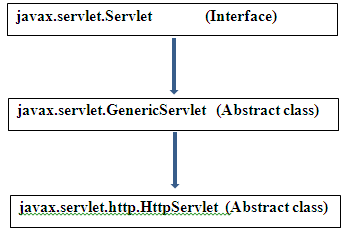
**What is Servlet vs GenericServlet vs HttpServlet?**

* Servlets are **platform-independent** server-side components, being written in Java.
* Before going for differences, first let us see how the three **Servlet**, **GenericServlet**, **HttpServlet** are related, their signatures and also at the end similarities.

**public class Validation extends HttpServlet**

* To write a servlet, everyone goes to extend the abstract class **HttpServlet**, like **Frame** is required to extend to create a frame.

###### Following figure shows the hierarchy of Servlet vs GenericServlet vs HttpServlet and to know from where HttpServlet comes.

  
**Figure on Servlet vs GenericServlet vs HttpServlet**

Observe the hierarchy and understand the relationship between the three (involved in multilevel inheritance). With the observation, a conclusion can be arrived, to write a Servlet three ways exist.

**a)** by implementing Servlet (it is interface)  
**b)** by extending GenericServlet (it is abstract class)  
**c)** by extending HttpServlet (it is abstract class)

* The minus point of the first way is, all the 5 abstract methods of the interface **Servlet** should be overridden even though Programmer is not interested in all (like the interface **WindowListener** to close the frame).
* A smart approach is inheriting **GenericServlet** (like using **WindowAdapter**) and overriding its only one abstract method **service()**.
* It is enough to the programmer to override only this method. It is a **callback** method (called implicitly).
* Still better way is extending HttpServlet and need not to override any methods as HttpServlet contains no abstract methods.
* Eventhough the HttpServlet does not contain any abstract methods, it is declared as abstract class by the Designers to not to allow the Programmer to create an object directly because a Servlet object is created by the system (here system is Servlet Container).

**1. Servlet interface**

It is the super interface for the remaining two – GenericServlet and HttpServlet. It contains 5 abstract methods and all inherited by GenericServlet and HttpServlet. Programmers implement Servlet interface who would like to develop their own container.

**2. GenericServlet**

It is the immediate subclass of **Servlet interface**. In this class, only one abstract method **service()** exist. Other 4 abstract methods of Servlet interface are given implementation (given body). Anyone who extends this **GenericServlet** should override **service()** method. It was used by the Programmers when the Web was not standardized to **HTTP** protocol. It is protocol independent; it can be used with any protocol, say, SMTP, FTP, CGI including HTTP etc.

**Syntax:**

**public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable**  
**3. HttpServlet**

When HTTP protocol was developed by W3C people to suit more Web requirements, the Servlet designers introduced **HttpServlet** to suit more for HTTP protocol. HttpServlet is protocol dependent and used specific to HTTP protocol only.

The immediate super class of **HttpServlet** is **GenericServlet**. HttpServlet overrides the **service()** method of GenericServlet. HttpServlet is abstract class but without any abstract methods.

With HttpServlet extension, **service()** method can be replaced by **doGet()** or **doPost()** with the same parameters of service() method.

**Syntax:**

**public abstract class HttpServlet extends GenericServlet implements java.io.Serializable**  
Being subclass of GenericServlet, the HttpServlet inherits all the properties (methods) of GenericServlet. So, if you extend HttpServlet, you can get the functionality of both.

Let us tabulate the differences for easy understanding and remembering.

|  |  |
| --- | --- |
| **GENERICSERVLET** | **HTTPSERVLET** |
| Can be used with any protocol (means, can handle any protocol). Protocol independent. | Should be used with HTTP protocol only (can handle HTTP specific protocols) . Protocol dependent. |
| All methods are concrete except service() method. service() method is abstract method. | All methods are concrete (non-abstract). service() is non-abstract method. |
| service() should be overridden being abstract in super interface. | service() method need not be overridden. |
| It is a must to use service() method as it is a callback method. | Being service() is non-abstract, it can be replaced by doGet() or doPost() methods. |
| Extends Object and implements interfaces Servlet, ServletConfig and Serializable. | Extends GenericServlet and implements interface Serializable |
| Direct subclass of Servet interface. | Direct subclass of GenericServlet. |
| Defined javax.servlet package. | Defined javax.servlet.http package. |
| All the classes and interfaces belonging to javax.servlet package are protocol independent. | All the classes and interfaces present in javax.servlet.http package are protocol dependent (specific to HTTP). |
| Not used now-a-days. | Used always. |

**Similarities:**

1. One common feature is both the classes are abstract classes.  
2. Used with Servlets only.

1. **javax.Servlet.ServletConfig**

* ServletConfig available in javax.servlet.\*; package
* ServletConfig object is one per servlet class
* Object of ServletConfig will be created during initialization process of the servlet
* This Config object is public to a particular servlet only
* *Scope*: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.
* We should give request explicitly, in order to create ServletConfig object for the first time
* In web.xml – <*init-param*> tag will be appear under <*servlet-class*> tag

### Advantage of ServletConfig

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

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

#### Syntax of getServletConfig() method

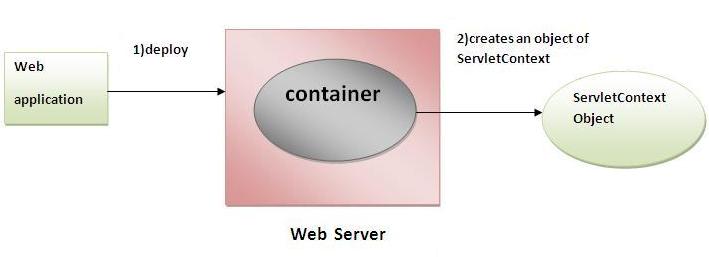
**public** ServletConfig getServletConfig();

1. **javax.Servlet.ServletContext**

* ServletContext available in javax.servlet.\*; package
* ServletContext object is global to entire web application
* Object of ServletContext will be created at the time of web application deployment
* *Scope*: As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
* ServletContext object will be available even before giving the first request
* In web.xml – <*context-param*> tag will be appear under <*web-app*> tag

### 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 servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.



### Commonly used methods of ServletContext interface

|  |
| --- |
| There is given some commonly used methods of ServletContext interface.   * **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 Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects. * **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

#### Syntax of getServletContext() method

**public** ServletContext getServletContext()

1. **javax.Servlet.GenericServlet**

* **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.
* GenericServlet class can handle any type of request so it is protocol-independent.
* You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

### Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

* **public void init(ServletConfig config)** is used to initialize the servlet.
* **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
* **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
* **public ServletConfig getServletConfig()** returns the object of ServletConfig.
* **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
* **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
* **public ServletContext getServletContext()** returns the object of ServletContext.
* **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
* **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
* **public String getServletName()** returns the name of the servlet object.
* **public void log(String msg)** writes the given message in the servlet log file.
* **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

1. **javax.Servlet.Servlet**

* **Servlet interface** provides common behaviour to all the servlets.
* Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

### Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void init(ServletConfig config)** | initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once. |
| **public void service(ServletRequest request,ServletResponse response)** | provides response for the incoming request. It is invoked at each request by the web container. |
| **public void destroy()** | is invoked only once and indicates that servlet is being destroyed. |
| **public ServletConfig getServletConfig()** | returns the object of ServletConfig. |
| **public String getServletInfo()** | returns information about servlet such as writer, copyright, version etc |

1. **javax.Servlet.ServletException**

This type of exception is thrown to indicate a Servlet problem.

**Constructors**

ServletException()

Public ServletException()

Public ServletException(String msg)

1. **javax.Servlet.ServletInputStream**

* **ServletInputStream** class provides stream to read binary data such as image etc. from the request object. It is an abstract class.
* The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class. So can be get as:

ServletInputStream sin=request.getInputStream();

### Method of ServletInputStream class

There are only one method defined in the ServletInputStream class.

**int readLine(byte[] b, int off, int len)**it reads the input stream.

**Constructors:**

ServletInputStream()

protected ServletInputStream()

1. **javax.Servlet.ServletOutputStream**

* **ServletOutputStream** class provides a stream to write binary data into the response. It is an abstract class.
* The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class. It may be get as:

ServletOutputStream out=response.getOutputStream();

### Methods of ServletOutputStream class

The ServletOutputStream class provides print() and println() methods that are overloaded.

* void print(boolean b){}
* void print(char c){}
* void print(int i){}
* void print(long l){}
* void print(float f){}
* void print(double d){}
* void print(String s){}
* void println{}
* void println(boolean b){}
* void println(char c){}
* void println(int i){}
* void println(long l){}
* void println(float f){}
* void println(double d){}
* void println(String s){}

1. **javax.Servlet.ServletRequest**

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

### Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public String getParameter(String name)** | is used to obtain the value of a parameter by name. |
| **public String[] getParameterValues(String name)** | returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box. |
| **java.util.Enumeration getParameterNames()** | returns an enumeration of all of the request parameter names. |
| **public int getContentLength()** | Returns the size of the request entity data, or -1 if not known. |
| **public String getCharacterEncoding()** | Returns the character set encoding for the input of this request. |
| **public String getContentType()** | Returns the Internet Media Type of the request entity data, or null if not known. |
| **public ServletInputStream getInputStream() throws IOException** | Returns an input stream for reading binary data in the request body. |
| **public abstract String getServerName()** | Returns the host name of the server that received the request. |
| **public int getServerPort()** | Returns the port number on which this request was received. |

1. **javax.Servlet.ServletResponse**

ServletResponse encapsulates basic properties of a response independent of protocol and provide output streams to write content to client, independent of protocol.

There are two response streams

1. Java.io.OutputStream for binary data
2. Java.io.writer for character data

**HtpServletResponse Interface adds support for:**

* HTTP headers and status codes
* URL rewriting used for session management
* Attributes to supply cookies to the client

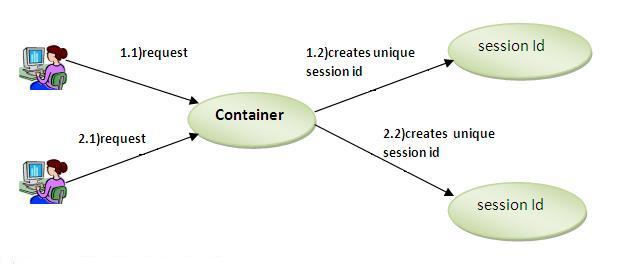
**Stream related methods** in ServletResponse

* ServletOutputStream **getOutputStream**()
* PrintWriter **getWriter**()
* void **flushBuffer**()
* int **getBufferSize**() – 0 if no buffering is used
* boolean **isCommited**()
* void **reset**()
* void **resetBuffer**()
* void **setBufferSize**(int size)
* String **getCharacterEncoding**()
* void **setContentype** (String type)
* String **getContentType**()

**HttpSession Interface**

In such 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:

1. bind objects
2. 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:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **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

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **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.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

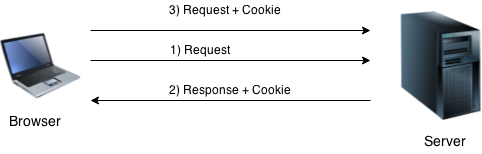
# Cookies in Servlet

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.



### Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. 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

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

**Disadvantage of Cookies**

1. It will not work if cookie is disabled from the browser.
2. 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.

### Constructor of Cookie class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| Cookie() | constructs a cookie. |
| Cookie(String name, String value) | constructs a cookie with a specified name and value. |

### Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

|  |  |
| --- | --- |
| **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. |

### Other methods required for using Cookies

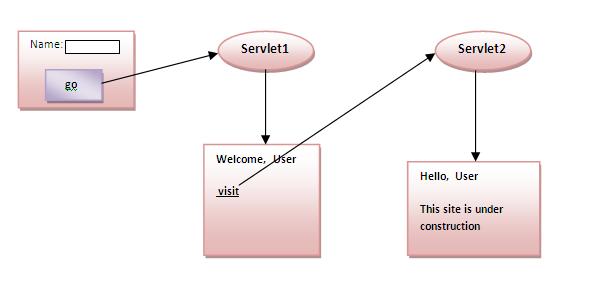
|  |
| --- |
| For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:   1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object. 2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser. |

# 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:

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.

1. 

### Advantage of URL Rewriting

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

### Disadvantage of URL Rewriting

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