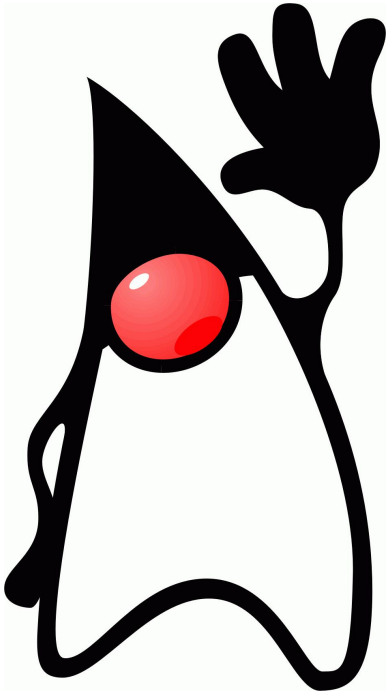
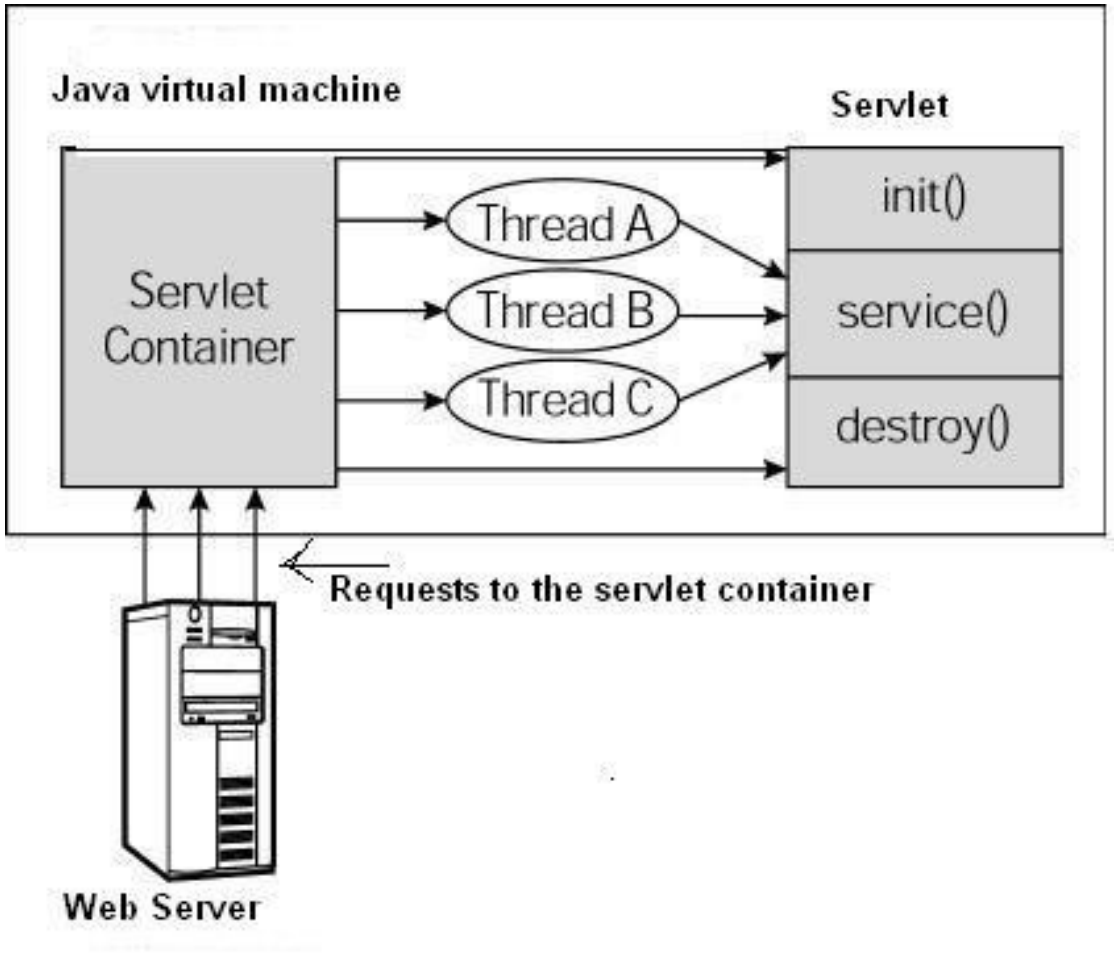
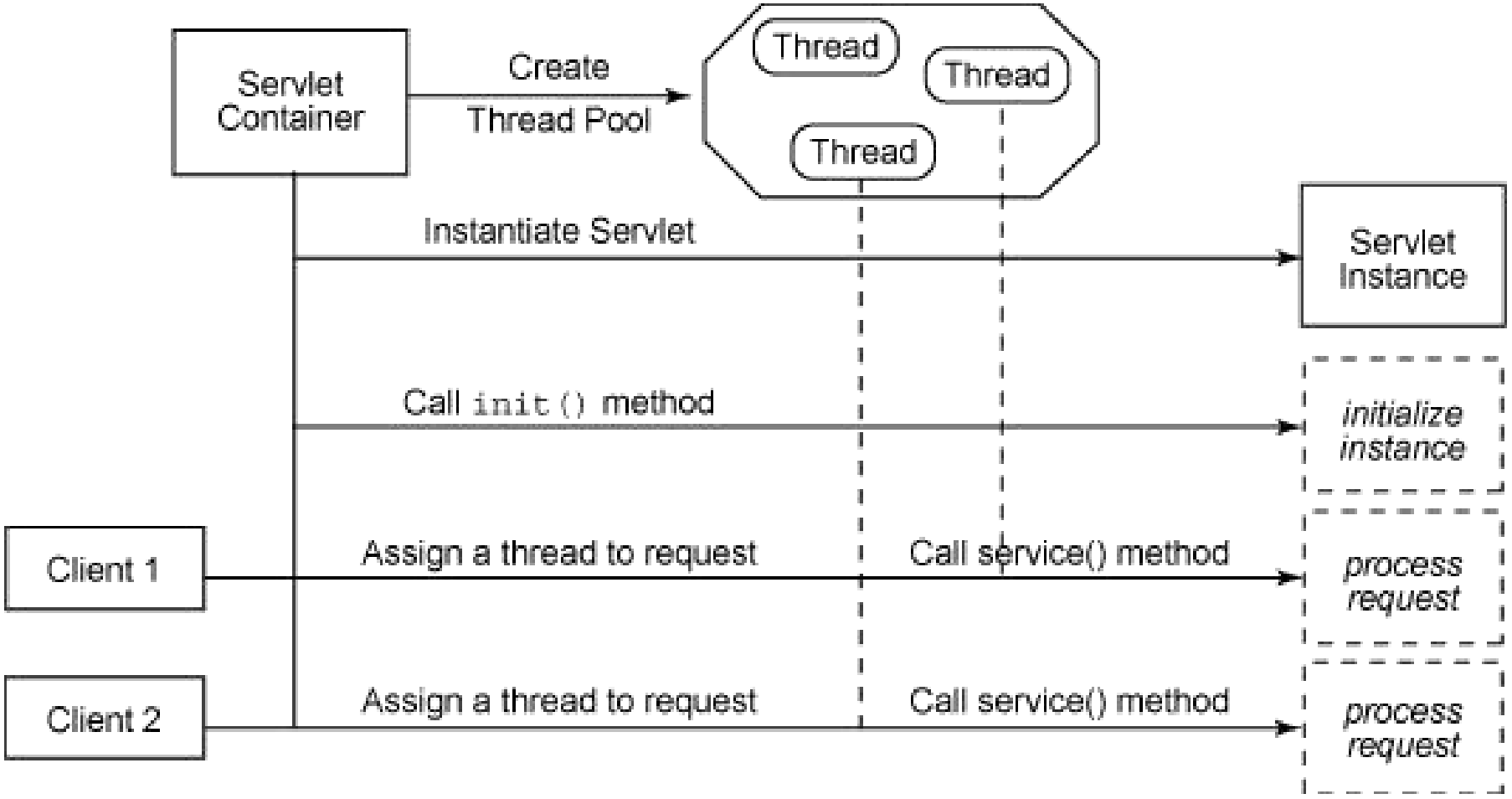
Part-2



1. First the HTTP requests coming to the server are delegated to the servlet container.
2. The servlet container loads the servlet before invoking the service() method.
3. Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet




Servlet Container
Create Thread Pool
Thread
Thread
Thread
Instantiate Servlet
Servlet Instance
Call init() method
initialize instance
Client 1
Assign a thread to request
Call service() method
process request
Client 2
Assign a thread to request
Call service() method
process request

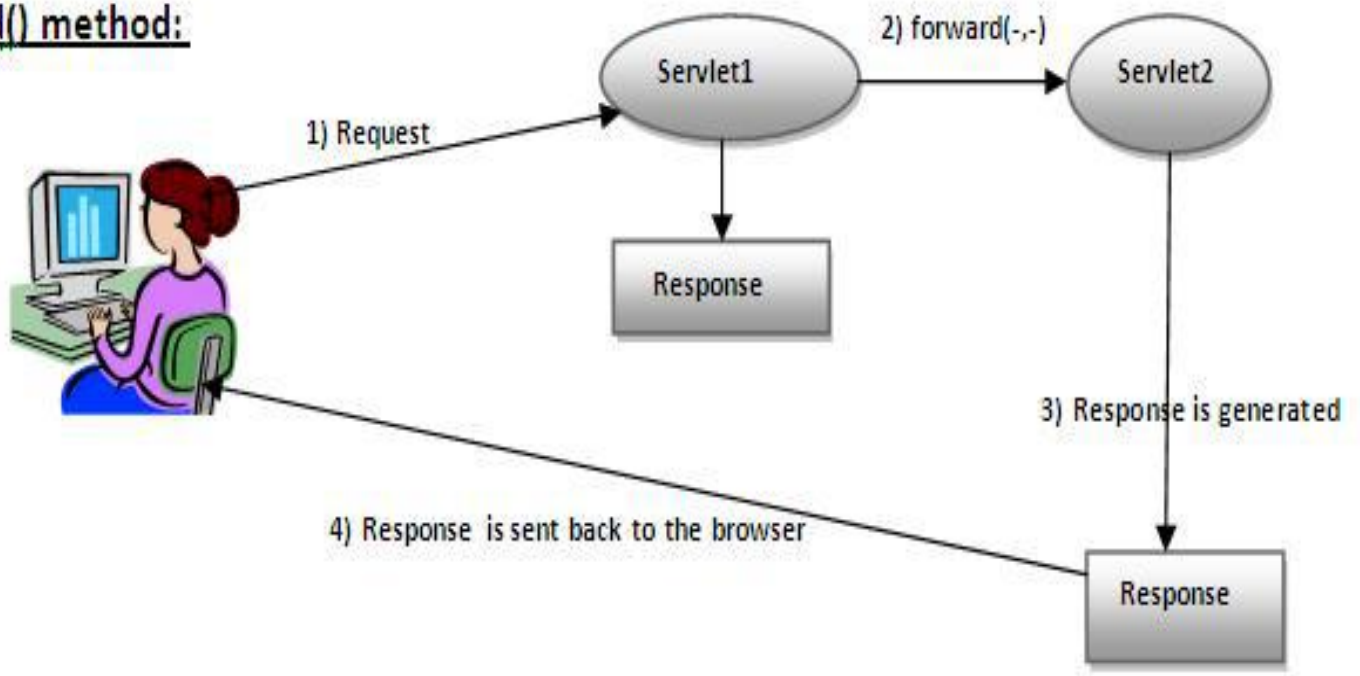
RequestDispatcher

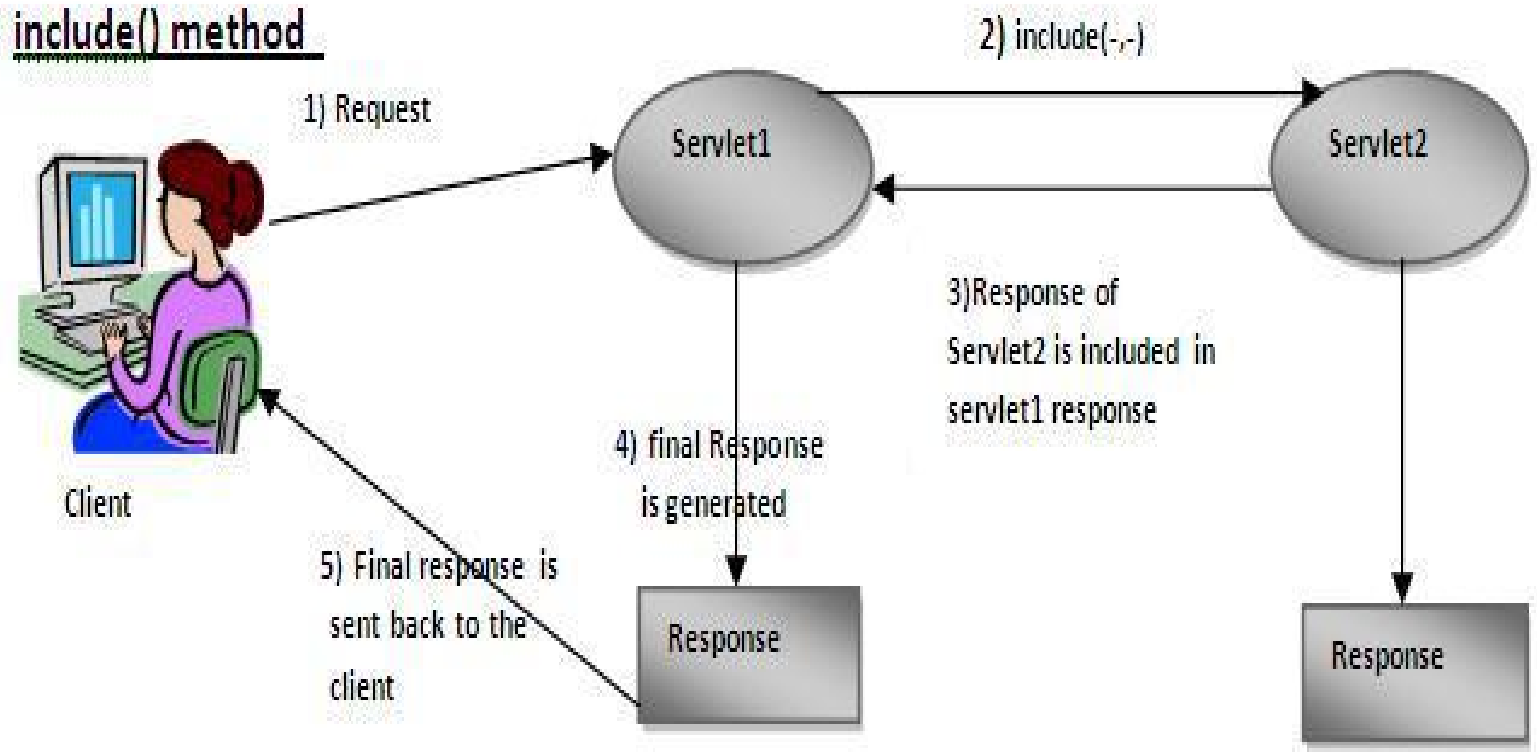
- The **RequestDispacher** interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
- This interface can also be used to include the content of another resource also.

The RequestDispatcher interface provides two methods. They are:

- ***public void forward(ServletRequest request,ServletResponse response)throws ServletException, java.io.IOException:*** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- ***public void include(ServletRequest request,ServletResponse response)throws ServletException, java.io.IOException:*** Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

forward() method:




include() method
1) Request
2) include(-,-)
Servlet1
Servlet2
3)Response of
Servlet2 is included in
servlet1 response
Client
4) final Response
is generated
5) Final response is
sent back to the
client
Response
Response

SendRedirect in servlet

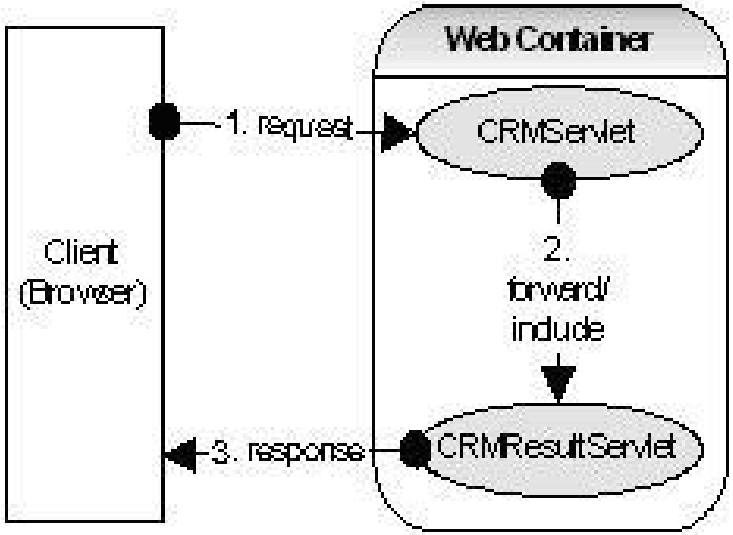
- The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

- It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

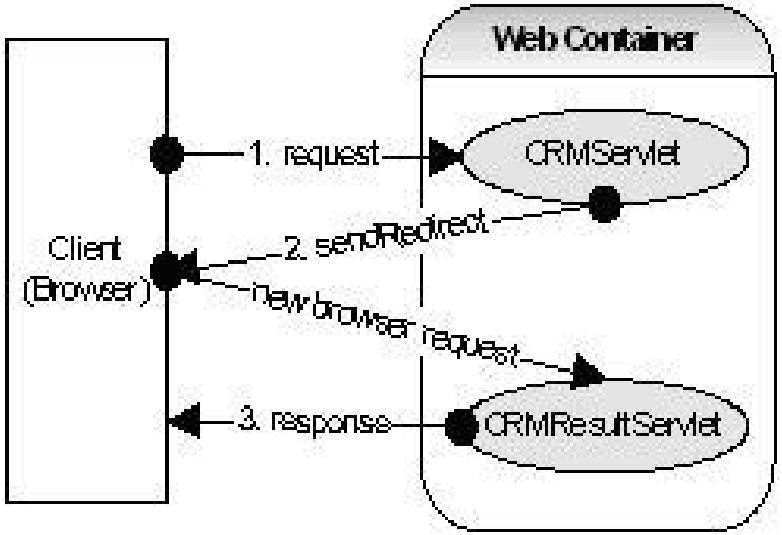
forward() or include() vs sendRedirect()

forward() or include()



Note: path supplied to RequestDispatcher will be something like "/CRMResultServlet"

sendRedirect()



Note: path supplied to RequestDispatcher will be something like "http://myserver:8080/myContext/CRMResultServlet".

ServletConfig Interface

- An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.
- The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

- Methods of ServletConfig interface
  - **public String getInitParameter(String name):** Returns the parameter value for the specified parameter name.
  - **public Enumeration getInitParameterNames():** Returns an enumeration of all the initialization parameter names.
  - **public String getServletName():** Returns the name of the servlet.
  - **public ServletContext getServletContext():** Returns an object of ServletContext.

ServletContext Interface

- An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file.
- There is only one ServletContext object per web application.
- If any information is shared to many servlets, it is better to provide it from the web.xml file using the**<context-param>** element.

Advantage of ServletContext

- **Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlets. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet.

Example of getServletContext() method

//We can get the ServletContext object from ServletConfig object

- ServletContext application = getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object

- ServletContext application = getServletContext();

Scope

An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

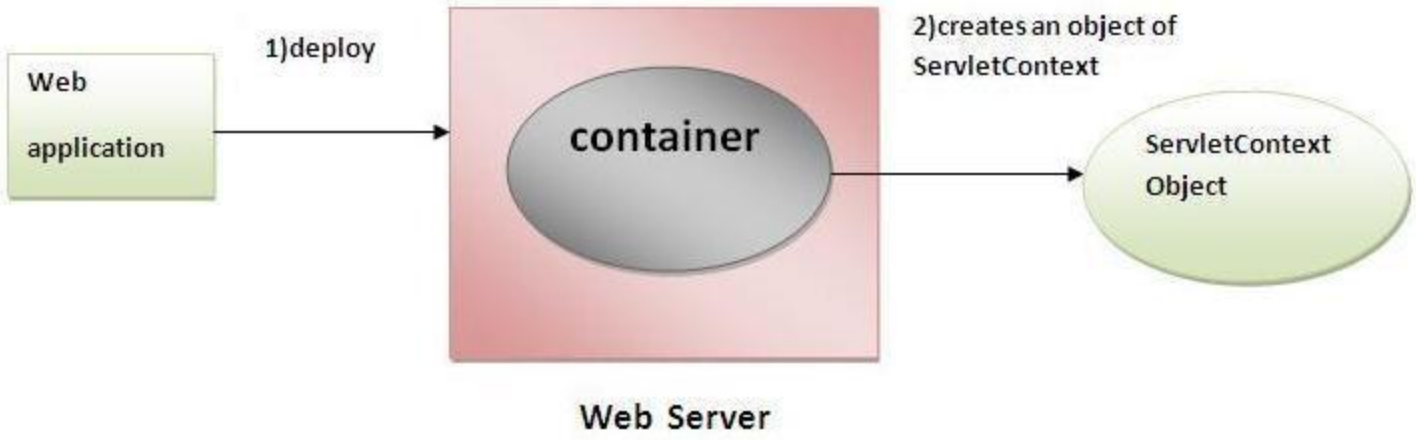
- request scope
- session scope
- application scope

- The servlet programmer can pass information from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

Usage of ServletContext

Usage of ServletContext Interface

- The object of ServletContext provides an interface between the container and servlet.
- The ServletContext object can be used to get configuration information from the web.xml file.
- The ServletContext object can be used to set, get or remove attribute from the web.xml file.
- The ServletContext object can be used to provide inter-application communication.



Difference between ServletConfig and ServletContext

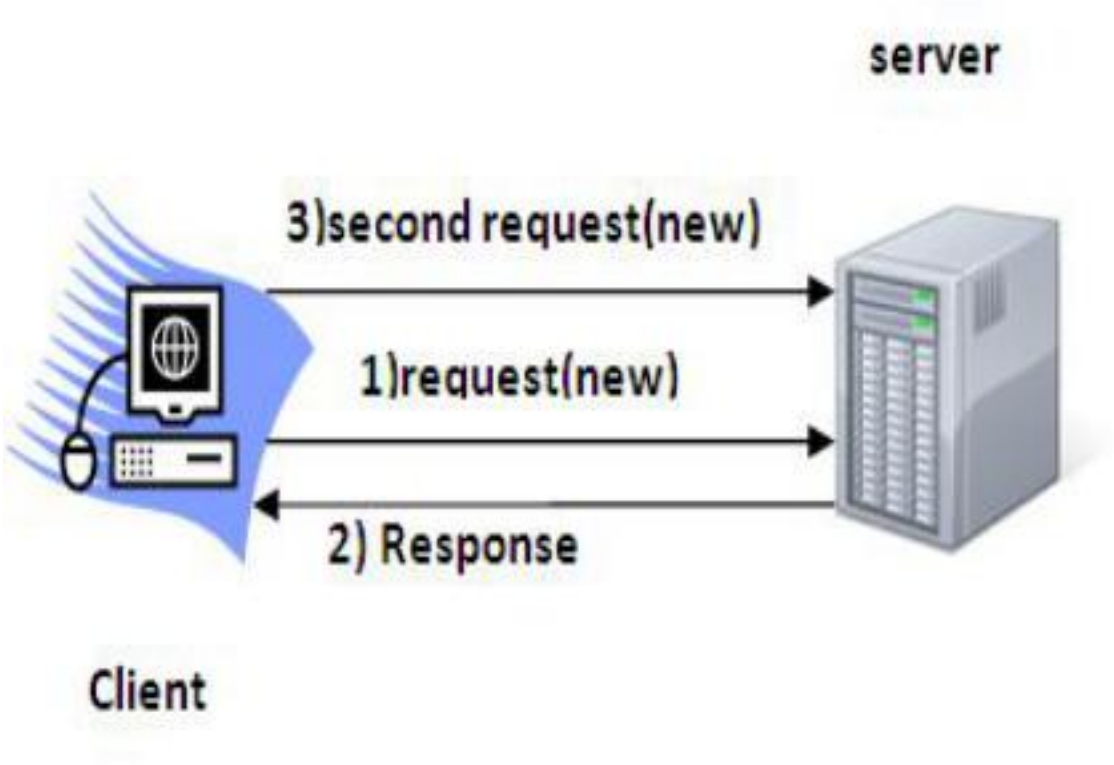
The **servletconfig** object refers to the single servlet whereas **servletcontext** object refers to the whole web application.

Session Tracking in Servlets

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

Why use Session Tracking?

It is used to **recognize** the particular user.



Session Tracking Techniques

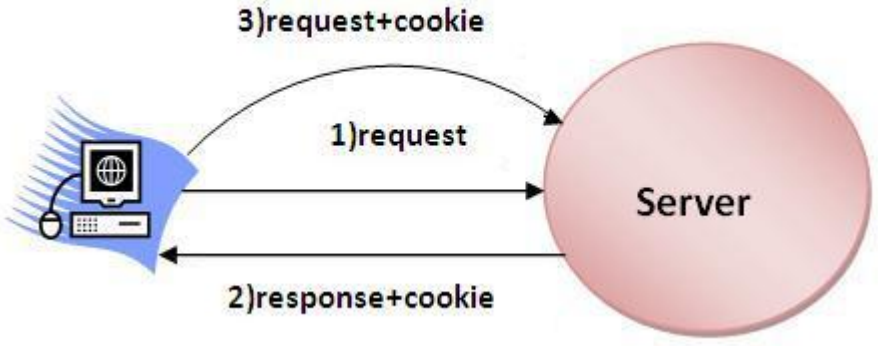
- There are four techniques used in Session tracking:
  - **Cookies**
  - **Hidden Form Field**
  - **URL Rewriting**
  - **HttpSession**

Cookies in Servlet

- A **cookie** is a kind of information that is stored at client side.
- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works?

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet.
- So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user



Cookies in Servlet

Types of Cookie

- There are 2 types of cookies in servlets.
  - Non-persistent cookie
  - Persistent cookie
- **Non-persistent cookie**
  - It is **valid for single session** only. It is removed each time when user closes the browser.
- **Persistent cookie**
  - It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

Cookie class

- **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.
public void addCookie(Cookie ck)	method of HttpServletResponse interface is used to add cookie in response object
public Cookie[] getCookies()	method of HttpServletRequest interface is used to return all the cookies from the browser

Hidden Form Field

- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet.
- This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Example:

<input type="hidden" name="uname" value="CDAC">

Advantage of Hidden Form Field

- It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.

URL Rewriting

- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:

Example:

url?name1=value1&name2=value2&??

- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

Advantage of URL Rewriting

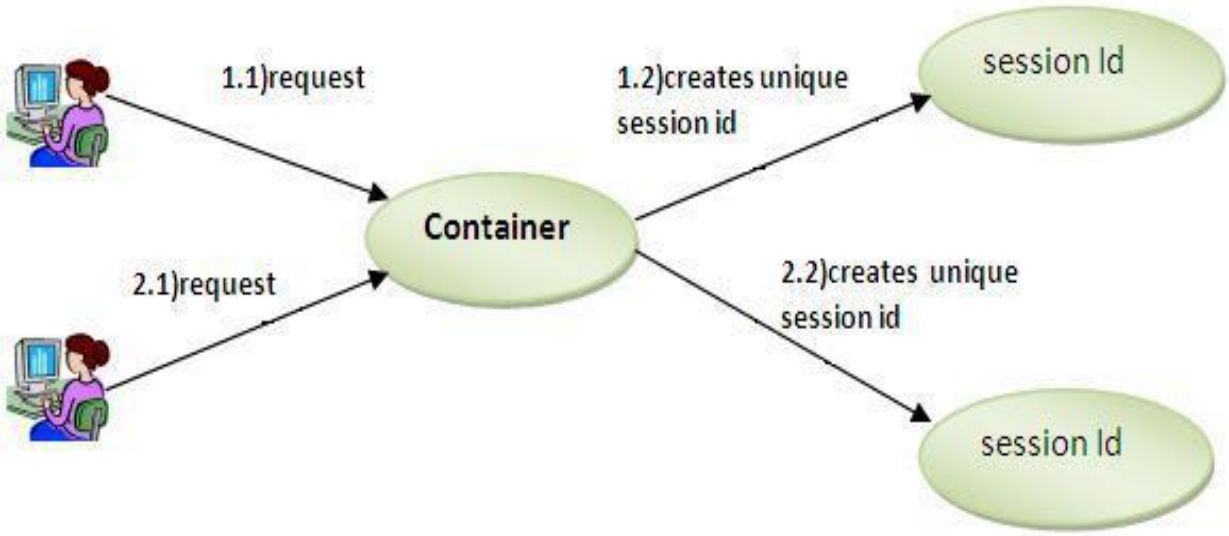
- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

- It will work only with links.
- It can send Only textual information.

HttpSession interface

- In such a case, container creates a session id for each user.
- The container uses this id to identify the particular user.
- An object of HttpSession can be used to perform two tasks:
  - bind objects
  - view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

- The HttpServletRequest interface provides two methods to get the object of HttpSession:
  - **public HttpSession getSession():** Returns the current session associated with this request, or if the request does not have a session, creates one.
  - **public HttpSession getSession(boolean create):** Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

- **public String getId():** Returns a string containing the unique identifier value.
- **public long getCreationTime():** Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
- **public long getLastAccessedTime():** Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
- **public void invalidate():** Invalidates this session then unbinds any objects bound to it.

Creating a new session

Returns the current session associated with this request, or if the request does not have a session, creates one.

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

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

Returns the current HttpSession associated with this request (or),
if there is no current session
and create is true, returns a new session.

Getting a pre-existing session

and create is false, returns null.

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

Destroying a session

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

destroy a session