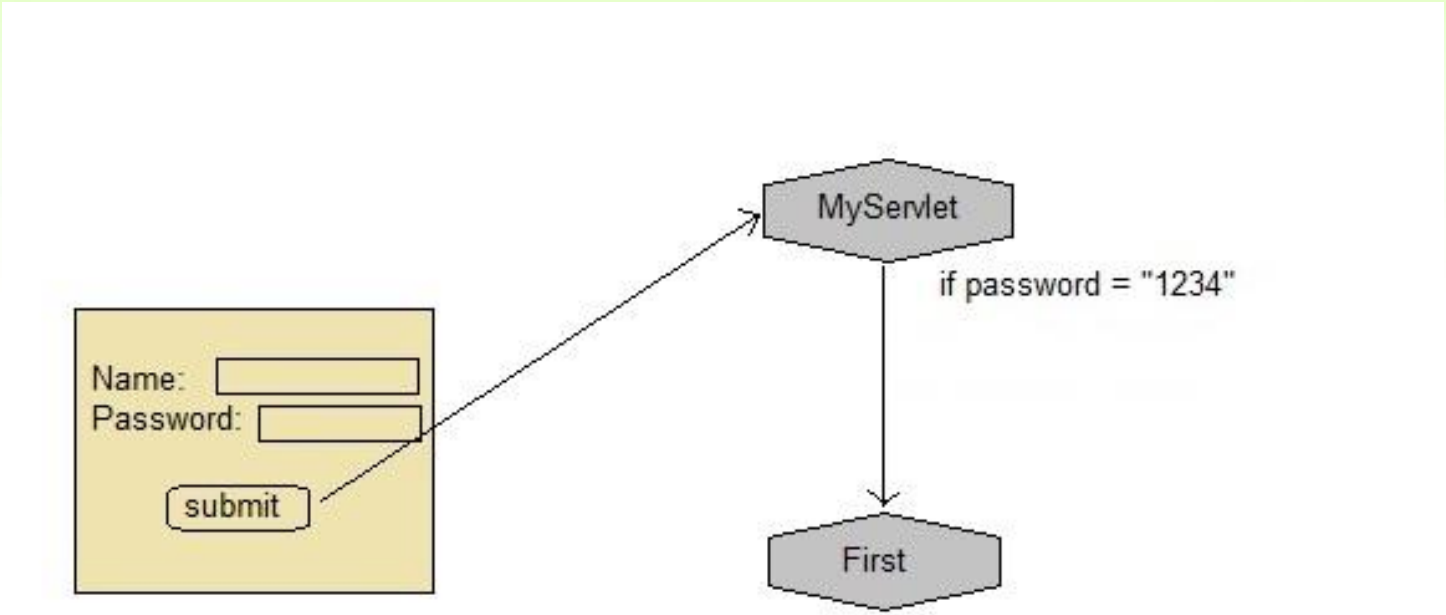
adding cookie to **response** object

Getting cookies from client request

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

getting the cookie for **request** object

COOKIES



URL REWRITING

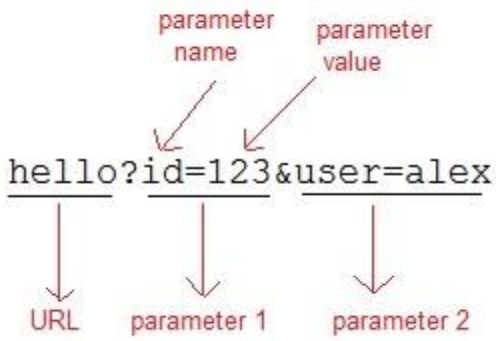
❖If the client has disabled cookies in the browser then session management using cookie wont work. In that case URL Rewriting can be used as a backup. URL rewriting will always work.

❖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.

URL REWRITING

❖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.



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.

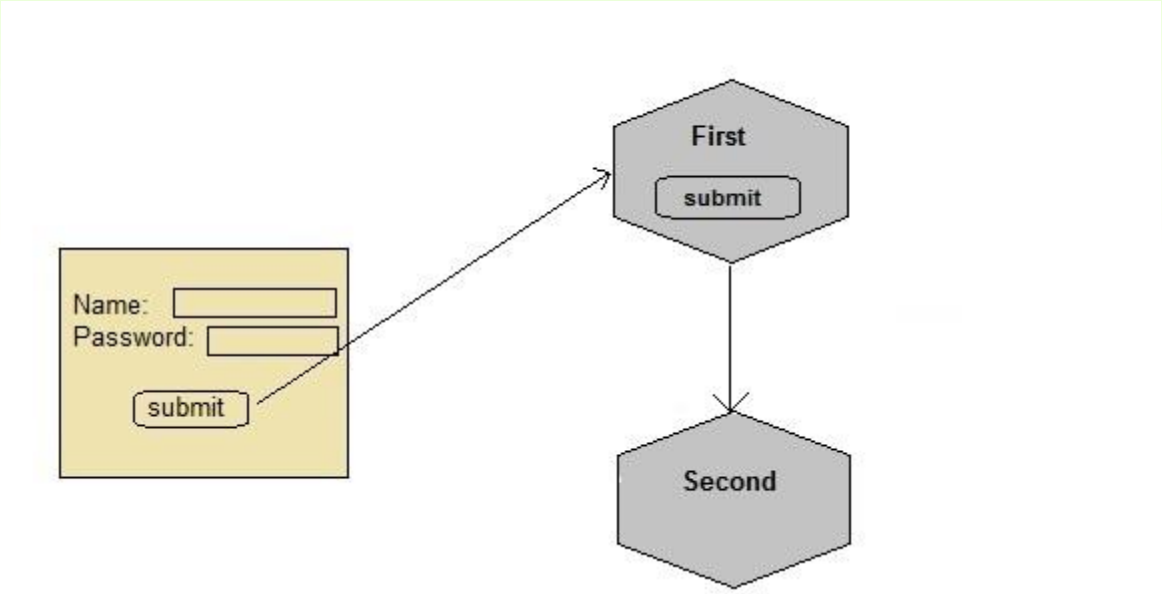
❖Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.

❖Disadvantage :

❖Extra form submission is required on every page. This is a big overhead.

❖it’s not secure because we can get the hidden field value from the HTML source and use it to hack the session.

HIDDEN FORM FIELD



EXCEPTION HANDLING

❖If you notice, doGet() and doPost() methods throw javax.servlet.ServletException and IOException.

❖When we throw these exception from our application, we get response like below image.

EXCEPTION HANDLING

❖Since browser understand only HTML, when our application throw exception, servlet container processes the exception and generate a HTML response.

❖The problem with this response is that it’s of no value to user. Also it’s showing our application classes and server details to user that makes no sense to user and it’s not good from security point of view.

EXCEPTION HANDLING

❖Servlet API provides support for custom Exception and Error Handler servlets that we can configure in deployment descriptor. The whole purpose of these servlets are to handle the Exception or Error raised by application and send useful HTML response to user. We can provide link to application home page or some details to let user know what went wrong.

❖So first of all we need to create a custom Exception and Error Handler servlet.

EXCEPTION HANDLING

❖When a servlet throws an exception, the web container searches the configurations in web.xml that use the exception-type element for a match with the thrown exception type.

❖You would have to use the error-page element in web.xml to specify the invocation of servlets in response to certain exceptions or HTTP status codes.

WEB.XML CONFIGURATION

```
<!-- servlet definition -->
<servlet>
   <servlet-name>ErrorHandler</servlet-name>
   <servlet-class>ErrorHandler</servlet-class>
</servlet>

<!-- servlet mappings -->
<servlet-mapping>
   <servlet-name>ErrorHandler</servlet-name>
   <url-pattern>/ErrorHandler</url-pattern>
</servlet-mapping>

<!-- error-code related error pages -->
<error-page>
   <error-code>404</error-code>
   <location>/ErrorHandler</location>
</error-page>

<error-page>
   <error-code>403</error-code>
   <location>/ErrorHandler</location>
</error-page>

<!-- exception-type related error pages -->
<error-page>
   <exception-type>
      javax.servlet.ServletException
   </exception-type >
   <location>/ErrorHandler</location>
</error-page>

<error-page>
   <exception-type>java.io.IOException</exception-type >
   <location>/ErrorHandler</location>
</error-page>
```

EXCEPTION HANDLING

❖Following are the points to be noted about above web.xml for Exception Handling −

❖The servlet ErrorHandler is defined in usual way as any other servlet and configured in web.xml.

❖If there is any error with status code either 404 (Not Found) or 403 (Forbidden), then ErrorHandler servlet would be called.

❖If the web application throws either ServletException or IOException, then the web container invokes the /ErrorHandler servlet.

❖You can define different Error Handlers to handle different type of errors or exceptions.

SERVLET ANNOTATION

❖Servlet API 3.0 has introduced a new package called javax.servlet.annotation. It provides annotation types which can be used for annotating a servlet class. If you use annotation, then the deployment descriptor (web.xml) is not required. But you should use tomcat7 or any later version of tomcat.

❖Annotations can replace equivalent XML configuration in the web deployment descriptor file (web.xml) such as servlet declaration and servlet mapping. Servlet containers will process the annotated classes at deployment time.

Annotation Types

- **@WebServlet** : To declare a servlet.
- **@WebInitParam** : To specify an initialization parameter.
- **@WebFilter** : To declare a servlet filter.
- **@WebListener** : To declare a WebListener
- **@HandlesTypes :** To declare the class types that a ServletContainerInitializer can handle.
- **@HttpConstraint** : This annotation is used within the ServletSecurity annotation to represent the security constraints to be applied to all HTTP protocol methods for which a corresponding HttpMethodConstraint element does NOT occur within the ServletSecurity annotation.

Annotation Types

❖**@HttpMethodConstraint :** This annotation is used within the ServletSecurity annotation to represent security constraints on specific HTTP protocol messages.

❖**@MultipartConfig :** Annotation that may be specified on a Servlet class, indicating that instances of the Servlet expect requests that conform to the multipart/form-data MIME type.

❖**@ServletSecurity :** This annotation is used on a Servlet implementation class to specify security constraints to be enforced by a Servlet container on HTTP protocol messages.