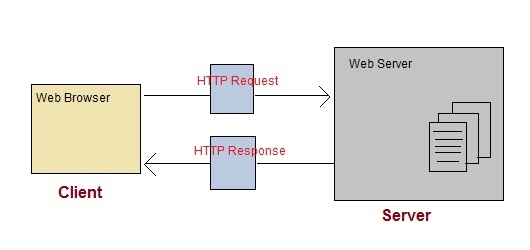
**Web-apps and Servlets**

**Web Application**

A website is a collection of static files (webpages) such as HTML pages, images, graphics etc. A **Web application** is a web site with dynamic functionality on the server. **Google**, **Facebook**, **Twitter** are examples of web applications.

#### HTTP (Hypertext Transfer Protocol)

* HTTP is a protocol that clients and servers use on the web to communicate.
* It is similar to other internet protocols such as SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) but there is one fundamental difference.
* HTTP is a **stateless protocol** i.e. HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
* The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.

## HTTP Request Methods

The following table lists some other HTTP request methods:

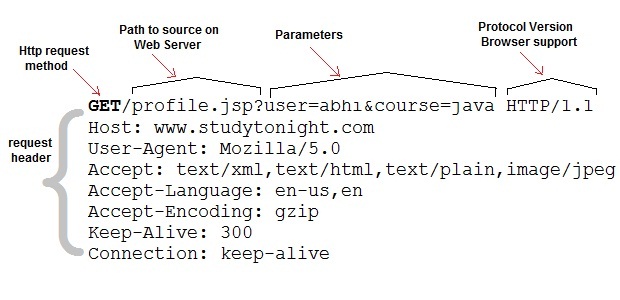
|  |  |
| --- | --- |
| Method | Description |
| **GET** | Requests data from a specified resource or from server using a given URL. |
| **POST** | Submits data to be processed to a specified resource |
| Request for server to accept the entity enclosed in the body of HTTP method. |
| **HEAD** | Same as GET but returns only HTTP headers and no document body |
| **PUT** | Uploads a representation of the specified URL |
| It replaces all current representations of the target resource with the uploaded content. |
| **DELETE** | Request for the Server to deletes the specified resource |
| **OPTIONS** | Returns the HTTP methods that the server supports |
| **CONNECT** | Converts the request connection to a transparent TCP/IP tunnel |

**Difference between GET and POST requests**

|  |  |
| --- | --- |
| **GET Request** | **POST Request** |
| Data is sent in header to the server | Data is sent in the request body |
| Get request can send only limited amount of data | Large amount of data can be sent. |
| Get request is not secured because data is exposed in URL | Post request is secured because data is not exposed in URL. |
| Get request can be bookmarked and is more efficient. | Post request cannot be bookmarked. |

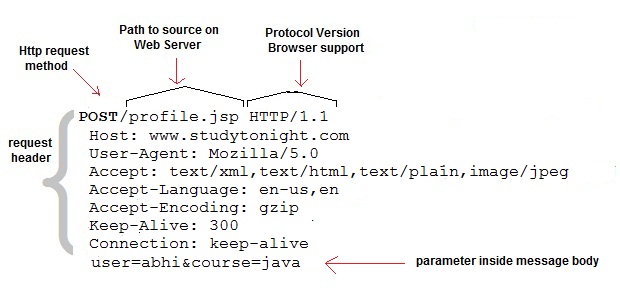
**Anatomy of an HTTP GET request**

Get request contains path to server and the parameters added to it.



**Anatomy of an HTTP POST request**

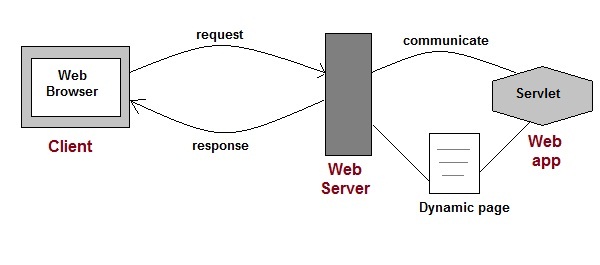
Post requests are used to make more complex requests on the server. For instance, if a user has filled a form with multiple fields and the application wants to save all the form data to the database. Then the form data will be sent to the server in POST request body, which is also known as Message body.



### Introduction to Servlet

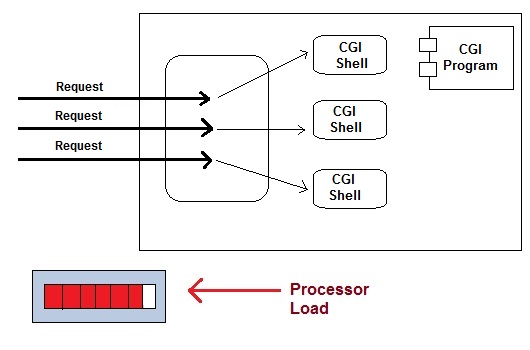
**Servlet** Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are a helper application that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.



As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

#### CGI (Common Gateway Interface)

Before Servlets, CGI (Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works:

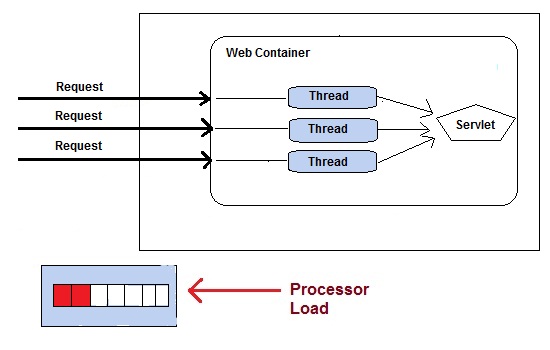
* User clicks a link that has URL to a dynamic page instead of a static page.
* The URL decides which CGI program to execute.
* Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
* The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.

#### Drawbacks of CGI programs

* High response time because CGI programs execute in their own OS shell.
* CGI is not scalable.
* CGI programs are not always secure or object-oriented.
* It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

#### Advantages of using Servlets

* Less response time because each request runs in a separate thread.
* Servlets are scalable.
* Servlets are robust and object oriented.
* Servlets are platform independent.

### Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely:

* **javax.servlet**
* **javax.servlet.http**

#### Some Important Classes and Interface of javax.servlet.http

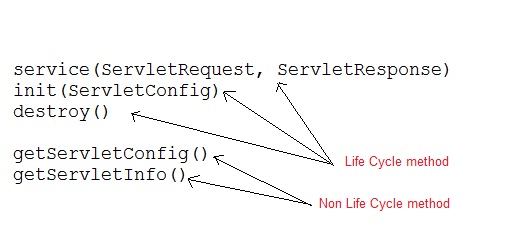
|  |  |
| --- | --- |
| **CLASSES and INTERFACES** | |
| HttpServlet | HttpServletRequest |
| HttpServletResponse | HttpSessionAttributeListener |
| HttpSession | HttpSessionListener |
| Cookie | HttpSessionEvent |

#### Some Important Classes and Interfaces of javax.servlet

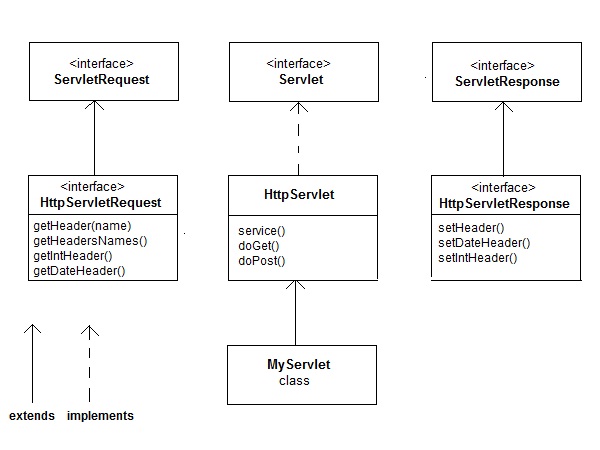
|  |  |
| --- | --- |
| **INTERFACES** | **CLASSES** |
| Servlet | ServletInputStream |
| ServletContext | ServletOutputStream |
| ServletConfig | ServletRequestWrapper |
| ServletRequest | ServletResponseWrapper |
| ServletResponse | ServletRequestEvent |
| ServletContextListener | ServletContextEvent |
| RequestDispatcher | ServletRequestAttributeEvent |
| SingleThreadModel | ServletContextAttributeEvent |
| Filter | ServletException |
| FilterConfig | UnavailableException |
| FilterChain | GenericServlet |
| ServletRequestListener |  |

#### Servlet Interface

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non-life cycle methods.



#### HttpServlet class

HttpServlet is also an abstract class. This class gives implementation of various service() methods of **Servlet** interface.

To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, **must not override service()** method. Our servlet class will override only the doGet()and/or doPost() methods.

The service() method of **HttpServlet** class **listens** to the Http methods (GET, POST etc.) from request stream and invokes doGet() or doPost() methods based on Http Method type.

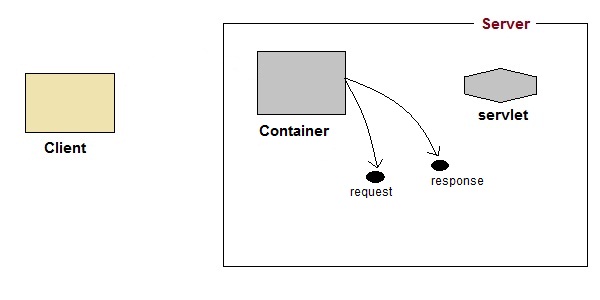
### How a Servlet Application works

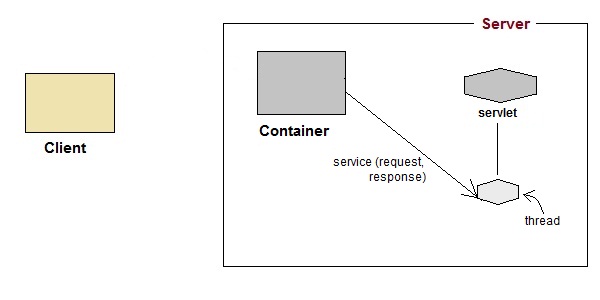
**Web container** is responsible for managing execution of servlets and JSP pages for Java EE application.

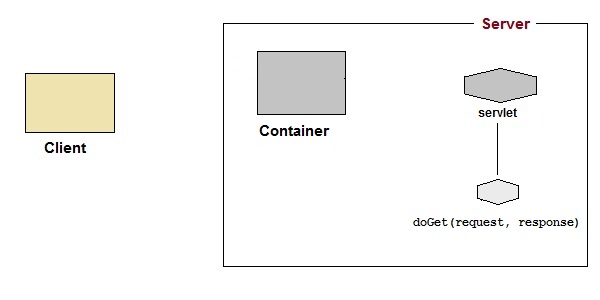
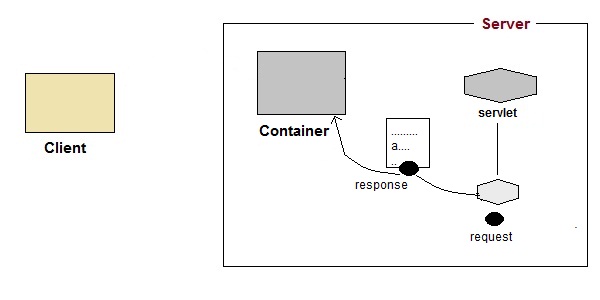
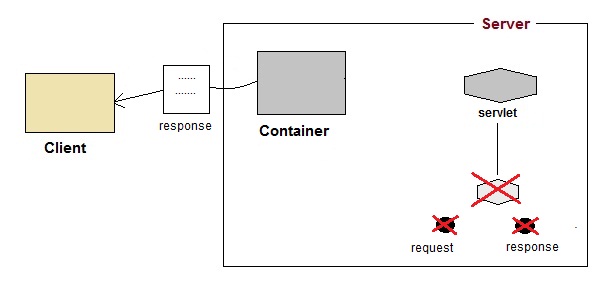
When a request comes in for a servlet, the server hands the request to the Web Container. **Web Container** is responsible for instantiating the servlet or creating a **new thread to handle the request**. It’s the job of Web Container to get the request and response to the servlet. The **container creates multiple threads to process multiple requests to a single servlet.**

**Servlets don't have a main() method**. Web Container manages the life cycle of a Servlet instance.

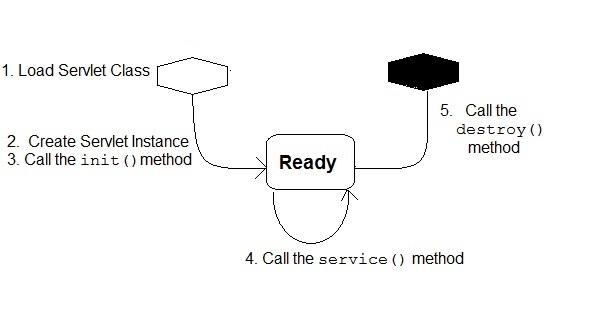
#### how a servlet application worksQuick Revision on How a Servlet works

1. User sends request for a servlet by clicking a link that has URL to a servlet.
2. The container finds the servlet using **deployment descriptor** and creates two objects :
   1. **HttpServletRequest**
   2. **HttpServletResponse**



1. Then the container creates or allocates a thread for that request and calls the Servlet's service()method and passes the **request, response** objects as arguments.
2. The service() method, then decides which servlet method, doGet() or doPost() to call, based on**HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.
3. Then the Servlet uses response object to write the response back to the client.
4. After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.

### Servlet Life Cycle

1. **Loading Servlet Class:** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation:** After the Servlet class is loaded; Web Container creates the instance of it. **Servlet instance is created only once in the life cycle**.
3. **Call to the init() method :** init() method is called by the Web Container on servlet instance to initialize the servlet.

**Signature of init() method :**

public void **init**(ServletConfig config) throws ServletException

1. **Call to the service() method :** The containers call the service() method each time the request for servlet is received. The service() method will then call the doGet() or doPost() methos based ont eh type of the HTTP request, as explained in previous lessons.

**Signature of service() method :**

public void **service**(ServletRequest request, ServletResponse response) throws ServletException, IOException

1. **Call to destroy() method:** The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity.

### Steps to Create Servlet Application using tomcat server

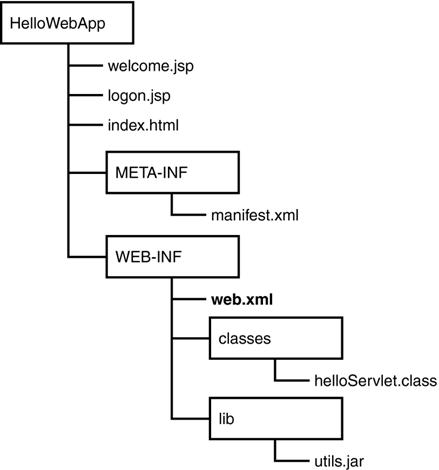
To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Download latest version of [Tomcat Server](http://tomcat.apache.org/) and install it on your machine.

After installing Tomcat Server on your machine follow the below mentioned steps:

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployment Descriptor for your application
5. Start the server and deploy the application

All these 5 steps are explained in details below, lets create our first Servlet Application.

#### 1. Creating the Directory Structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.

Create the above directory structure inside **Apache-Tomcat\webapps** directory. All HTML, static files(images, css etc) are kept directly under **Web application** folder. While all the Servlet classes are kept inside classesfolder.

The web.xml (deployement descriptor) file is kept under WEB-INF folder.

#### Creating a Servlet

There are three different ways to create a servlet.

* By implementing **Servlet** interface
* By extending **GenericServlet** class
* By extending **HttpServlet** class

But mostly a servlet is created by extending **HttpServlet** abstract class. As discussed earlier **HttpServlet** gives the definition of service() method of the **Servlet** interface. The servlet class that we will create should not override service() method. Our servlet class will override only doGet() or doPost() method.

When a request comes in for the servlet, the Web Container calls the servlet's service() method and depending on the type of request the service() method calls either the doGet() or doPost() method.

**NOTE:** By default a request is **Get** request.

#### Compiling a Servlet

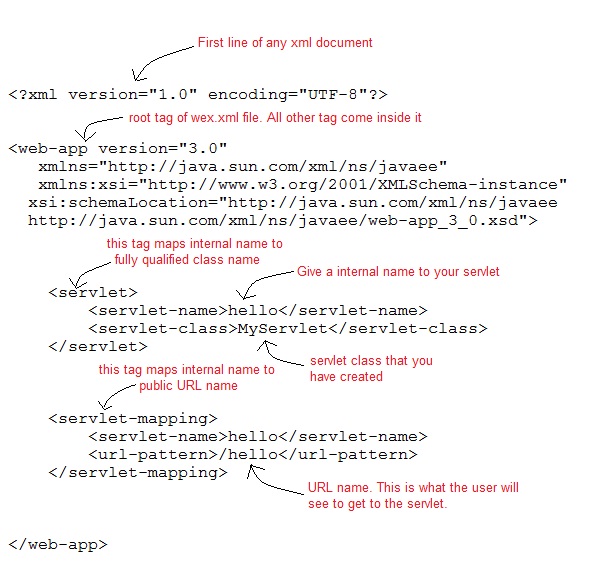
To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

Steps to compile a Servlet

* Set the Class Path.
* Download **servlet-api.jar** file.
* Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.
* Compile the Servlet class.

**NOTE:**After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/directory.

#### Create Deployment Descriptor

**Deployment Descriptor(DD)** is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

* Mapping URL to Servlet class.
* Initializing parameters.
* Defining Error page.
* Security roles.
* Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple **web.xml** file for our web application.

#### Start the Server

Double click on the **Tomcat7.exe** file to start your Apache Tomcat Server.

Or, execute the following command on your windows machine using RUN prompt.

C:\apache-tomcat-7.0.14\bin\Tomcat7.exe

#### Run Servlet Application

Open Browser and type **http:localhost:8080/folder\_name/url\_mapping\_name**

**Servlets**

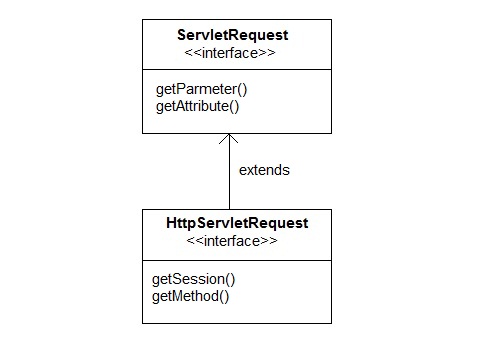
**Introduction to Servlet Request**

True job of a Servlet is to handle client request. Servlet API provides two important interfaces**javax.servlet.ServletRequest** and **javax.servlet.http.HttpServletRequest** to encapsulate client request. Implementation of these interfaces provides important information about client request to a servlet.

**Some Important Methods of ServletRequest**

|  |  |
| --- | --- |
| Methods | Description |
| Object getAttribute(String name) | return attribute set on request object by name |
| Enumeration getAttributeName() | return an Enumeration containing the names of the attributes available in this request |
| int getContentLength() | return size of request body |
| int getContentType() | return media type of request content |
| ServletInputStream getInputStream() | returns a input stream for reading binary data |
| String getParameter(String name) | returns value of parameter by name |
| String getLocalAddr() | returns the Internet Protocol(IP) address of the interface on which the request was received |
| Enumeration getParameterNames() | returns an enumeration of all parameter names |
| String[]getParameterValues(String name) | returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist |
| ServletContext getServletContext() | Return the servlet context of current request. |
| String getServerName() | returns the host name of the server to which the request was sent |
| int getServerPort() | returns the port number to which the request was sent |
| boolean isSecure() | Returns a Boolean indicating whether this request was made using a secure channel, such as HTTPS. |
| void removeAttribute(String name) | removes an attribute from this request |
| voidsetAttribute(String name, Object o) | Stores an attribute in this request. |

**HttpServletRequest interface**

**HttpServletRequest** interface adds the methods that relates to the **HTTP** protocol.

**Some important methods of HttpServletRequest**

|  |  |
| --- | --- |
| Methods | Description |
| String getContextPath() | returns the portion of the request URI that indicates the context of the request |
| Cookies getCookies() | returns an array containing all of the Cookie objects the client sent with this request |
| String getQueryString() | returns the query string that is contained in the request URL after the path |
| HttpSession getSession() | returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session |
| String getMethod() | Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT. |
| Part getPart(String name) | gets the Part with the given name |
| String getPathInfo() | Returns any extra path information associated with the URL the client sent when it made this request. |
| String getServletPath() | returns the part of this request's URL that calls the servlet |

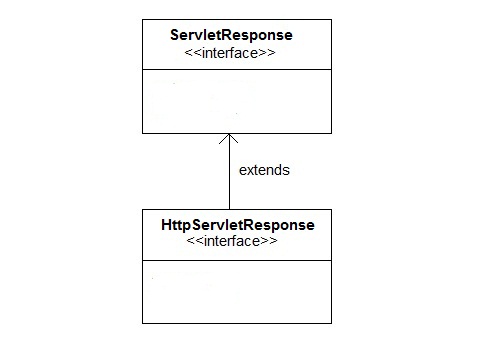
**Introduction to Servlet Response**

Servlet API provides two important interfaces **ServletResponse** and **HttpServletResponse** to assist in sending response to client.

**Some Important Methods of ServletResponse**

|  |  |
| --- | --- |
| Methods | Description |
| PrintWriter getWriter() | returns a PrintWriter object that can send character text to the client. |
| void setBufferSize(int size) | Sets the preferred buffer size for the body of the response |
| Void setContentLength(int len) | Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header |
| Void setContentType(String type) | sets the content type of the response being sent to the client before sending the respond. |
| void setBufferSize(int size) | sets the preferred buffer size for the body of the response. |
| boolean isCommitted() | returns a boolean indicating if the response has been committed |
| void setLocale(Locale loc) | sets the locale of the response, if the response has not been committed yet. |

**HttpServletResponse Interface**

**HttpServletResponse** interface adds the methods that relates to the **HTTP** response.

**Some Important Methods of HttpServletResponse**

|  |  |
| --- | --- |
| Methods | Description |
| void addCookie(Cookie cookie) | adds the specified cookie to the response. |
| void sendRedirect(String location) | Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer |
| int getStatus() | gets the current status code of this response |
| String getHeader(String name) | gets the value of the response header with the given name. |
| voidsetHeader(String name, String value) | sets a response header with the given name and value |
| void setStatus(int sc) | sets the status code for this response |
| void sendError(int sc, String msg) | sends an error response to the client using the specified status and clears the buffer |

### Introduction to Request Dispatcher

**RequestDispatcher** is an interface, implementation of which defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server.

#### Methods of RequestDispatcher

**RequestDispatcher** interface provides two important methods

|  |  |
| --- | --- |
| Methods | Description |
| voidforward(ServletRequest request, ServletResponse response) | forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server |
| voidinclude(ServletRequest request, ServletResponse response) | includes the content of a resource (servlet, JSP page, HTML file) in the response |

#### How to get an Object of RequestDispatcher

getRequestDispatcher() method of **ServletRequest** returns the object of **RequestDispatcher**.

RequestDispatcher *rs* = request.**getRequestDispatcher**("hello.html");

rs.**forward**(request,response);



**OR**

RequestDispatcher *rs* = request.**getRequestDispatcher**("hello.html");

rs.**include**(request,response);

**Introduction to 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() and 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.

### Introduction to ServletConfig interface

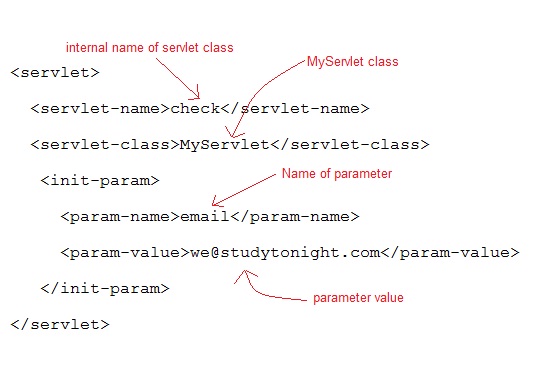
When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet. ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from **web.xml**(Deployment Descriptor).

#### 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

#### How to Initialize a Servlet inside web.xml

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



**Or, Inside the Servlet class, using following code,**

ServletConfig **sc** = getServletConfig();

out.println(sc.**getInitParameter**("email"));

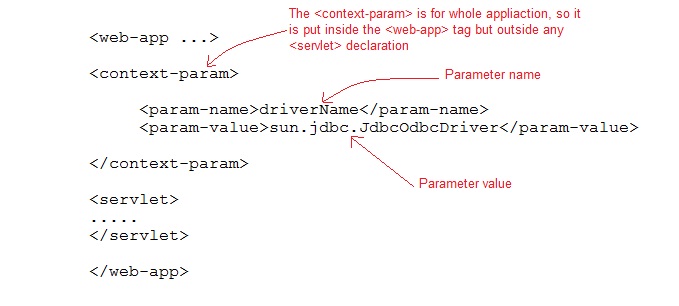
### Introduction to ServletContext Interface

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.

#### Some Important method of ServletContext

|  |  |
| --- | --- |
| Methods | Description |
| 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 |

#### How Context Parameter is Initialized inside web.xml



#### How to get the Object of ServletContext

ServletContext **app** = getServletContext();

*OR*

ServletContext **app** = getServletConfig().getServletContext();

#### Advantages of ServletContext

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

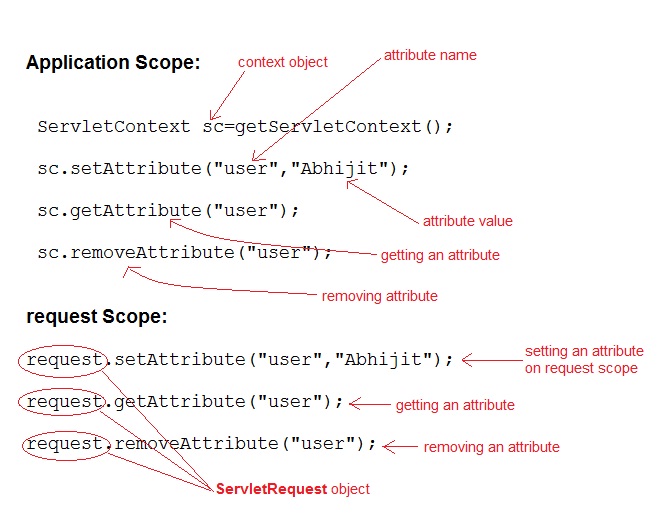
#### Difference between Context Init Parameters and Servlet Init Parameter

|  |  |
| --- | --- |
| 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 |

### Introduction to Attribute

An **attribute** is an object that is used to **share information** in a web app. Attribute allows Servlets to share information among them. Attributes can be SET and GET from one of the following scopes:

1. request
2. session
3. application



#### How to SET an Attribute

public void setAttribute(String name, Object obj) method is used to SET an Attribute.

#### How to GET an Attribute

Object getAttribute(String name) method is used to GET an attribute.

### Managing Session in Servlets

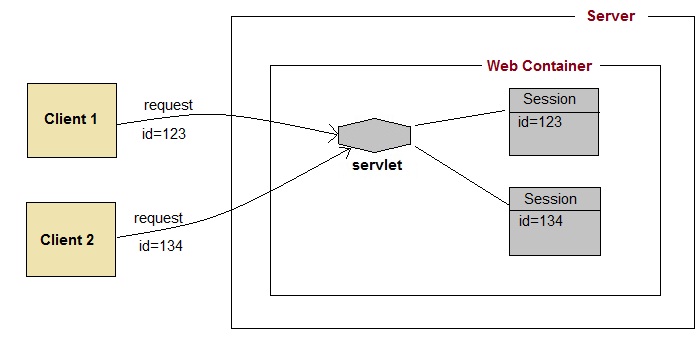
We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For e.g. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will 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. 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

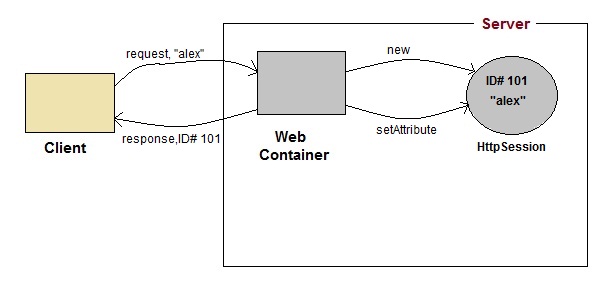


The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always 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 thegetSession() method of the **HttpServletRequest** object.

#### 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



#### Some Important Methods of HttpSession

|  |  |
| --- | --- |
| Methods | Description |
| 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. |

**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.

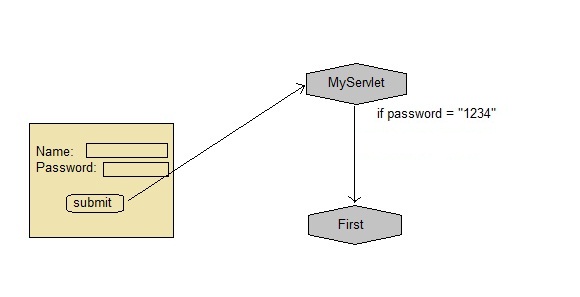
Using Cookies for storing client state has one shortcoming though, if the client has turned off Cookie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

**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.



**Example demonstrating usage of Cookies**

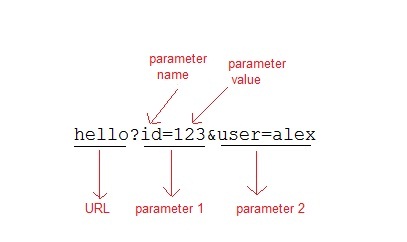


### Using URL Rewriting for Session Management

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.

**For Example:**



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.

### Using Hidden Form Field for Session Management

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.

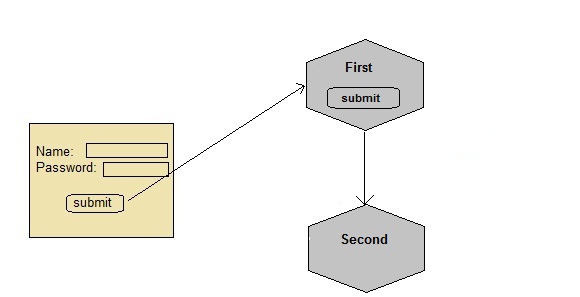
#### Advantages:

* 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, it’s easier to implement.

#### Disadvantage:

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

#### Example demonstrating usage of Hidden Form Field for Session



Like we created a hidden field in **First** Servlet, populated the value of user, and sent it to the **Second** Servlet, now Second servlet also has the user information. Similarly we will have to keep sending this information, wherever we need this, using hidden fields.

**ServletContextEvent and ServletContextListener?**

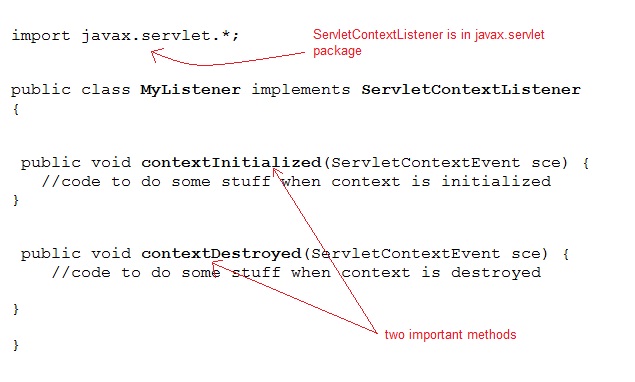
**ServletContextEvent** class gives notifications about changes to the servlet context of a web application.**ServletContextListener** receive the notifications about changes to the servlet context and perform some action. ServletContextListener is used to perform important task at the time when context is initialized and destroyed. In short, ServletContextEvent and ServletContextListener works in pair, whenever Servlet COntext changes, ServletContextEvent publishes a notification which is received by ServletContextListener and then, based on that certain tasks are performed by it.

**Methods of ServletContextListener Interface**

|  |  |
| --- | --- |
| Methods | Description |
| void contextDestroyed(ServletContextEvent e) | is invoked when the application is destroyed. |
| void contextInitialized(ServletContextEvent e) | is invoked when the application is initialized. |

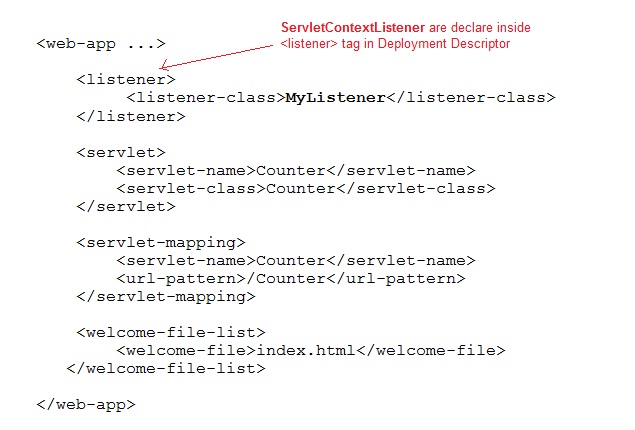
**Making and using a context listener**

Context listener is not a servlet or JSP, it's a class that implements **ServletContextListener** interface and provides definition of contextDestroyed() and contextInitialized().



**Example demonstrating usage of ServletContextListener**

**web.xml**



**What is ServletRequestAttributeEvent?**

**ServletRequestAttributeEvent** class gives notifications about changes to the attributes of **ServletRequest** in an application.

This class listens to the notifications and performs some important tasks whenever there is any change to the request attribute.

**Some Important Methods of ServletRequestAttributeListener**

|  |  |
| --- | --- |
| Methods | Description |
| Void attributeAdded(ServletRequestAttributeEvent e) | Notification that a new attribute was added to the servlet request. |
| Void attributeRemoved(ServletRequestAttributeEvent e) | Notification that an existing attribute was removed from the servlet request. |
| Void attributeReplaced(ServletRequestAttributeEvent e) | notification that an attribute was replaced on the servlet request |

**What is ServletContextAttributeEvent?**

**ServletContextAttributeEvent** class lets you know if an attribute in a web application *context* has been added, removed or replaced.

Implementation of ServletContextAttributeListener interface, receive notifications of changes to the attribute list on the servlet context of a web application.

**Some Important Methods of ServletContextAttributeListener**

|  |  |
| --- | --- |
| Methods | Description |
| Void attributeAdded(ServletContextAttributeEvent e) | Notification that a new attribute was added to the context. |
| Void attributeRemoved(ServletContextAttributeEvent e) | Notification that an existing attribute was removed from the context. |
| Void attributeReplaced(ServletContextAttributeEvent e) | notification that an attribute was replaced on the context |

**HttpSessionEvent and HttpSessionListener**

**HttpSessionEvent** class gives notifications for changes to sessions within a web application. **HttpSessionListener** receives notifications of changes to the list of active sessions in a web application and perform some action. HttpSessionListener is used to perform some important tasks when a session is created or destroyed. For example: counting the number of active session.

**Some other Session related Listeners**

|  |  |
| --- | --- |
| Listener | Description |
| HttpSessionActivationListener | Let's you know when a session moves from one Virtual machine to another. |
| HttpSessionBindingListener | Let's your attribute class object get notified when they are added or removed from session. |
| HttpSessionAttributeListener | Let's you know when any attribute is added, removed or replaced in a session. |

**Methods of HttpSessionListener**

|  |  |
| --- | --- |
| Methods | Description |
| void sessionCreated(HttpSessionEvent e) | Notification that a session was created. |
| void sessionDestroyed(HttpSessioEvent e) | Notification that a session was destroyed. |

**ServletRequestEvent and ServletRequestListener**

**ServletRequestEvent** class gives notification about lifecycle events for a **ServletRequest**. The source of the event is the ServletContext of the web application. **ServletRequestListener** receives the notifications generated by ServletRequestEvent and performs some specified tasks based on them.

**Some Importants Methods of ServletRequestListener**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Void requestDestroyed(ServletRequestEvent e) | Is invoked when request is about to go out of scope of the web application. |
| Void requestInitialized(ServletRequestEvent e) | Is invoked when request is about to come into scope of the web application. |

**Example**



Hence one can easily implement the ServletRequestListener Interface and provide definitions to the two methods provided by it, requestDestroyed() and requestInitialized().

As the name suggests these methods will be called when request is initialized and when it is destroyed, hence we can use them for various purposes, like **variable cleanup** when request is destroyed, or performing some**initial checks** when request is received etc.

### filter api in servletIntroduction to Filter API

**Filters** are components that you can use and configure to perform some filtering tasks. Filter is used for **pre-processing of requests and post-processing of responses**. You can have any number of filters for pre-processing of a request and post-processing of a response. Filters are **configured in the deployment descriptor** of a web application.

#### How Filters Works?

* When a request reaches the **Web Container**, it checks if any filter has URL patterns that matches the requested URL.
* The **Web Container** locates the first filter with a matching URL pattern and filter's code is executed.
* If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns left.
* If no error occurs, the request passes to the target servlet. Hence we know that the request will be passed to the target servlet only when all the related Filters are successfully executed.
* The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.
* At last the response will be passed to the **Web Container** which passes it to the client.

#### More about Filter API

**Filter API** is part of **Servlet API**. Filter interface is found in the **javax.servlet** package.

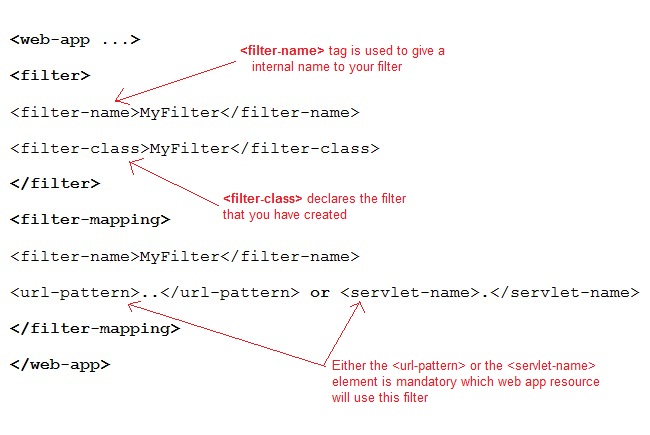
For creating a filter, we must implement Filter interface. Filter interface gives the following life cycle methods for a filter:

1. void init(FilterConfig filterConfig): invoked by the web container to indicate to a filter that it is being placed into service.
2. void doFilter(ServletRequest request, ServletResponse response, FilterChain chain): invoked by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
3. void destroy(): invoked by the web container to indicate to a filter that it is being taken out of service.

#### What is FilterChain Interface?

**FilterChain** object is used to invoke the next filter in the chain, or if the calling filter is the last filter in the chain then the rosource at the end of the chain invoked. The resources at the end of Filter chain can either be a target Servlet (in case of request flow) or the Client (in case of response flow) as described in the diagram above.

#### Declaring a Filter inside Deployment Descriptor



#### Example demonstrating Filter usage

In this example we are using Filter to authenticate (check correct username and password). Here **index.html** will ask username and password from the user, **MyFilter** will validate the password entered by the user, if the user has entered "1234" as password, then he will be forwarded to **first** servlet else the index.html will be shown again to the user.

This is exactly what we used to do earlier using two servlet classes earlier, one for validation and the other to welcome the user. Now we will insert a Filter for validating the user.

