Chapter 6
Servlet API and Lifecycle

6.1 Introduction

Now that the concept of servlet is in place, let's move one step further and understand the basic classes and interfaces that java provides to deal with servlets.

Java provides a servlet Application Programming interface[API] which is class library [basic classes and interfaces for implementing Servlets.

Servlets can access the entire family of Java APIs. The java Servlet API is the class library using which request can be processed and responses can be constructed, dynamically. The API thus helps define the expected interactions of a web container and a servlet.

6.2 servlet API

The servlet API is included into two packages: **javax.servlet** and **javax.servlet.http.**

The first package is intended to be generic and the second specific to the HTTP and HTTPS protocols.

Servlet are java classes that extend javax.servlet.http.HttpServlet. servlet perform their role in web apps by calling the servlet API.

6.2.1 javax.servlet package

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

Javax.servlet package is the core of servlet API.

This package includes classes for:

- Communicating with the host server and client:
  ServletRequest
  ServletResponse

- Communicating with the client:
  ServletInputStream
  ServletOutputStream

6.2.2 Servlet Interface

This interface is for developing servlets. A servlet is a body of Java code that is loaded into and runs inside a servlet engine, such as a web server. It receives and responds to requests from clients. For example, a client may need information from a database; a servlet can be written that receives the request, gets and processes the data as needed by the client, and then returns it to the client.

All servlets implement this interface. Servlet writers typically do this by subclassing either GenericServlet, which implements the Servlet interface, or by subclassing GenericServlet's descendent, HttpServlet.

The Servlet interface defines methods to initialize a servlet, to receive and respond to client requests, and to destroy a servlet and its resources. These methods are called by the network service in the following manner:

1. Servlet is created then **init**ialized.
2. Zero or more **service** calls from clients are handled
3. Servlet is **destroy**ed then garbage collected and finalized

The following are the methods available in this interface:

Methods	Description

`init()`	Initializes the servlet. The method is called once, automatically, by the servlet engine when it loads the servlet. It is guaranteed to finish before any service requests are accepted. The init method should save the ServletConfig object so that it can be returned by the getServletConfig method. If a fatal initialization error occurs, the init method should throw an appropriate "UnavailableException" exception.
`getServletConfig()`	Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet. This is the ServletConfig object passed to the init method; the init method should have stored this object so that this method could return it.
`Service()`	Carries out a single request from the client. The method implements a request and response paradigm. The request object contains information about the service request, including parameters provided by the client. The response object is used to return information to the client. The request and response objects rely on the underlying network transport for quality of service guarantees, such as reordering, duplication, privacy, and authentication. Service requests are not handled until servlet initialization has completed. Any requests for service that are received during initialization block until it is complete. Note that servlets typically run inside multi-threaded servers; servers can handle multiple service requests simultaneously. It is the servlet writer's responsibility to synchronize access to any shared resources, such as network connections or the servlet's class and instance variables.
`getServletInfo()`	Returns a string containing information about the servlet, such as its author, version, and copyright. As this method may be called to display such information in an administrative tool that is servlet

	engine specfic, the string that this method returns should be plain text and not contain markup.
`destroy()`	Cleans up whatever resources are being held (e.g., memory, file handles, threads) and makes sure that any persistent state is synchronized with the servlet's current in-memory state. The method is called once, automatically, by the network service when it unloads the servlet. After destroy is run, it cannot be called again until the network service reloads the servlet. When the network service removes a servlet, it calls destroy after all service calls have been completed, or a service-specific number of seconds have passed, whichever comes first. In the case of long-running operations, there could be other threads running service requests when destroy is called. The servlet writer is responsible for making sure that any threads still in the service method complete.

6.3 servletContext interface

Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

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 servlet, it is better to provide it from the web.xml file using the **`<context-param>`** element.

The ServletContext object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.

2. The ServletContext object can be used to get configuration information from the web.xml file.

3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.

4. The ServletContext object can be used to provide inter-application communication.

The following are the methods available in this interface:

Methods	Description
`getContextPath()`	Returns the context path of the web application.
`getContext()`	Returns a ServletContext object that corresponds to a specified URL on the server.
`getMimeType()`	Returns the MIME type of the specified file, or null if the MIME type is not known.
`getResource()`	Returns a URL to the resource that is mapped to the given path.
`log`	Writes the specified message to a servlet log file, usually an event log. The name and type of the servlet log file is specific to the servlet container.
`getRealPath`	Gets the *real* path corresponding to the given *virtual* path.
`getServerInfo()`	Returns the name and version of the servlet container on which the servlet is running.
`getInitParameter`	Returns a String containing the value of the named context-wide initialization parameter, or null if the parameter does not exist.
`getInitParameterNames()`	Returns the names of the context's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the context has no initialization parameters.
`setInitParameter`	Sets the context initialization parameter with the given name and value on this ServletContext.
`getAttribute`	Returns the servlet container attribute with the given name, or null if there is no attribute by that name.
`getAttributeNames()`	Returns an Enumeration containing the attribute names available within this ServletContext.
`setAttribute`	Binds an object to a given attribute name in this ServletContext. If the name specified is already used for an attribute, this method will replace the attribute with the new to the new attribute.
`removeAttribute`	Removes the attribute with the given name from this ServletContext.

`getServletContextName()`	Returns the name of this web application corresponding to this ServletContext as specified in the deployment descriptor for this web application by the display-name element.
`addServlet`	Adds the servlet with the given name and class name to this servlet context.

- How to get the object of ServletContext interface

1. `getServletContext()` method of ServletConfig interface returns the object of ServletContext.
2. `getServletContext()` method of GenericServlet class returns the object of ServletContext.

- Syntax of `getServletContext()` method

1. `public ServletContext getServletContext()`

- 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();
```

6.4 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.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

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.

The following are the methods available in this interface:

Methods	Description

`getServletName`	Returns the name of this servlet instance. The name may be provided via server administration, assigned in the web application deployment descriptor, or for an unregistered (and thus unnamed) servlet instance it will be the servlet's class name.
`getServletContext`	Returns a reference to the ServletContext in which the caller is executing.
`getInitParameter`	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
`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.

- How to get the object of ServletConfig

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

- Syntax of `getServletConfig()` method

```
public ServletConfig getServletConfig();
```

- Example of `getServletConfig()` method

```
ServletConfig config=getServletConfig();
```

//Now we can call the methods of ServletConfig interface

- Difference between ServletConfig and ServletContext

Signature: `public interface ServletConfig`

ServletConfig is implemented by the servlet container **to initialize a single servlet** using init(). That is, you can pass initialization parameters to the servlet using the web.xml deployment descriptor. For understanding, this is similar to a constructor in a java class.

Example code:

```
<servlet>
<servlet-name>ServletConfigTest</servlet-name>
<servlet-class>com.javapapers.ServletConfigTest</servlet-class>
<init-param>
<param-name>topic</param-name>
<param-value>Difference between ServletConfig and
ServletContext</param-value>
</init-param>
</servlet>
```

Signature: `public interface ServletContext`

ServletContext is implemented by the servlet container **for all servlet**to communicate with its servlet container, for example, to get the MIME type of a file, to get dispatch requests, or to write to a log file. That is to get detail about its execution environment. It is applicable only within a single Java Virtual Machine. If a web applicationa is distributed between multiple JVM this will not work. For understanding, this is like a application global variable mechanism for a single web application deployed in only one JVM.

The ServletContext object is contained within the ServletConfig object. That is, the ServletContext can be accessed using the ServletConfig object within a servlet. You can specify param-value pairs for ServletContext object in <context-param> tags in web.xml file.

Example code:

```
<context-param>
<param-name>globalVariable</param-name>
<param-value>javapapers.com</param-value>
</context-param>
```

6.5 ServletRequest and ServletResponse Interfaces

6.5.1public interface ServletRequest

This interface is for getting data from the client to the servlet for a service request. Network service developers implement the ServletRequest interface. The methods are then used by servlets when the service method is executed; the ServletRequest object is passed as an argument to the service method.

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.

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

Methods	Description
`getParameter`	Returns the value of a request parameter as a String, or null if the parameter does not exist. Request parameters are extra information sent with the request.
`getParameterValues`	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.
`getParameterNames`	Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.
`getContentLength`	Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
`getContentType`	Returns the Internet Media Type of the request entity data, or null if not known.
`getInputStream` [**`public ServletInputStream getInputStream() throws IOException`**]	Retrieves the body of the request as binary data using a ServletInputStream. Either this method or getReader() may be called to read the body, not both.
`getServerName`	Returns the host name of the server to which the request was sent. It is the value of the part before ":" in the Host header value, if any, or the resolved server name, or the server IP address.
`getServerPort`	Returns the port number to which the request was sent. It is the value of the part after ":" in the Host header value, if any, or the server port where the client connection was accepted on.

Example of ServletRequest to display the name of the user

In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

index.html

```
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login">
</form>
```

DemoServ.java

```
import javax.servlet.http.*;

import javax.servlet.*;

import java.io.*;

public class DemoServ extends HttpServlet

{

public void doGet(HttpServletRequest req,HttpServletResponse res)

throws ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();


String name=req.getParameter("name");//will return value

pw.println("Welcome "+name);


pw.close();

}

}
```

6.5.2public interface ServletResponse

ServletResponse interface defines object to send response to the client.object of ServletResponse is created by the servlet container and works as argument for the servlet's service method. If sending binary data in MIME body response then you have to use **ServletOutputStream** returned by*getOutputStream()*. For Character data, use **PrintWriter** object returned by *getWriter()*.

It provides many methods. Here, we are defining some of them.

Methods	Description
`getOutputStream` `[public ServletOutputStream getOutputStream() throws java.io.IOException ]`	Returns a ServletOutputStream suitable for writing binary data in the response. The servlet container does not encode the binary data. Calling flush() on the ServletOutputStream commits the response. Either this method or getWriter() may be called to write the body, not both.
`getWriter` `[public java.io.PrintWriter getWriter() throws java.io.IOException]`	Returns a PrintWriter object that can send character text to the client. The PrintWriter uses the character encoding returned by getCharacterEncoding(). Calling flush() on the PrintWriter commits the response. Either this method or getOutputStream() may be called to write the body, not both.
`getCharacterEncoding`	Returns the name of the character encoding (MIME charset) used for the body sent in this response.
`setBufferSize`	Sets the preferred buffer size for the body of the response. The servlet container will use a buffer at least as large as the size requested. The actual buffer size used can be found using getBufferSize. A larger buffer allows more content to be written before anything is actually sent, thus providing the

	servlet with more time to set appropriate status codes and headers. A smaller buffer decreases server memory load and allows the client to start receiving data more quickly.
`getBufferSize`	Returns the actual buffer size used for the response. If no buffering is used, this method returns 0.
`flushBuffer`	Forces any content in the buffer to be written to the client. A call to this method automatically commits the response, meaning the status code and headers will be written.

Example :

ServletResponseExample.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletResponseExample extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head><title>ServletResponse</title></title>");
out.println("<body>");
out.println("<h1>ServletResponse Example</h1>");
out.println("</body></html>");
}
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
```

```
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
<display-name>ServletExample</display-name>

<!-- servlet response mapping-->
<servlet>
<servlet-name>ServletResponseExample</servlet-name>
<servlet-class>net.roseindia.ServletResponseExample</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>ServletResponseExample</servlet-name>
<url-pattern>/servletResponseExample</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>form.jsp</welcome-file>

</welcome-file-list>
</web-app>
```

6.6 GenericServlet Class

GenericServlet class implements **Servlet**,**ServletConfig** and **Serializable** interfaces. It provides the implementaion 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.

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

Methods	Description
`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.

Let's see the simple example of servlet by inheriting the GenericServlet class.

```
import java.io.*;
import javax.servlet.*;
```

```
public class First extends GenericServlet
{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException
{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");
}
}
```

6.7 ServletInputStream And ServletOutputStream Classes

6.7.1Class ServletInputStream

```
java.lang.Object
  └java.io.InputStream
      └javax.servlet.ServletInputStream
```

```
public abstract class ServletInputStream
extends java.io.InputStream
```

Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time.

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();
```

There are only one method defined in the ServletInputStream class.

```
public int readLine(byte[] b,
                    int off,
                    int len)
             throws java.io.IOException
```

Reads the input stream, one line at a time. Starting at an offset, reads bytes into an array, until it reads a certain number of bytes or reaches a newline character, which it reads into the array as well.

This method returns -1 if it reaches the end of the input stream before reading the maximum number of bytes.

Parameters:
b - an array of bytes into which data is read
off - an integer specifying the character at which this method begins reading
len - an integer specifying the maximum number of bytes to read

Returns:
an integer specifying the actual number of bytes read, or -1 if the end of the stream is reached

Throws:
java.io.IOException - if an input or output exception has occurred

6.7.2Class ServletOutputStream

```
java.lang.Object
  └java.io.OutputStream
      └javax.servlet.ServletOutputStream
```

```
public abstract class ServletOutputStream
extends java.io.OutputStream
```

Provides an output stream for sending binary data to the client. A ServletOutputStream object is normally retrieved via the ServletResponse.getOutputStream() method.

This is an abstract class that the servlet container implements. Subclasses of this class must implement the `java.io.OutputStream.write(int)` method.

Constructor Detail:

`ServletOutputStream`

`protected ServletOutputStream()`

Does nothing, because this is an abstract class.

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

Methods	Description
`public void` **`print`**`(java.lang.String s)`	Writes a String to the client, without a carriage return-line feed (CRLF) character at the end.
`public void` **`print`**`(boolean b)`	Writes a boolean value to the client, with no carriage return-line feed (CRLF) character at the end.
`public void` **`print`**`(char c)`	Writes a character to the client, with no carriage return-line feed (CRLF) at the end.
`public void` **`print`**`(int i)`	Writes an int to the client, with no carriage return-line feed (CRLF) at the end.
`public void` **`print`**`(long l)`	Writes a long value to the client, with no carriage return-line feed (CRLF) at the end.
`public void` **`print`**`(float f)`	Writes a float value to the client, with no carriage return-line feed (CRLF) at the end.
`public void` **`print`**`(double d)`	Writes a double value to the client, with no carriage return-line feed (CRLF) at the end.
`public void` **`println`**`()`	Writes a carriage return-line feed (CRLF) to the client.
`public void` **`println`**`(java.lang.String s)`	Writes a String to the client, followed by a carriage return-line feed (CRLF).
`public void` **`println`**`(boolean b)`	Writes a boolean value to the client, followed by a carriage return-line feed (CRLF).
`public void` **`println`**`(char c)`	Writes a character to the client, followed by a carriage return-line feed (CRLF).
`public void` **`println`**`(int i)`	Writes an int to the client, followed by a carriage return-line feed (CRLF) character.
`public void` **`println`**`(long l)`	Writes a long value to the client, followed by a carriage return-line feed (CRLF).
`public void` **`println`**`(float f)`	Writes a float value to the client, followed by a carriage return-line feed (CRLF).

`public void println(double d)`	Writes a double value to the client, followed by a carriage return-line feed (CRLF).

6.8 RequestDispatcherInterface

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. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

Methods	Description
forward `[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. **Parameters:** request - a ServletRequest object that represents the request the client makes of the servlet response - a ServletResponse object that represents the response the servlet returns to the client
include `[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. **Parameters:** request - a ServletRequest object that contains the client's request response - a ServletResponse object that contains the servlet's response

Forward method():

As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.

include method():

As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

- How to get the object of RequestDispatcher

The `getRequestDispatcher()` method of ServletRequest interface returns the object of RequestDispatcher.

- Syntax of `getRequestDispatcher` method

```
public RequestDispatcher getRequestDispatcher(String resource);
```

- Example of using getRequestDispatcher method

```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
```
//servlet2 is the url-pattern of the second servlet
```
  rd.forward(request, response);
```
//method may be include or forward

6.9 HttpServlet Class

```
java.lang.Object
  └javax.servlet.GenericServlet
      └javax.servlet.http.HttpServlet
```

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

```
public abstract class HttpServlet
extends GenericServlet
implements java.io.Serializable
```

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests
- `doPut`, for HTTP PUT requests
- `doDelete`, for HTTP DELETE requests
- `init` and `destroy`, to manage resources that are held for the life of the servlet
- `getServletInfo`, which the servlet uses to provide information about itself

There's almost no reason to override the `service` method. `service` handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the `do`*`XXX`* methods listed above).

Likewise, there's almost no reason to override the `doOptions` and `doTrace` methods.

Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

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

Methods	Description
doGet `[protected void doGet(HttpServletRequest req, HttpServletResponse resp)]`	handles the GET request. It is invoked by the web container. **Parameters:** req - an HttpServletRequest object that contains the request the client has made of the servlet resp - an HttpServletResponse object that contains the response the servlet sends to the client

getLastModified [protected long **getLastModified**(HttpServletRequest req)]	returns the time when HttpServletRequest was last modified since midnight May 18, 1991 GMT.
doHead [protected void **doHead**(HttpServletRequest req, HttpServletResponse resp)]	handles the HEAD request. It is invoked by the web container. The client sends a HEAD request when it wants to see only the headers of a response, such as Content-Type or Content-Length. The HTTP HEAD method counts the output bytes in the response to set the Content-Length header accurately.
doPost [protected void **doPost**(HttpServletRequest req, HttpServletResponse resp)]	handles the POST request. It is invoked by the web container. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.
doPut [protected void **doPut**(HttpServletRequest req, HttpServletResponse resp)]	handles the PUT request. It is invoked by the web container. The PUT operation allows a client to place a file on the server and is similar to sending a file by FTP.
doDelete [protected void **doDelete**(HttpServletRequest req, HttpServletResponse resp)]	handles the DELETE request. It is invoked by the web container. The DELETE operation allows a client to remove a document or Web page from the server.
doOptions [protected void **doOptions**(HttpServletRequest req, HttpServletResponse resp)]	handles the OPTIONS request. It is invoked by the web container. The OPTIONS request determines which HTTP methods the server supports and returns an appropriate header. For example, if a servlet overrides doGet, this

	method returns the following header: Allow: `GET, HEAD, TRACE, OPTIONS`
doTrace [protected void **doTrace**(HttpServletRequest req, HttpServletResponse resp)]	handles the TRACE request. It is invoked by the web container. A TRACE returns the headers sent with the TRACE request to the client, so that they can be used in debugging.
service [public void **service**(ServletRequest req,ServletResponse res)]	dispatches the request to the protected service method by converting the request and response object into http type.
service [protected void **service**(HttpServletRequest req, HttpServletResponse res)]	receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.

6.10 HttpServletRequest and HttpServletResponse Interfaces

6.10.1 HttpServletRequest Interface:

```
public interface HttpServletRequest
extends ServletRequest
```

Extends the ServletRequest interface to provide request information for HTTP servlets.

The servlet container creates an `HttpServletRequest` object and passes it as an argument to the servlet's service methods (`doGet, doPost`, etc).

The HttpServletRequest breaks a request down into parsed elements, such as request URI, query arguments and headers. Various get methods allow you to access different parts of the request.

There are many methods in HttpServletRequest Interface. Some of these are as follows:

Methods	Description
`getAuthType`	Returns the name of the authentication scheme used to protect the servlet. All servlet containers support basic, form and client certificate authentication, and may additionally support digest authentication. If the servlet is not authenticated null is returned.
`getCookies`	Returns an array containing all of the Cookie objects the client sent with this request. This method returns null if no cookies were sent.
`getDateHeader`	Returns the value of the specified request header as a long value that represents a Date object. Use this method with headers that contain dates, such as If-Modified-Since.
`getHeaderNames`	Returns an enumeration of all the header names this request contains. If the request has no headers, this method returns an empty enumeration.
`getPathInfo`	Returns any extra path information associated with the URL the client sent when it made this request.
`getContextPath`	Returns the portion of the request URI that indicates the context of the request. The context path always comes first in a request URI. The path starts with a "/" character but does not end with a "/" character. For servlets in the default (root) context, this method returns "". The container does not decode this string.
`getSession`	Returns the current session associated with this request, or if the request does not have a session, creates one.
`getServletPath`	Returns the path of this request's URL that calls the servlet.

6.10.2 HttpServletResponse

```
public interface HttpServletResponse
extends ServletResponse
```

Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response. For example, it has methods to access HTTP headers and cookies.

The servlet container creates an `HttpServletResponse` object and passes it as an argument to the servlet's service methods (`doGet`, `doPost`, etc).

There are many methods in HttpServletRequest Interface. Some of these are as follows:

Methods	Description
`addCookie`	Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.
`containsHeader`	Returns a boolean indicating whether the named response header has already been set.
`encodeURL`	Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged. The implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary.

`sendError`
`[public void` **`sendError`**`(int sc,`
`java.lang.String msg)]`
	Sends an error response to the client using the specified status. The server defaults to creating the response to look like an HTML-formatted server error page containing the specified message, setting the content type to "text/html", leaving cookies and other headers unmodified. If an error-page declaration has been made for the web application corresponding to the status code passed in, it will be served back in preference to the suggested msg parameter.
	Parameters:

	sc - the error status code msg - the descriptive message
`addDateHeader`	Adds a response header with the given name and date-value. The date is specified in terms of milliseconds since the epoch. This method allows response headers to have multiple values.

`setStatus`
`[public void` **`setStatus`**`(int sc)]`
	Sets the status code for this response.
	Parameters:
	sc - the status code

6.11 HttpSession Interface

`public interface` **`HttpSession`**

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to

- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
- Bind objects to sessions, allowing user information to persist across multiple user connections

- 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

Methods	Description
`getId`	Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.
`getCreationTime`	Returns the time when this session was created, measured in milliseconds since midnight May 18, 1991 GMT.
`getLastAccessedTime`	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight May 18, 1991 GMT, and marked by the time the container received the request.
`invalidate`	Invalidates this session then unbinds any objects bound to it.

6.12 Servlet Lifecycle

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.

5. destroy method is invoked.

As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

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

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

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

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
public void destroy()
```