Servlets Interview Questions and Answers

Q. Explain Servlets Lifecycle?

The web container maintains the life cycle of a servlet instance.

Stages of the Servlet Life Cycle:

1. The servlet is initialized by calling the **init()** method.
2. The servlet calls **service()** method to process a client's request.
3. The servlet is terminated by calling the **destroy()** method.
4. Finally, servlet is garbage collected by the garbage collector of the JVM.

- **The init() Method**
  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

```
public void init(ServletConfig config) throws ServletException {
    // Initialization code...
}
```

- **The service() Method**
  The servlet container calls the service() method to handle requests coming from the client and to write the formatted response back to the client. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc.

Syntax

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

- **The doGet() Method**
  A GET request results from a normal request for a URL or from an HTML form

that has no METHOD specified and it should be handled by doGet() method.

Syntax

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

- **The doPost() Method**
  A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

Syntax

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

- **The destroy() Method**
  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.

Syntax

```
public void destroy() {
   // Finalization code...
}
```

Q. What is ServletContext Interface?

ServletContext is a configuration Object which is created when web application is started. It contains different initialization parameter that can be configured in web.xml.

ServletContext Interface Methods

1. public String getInitParameter(String name): Returns the parameter value for the specified parameter name.**2. public Enumeration getInitParameterNames()**: Returns the names of the context's initialization parameters.**3. public void setAttribute(String name,Object object)**: sets the given object in the application scope.**4. public Object getAttribute(String name)**: Returns the attribute for the specified name.**5. public Enumeration getInitParameterNames()**: Returns the names of the context's initialization parameters as an Enumeration of String objects.**6. public void removeAttribute(String name)**: Removes the attribute with the given name from the servlet context.

Example: DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet{
   public void doGet(HttpServletRequest request,HttpServletResponse response)
   throws ServletException,IOException {

       response.setContentType("text/html");
       PrintWriter pwriter=response.getWriter();

       // ServletContext object creation
       ServletContext scontext=getServletContext();

       // fetching values of initialization parameters and printing it
       String userName=scontext.getInitParameter("uname");
       pwriter.println("User name is="+userName);
       String userEmail=scontext.getInitParameter("email");
       pwriter.println("Email Id is="+userEmail);
       pwriter.close();
   }
}
```

web.xml

```
<web-app>
    <servlet>
        <servlet-name>UserDetails</servlet-name>
        <servlet-class>DemoServlet</servlet-class>
    </servlet>
    <context-param>
        <param-name>uname</param-name>
        <param-value>Pradeep Kumar</param-value>
    </context-param>
    <context-param>
        <param-name>email</param-name>
        <param-value>pradeep.vwa@gmail.com</param-value>
    </context-param>
    <servlet-mapping>
        <servlet-name>UserDetails</servlet-name>
    <url-pattern>/context</url-pattern>
    </servlet-mapping>
</web-app>
```

Q. What is a servlet?

A servlet is an interface whose implementation extends the functionality of a server. A servlet interacts with clients through a request-response principle. Although servlets can handle any request, they are commonly used to expand web servers.

Most of the classes and interfaces required to create servlets are contained in `javax.servlet` and packages `javax.servlet.http`.

Basic servlet methods

- `public void init(ServletConfig config) throws ServletException` starts immediately after loading the servlet into memory;
- `public ServletConfig getServletConfig()` returns a reference to an object that provides access to servlet configuration information;
- `public String getServletInfo()` returns a string containing information about the servlet, for example: author and version of the servlet;
- `public void service(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException` called to process each request;
- `public void destroy()` performed before unloading the servlet from memory.

Q. What are the advantages of servlet technology over CGI (Common Gateway Interface)?

- Servlets provide better query processing performance and more efficient use of memory by taking advantage of multithreading (a new thread is created for each request, which is faster than allocating memory for a new object for each request, as happens in CGI).
- Servlets, both platform and system are independent. Thus, a web application written using servlets can be run in any servlet container that implements this standard and in any operating system.
- Using servlets increases the reliability of the program, as the servlet container itself takes care of the servlet life cycle (and therefore memory leaks), security, and the garbage collector.
- Servlets are relatively easy to learn and maintain, so the developer only needs to care about the business logic of the application, and not the internal implementation of web technologies.

Q. What is a servlet container?

A servlet container is a program that is a server that provides system support for servlets and ensures their life cycle in accordance with the rules defined in the specifications. It can work as a full-fledged stand-alone web server, be a page provider for another web server, or integrate into a Java EE application server.

The servlet container provides data exchange between the servlet and clients, takes on the execution of functions such as creating a software environment for a functioning servlet, identifying and authorizing clients, and organizing a session for each of them.

The most famous servlet container implementations are:

- Apache tomcat
- Jetty
- Jboss
- Glassfish
- IBM WebSphere
- Oracle Weblogic

Q. How does the servlet container manage the servlet life cycle, when and what methods are called?

The servlet container manages the four phases of the servlet life cycle:

- **Servlet class loading -** when the container receives a request for a servlet, the servlet class is loaded into memory and its constructor is called without parameters.
- **Initialization of the servlet class -** after the class is loaded, the container initializes the object ServletConfigfor this servlet and implements it through the `init()` method. This is where the servlet class is converted from a regular class to a servlet.
- **Request processing** - after initialization, the servlet is ready to process requests. For each client request, the servlet container spawns a new thread and calls the method `service()` by passing a reference to the object's responses and request.

- **Delete** - when the container stops or the application stops, the servlet container destroys the servlet classes by calling the `destroy()` method.

Thus, the servlet is created the first time it is accessed and lives throughout the entire application run time (unlike the class objects that are destroyed by the garbage collector after they are no longer used) and the entire servlet life cycle can be described as a sequence of method calls:

- `public void init(ServletConfig config)` - Used by the container to initialize the servlet. Called once during the servlet's lifetime.
- `public void service(ServletRequest request, ServletResponse response)` - called for each request. The method cannot be called before the init()method is executed .
- `public void destroy()` - called to destroy the servlet (once during the life of the servlet).

Q. What actions do you need to do when creating servlets?

To create a servlet `ExampleServlet`, you must describe it in the deployment descriptor:

```
<servlet-mapping>
    <servlet-name> ExampleServlet </servlet-name>
    <url-pattern> / example </url-pattern>
</servlet-mapping>
<servlet>
    <servlet-name> ExampleServlet </servlet-name>
    <servlet-class> xyz.company.ExampleServlet </servlet-class>
    <init-param>
        <param-name> config </param-name>
        <param-value> default </param-value>
    </init-param>
</servlet>
```

Then create a class `xyz.company.ExampleServlet` by inheriting from `HttpServlet` and implement the logic of its work in the method `service()`/ methods `doGet()`/ `doPost()`.

Q. What are the most common tasks performed in a servlet container?

- **Support data exchange**: The servlet container provides an easy way to exchange data between the web client (browser) and the servlet. Thanks to the container, there is no need to create a socket listener on the server to track requests from the client, as well as parse the request and generate a response. All these important and complex tasks are solved using the container and the developer can focus on the business logic of the application.

- **Servlet and resource lifecycle management**: From servlet loading into memory, initialization, implementation of methods, and ending with servlet destruction. The container also provides additional utilities, such as JNDI, for managing the resource pool.
- **Multithreading support**: The container independently creates a new thread for each request and provides it with a request and response for processing. Thus, the servlet is not reinitialized for each request and thereby saves memory and reduces the time before processing the request.
- **JSP support**: JSP classes are not like standard Java classes, but the servlet container converts each JSP into a servlet and is then managed by the container as a regular servlet.
- **Various tasks**: The servlet container manages the resource pool, application memory, and garbage collector. Security settings and more.

Q. How to call another servlet from one servlet?

To call a servlet from the same application, you must use the inter-servlet communication mechanisms through method calls `RequestDispatcher()`:

- `forward()` - passes the execution of the request to another servlet;
- `include()` - provides the ability to include the result of another servlet in the returned response.

If you need to call a servlet belonging to another application, then you `RequestDispatcher` will not be able to use it, because it is defined only for the current application. For such purposes, you must use the method `ServletResponse-sendRedirect()` which is provided with the full URL of another servlet. You can use to transfer data between servlets cookies.

Q. What is Request Dispatcher?

The `RequestDispatcher` interface is used to transfer the request to another resource, while it is possible to add data received from this resource to the servlet's own response. This interface is also used for internal communication between servlets in the same context.

Two methods are implemented in the interface:

- `void forward(ServletRequest var1, ServletResponse var2)` - transfers the request from the servlet to another resource (servlet, JSP or HTML file) on the server.
- `void include(ServletRequest var1, ServletResponse var2)` - includes the content of the resource (servlet, JSP or HTML page) in the response.

Access to the interface can be obtained using the interface method `ServletContext`- `RequestDispatcher getRequestDispatcher(String path)`, where the path starting with `/` is interpreted relative to the current root path of the context.

Q. What are the differences between forward() and sendRedirect() methods?

forward()

- Performed on the server side;
- The request is redirected to another resource within the same server;
- It does not depend on the client request protocol of the client, as it is provided by the servlet container;
- Cannot be used to inject a servlet in a different context;
- The client does not know about the actually processed resource and the URL in the string remains the same;
- It is faster than the method sendRedirect();
- Defined in the interface RequestDispatcher.

sendRedirect()

- Performed on the client side;
- A response is returned to the client 302 (redirect)and the request is redirected to another server;
- It can only be used with HTTP clients;
- Permitted to be used to implement a servlet in a different context;
- The URL is changed to the address of the new resource;

- Slower forward()because Requires creating a new request;
- Defined in the interface HttpServletResponse.

Q. What are servlet attributes used for and how do you work with them?

Servlet attributes are used for internal servlet communication.

The web application has the ability to work with attributes using the methods `setAttribute()`, `getAttribute()`, `removeAttribute()`, `getAttributeNames()`, who provided interfaces ServletRequest, HttpSessionand ServletContext(for the scope request, the session, context The respectively).

Q. How to get the real servlet location on the server?

The real path to the servlet location on the server can be obtained from the object ServletContext: `getServletContext().getRealPath(request.getServletPath())`.

Q. How to get server information from a servlet?

Information about the server can be obtained from the object ServletContext: `getServletContext().getServerInfo()`.

Q. How to get the client IP address on the server?

Client IP address can be obtained by calling `request.getRemoteAddr()`.

Q. What servlet wrapper classes do you know?

Custom handlers can `ServletRequest` also `ServletResponse` be implemented by adding new or overriding existing methods for wrapper classes `ServletRequestWrapper( HttpServletRequestWrapper)` and `ServletResponseWrapper( HttpServletRequestWrapper)`.

Q. What are the differences GenericServlet and HttpServlet?

An abstract class GenericServletis an implementation of the interface independent of the protocol used Servlet, and an abstract class, HttpServletin turn, extends GenericServletfor the HTTP protocol.

Q. What are the main methods present in the class HttpServlet?

- `doGet()` for processing HTTP requests GET;
- `doPost()` for processing HTTP requests POST;
- `doPut()` for processing HTTP requests PUT;
- `doDelete()` for processing HTTP requests DELETE;
- `doHead()` for processing HTTP requests HEAD;
- `doOptions()` for processing HTTP requests OPTIONS;
- `doTrace()` for processing HTTP requests TRACE.

Q. Which HTTP method is not immutable?

An HTTP method is called immutable if it always returns the same result on the same request. HTTP methods `GET`, `PUT`, `DELETE`, `HEAD` and `OPTIONS` are immutable, so it is necessary to implement an application so that these methods return the same results consistently. Variable methods include a method `POST` that is used to implement something that changes with each request.

For example, to access a static HTML page, a method is used `GET`, because it always returns the same result. If you need to save any information, for example in a database, you need to use the `POST` method.

Q. What are the different methods for managing session in servlets?

There are several ways to provide a unique session identifier:

- **User Authentication** - Providing credentials by the user at the time of authentication. Information transmitted in this way is then used to maintain the session. This method will not work if the user is logged in from several places at the same time.
- **HTML Hidden Field** - Assigning a unique value to the hidden field of an HTML page when a user starts a session. This method cannot be used with links, because it needs a form confirmation with a hidden field every time a request is generated. In addition, it is not safe, because there is the possibility of simple substitution of such an identifier.

- **URL Rewriting** - Add a session id as a URL parameter. It’s a tedious operation, because it requires constant monitoring of this identifier with every request or response.
- **Cookies** - The use of small pieces of data sent by the web server and stored on the user's device. This method will not work if the client disables the use of cookies.
- **Session Management API** - Using a special API for session tracking, built on the basis and on the methods described above and which solves particular problems of the listed methods:
    - Most often, just tracking a session is not enough, you also need to save any additional data about it that may be required when processing subsequent requests. Implementing this behavior requires a lot of extra effort.
    - All of the above methods are not universal: for each of them you can choose a specific scenario in which they will not work.

Q. What are cookies? What methods for working with cookies are provided in servlets?

Cookies (“cookies”) - a small piece of data sent by a web server and stored on the user's device. Each time you try to open a site page, the web client sends cookies corresponding to this site to the web server as part of the HTTP request. It is used to save data on the user side and in practice it is usually used to:

- user authentication;
- storage of personal preferences and user settings;
- tracking the status of the user's access session;
- maintaining a variety of statistics.

Servlet API provides support for cookies through the class `javax.servlet.http.Cookie`:

- To get an array of cookies from the request, you must use the method `HttpServletRequest.getCookies()`. There are no methods for adding cookies to `HttpServletRequest`.
- Used to add a cookie to the response `HttpServletResponse.addCookie(Cookie c)`. There is no method to receive cookies `HttpServletResponse`.

Q. What is URL Rewriting?

URL Rewriting - special rewriting (recoding) of the original URL. This mechanism can be used to control the session in servlets when cookies are disabled.

Q. What is difference between encodeURL() and encodeRedirectURL()?

`HttpServletResponse.encodeURL()` provides a way to convert a URL to HTML hyperlink with the conversion of special characters and spaces, as well as adding session id to the URL. This behavior is similar `java.net.URLEncoder.encode()`, but with the addition of an additional parameter `jsessionid` at the end of the URL.

The method `HttpServletResponse.encodeRedirectURL()` translates the URL for later use in the method `sendRedirect()`.

Thus for HTML hyperlinks when URL rewriting is necessary to use `encodeURL()`, and for the URL when redirecting - `encodeRedirectUrl()`.

Q. How to notify an object in a session that a session is invalid or has ended?

To be sure that the object will be notified about the termination of the session, you need to implement the interface `javax.servlet.http.HttpSessionBindingListener`. Two methods of this interface: `valueBound()` and `valueUnbound()` are used when adding an object as an attribute to a session and when destroying a session, respectively.

Q. What is an effective way to make sure that all servlets are only accessible to the user with the correct session?

Servlet filters are used to intercept all requests between the servlet container and the servlet. Therefore, it is logical to use the appropriate filter to check the necessary information (for example, the validity of the session) in the request.

Q. How can we provide transport layer security for our web application?

To provide transport layer security, you must configure SSL support for the container servlet. How to do this depends on the particular implementation of the servlet container.

Q. What are the main features that appeared in the Servlet 3 specification?

- **Servlet Annotations**: Prior to Servlet 3, the entire configuration was contained in web.xml, which led to errors and inconvenience when working with a large number of servlets. Examples of annotations @WebServlet, @WebInitParam, @WebFilter, @WebListener.
- **Web fragments**: A single-page web application can contain many modules: all modules are registered in fragment.xmla folder META-INF\. This allows you to split the web application into separate modules, assembled as .jar files in a separate lib\directory.
- **Dynamically add web components**: Now you can programmatically add filters and listeners using the ServletContextobject. For this purpose methods are used addServlet(), addFilter(), addListener(). Using this innovation, it became possible to build a dynamic system in which the necessary object will be created and called only when necessary.
- **Asynchronous execution**: Support for asynchronous processing allows you to transfer the execution of a request to another thread without keeping the entire server busy.

Q. What authentication methods are available to the servlet?

The servlet specification defines four types of authentication:

- **HTTP Basic Authentication -** `BASIC`: When accessing private resources, a window appears that asks you to enter authentication information.
- **Form Based Login -** `FORM`: The native html form is used:
- **HTTP Digest Authentication -** `DIGEST`: Digital authentication with encryption.
- **HTTPS Authentication -** `CLIENT-CERT`: Authentication with a client certificate.

```
<login-config>
    <auth-method> FORM </auth-method>
    <form-login-config>
        <form-login-page> /login.html </form-login-page>
        <form-error-page> /error.html </form-error-page>
    </form-login-config>
</login-config>
```

Q. What is a deployment descriptor?

A deployment descriptor is an artifact configuration file that will be deployed to a servlet container. In the Java Platform specification, Enterprise Edition, the deployment descriptor describes how a component, module, or application (such as a web or enterprise application) should be deployed.

This configuration file specifies the deployment settings for a module or application with specific settings, security settings, and describes specific configuration requirements. The deployment descriptor file syntax is XML.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app  xmlns = "<http://java.sun.com/xml/ns/j2ee>"
     xmlns : xsi = "<http://www.w3.org/2001/XMLSchema-instance>"
     xsi : schemaLocation = "<http://java.sun.com/xml/ns/j2ee> <http://java.sun.com/xm
l/ns/j2ee/web-app_2_4.xsd>"
     version="2.4">

    <display-name> Display name. </display-name>
    <description> Description text. </description>

    <servlet>
        <servlet-name> ExampleServlet </servlet-name>
        <servlet-class> xyz.company.ExampleServlet </servlet-class>
        <load-on-startup> 1 </load-on-startup>
        <init-param>
            <param-name> configuration </param-name>
            <param-value> default </param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name> ExampleServlet </servlet-name>
        <url-pattern> / example </url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name> ExampleJSP </servlet-name>
        <jsp-file> /sample/Example.jsp </jsp-file>
    </servlet>

    <context-param>
        <param-name> myParam </param-name>
        <param-value> the value </param-value>
    </context-param>
</web-app>
```

For web applications, the deployment descriptor must be named `web.xml` and located in the directory `WEB-INF` at the root of the web application. This file is the standard deployment descriptor defined in the specification. There are also other types of descriptors, such as a deployment descriptor file `sun-web.xml` that contains Sun GlassFish Enterprise Server- specific deployment information for that particular application server or a file `application.xml` in the J2EE `META-INF` application directory.

Q. Why do servlets use different listeners?

The Listener (listener) acts as a trigger, performing certain actions when an event occurs in the servlet's life cycle.

Listeners divided by scope:

- Request:
    - `ServletRequestListener` used to catch the moment of creation and destruction of the request;
    - `ServletRequestAttributeListener` used to listen for events occurring with request attributes.
- Context:
    - `ServletContextListener` allows you to catch the moment when the context is initialized or destroyed;
    - `ServletContextAttributeListener` used to listen for events occurring with attributes in context.
- Session:
    - `HttpSessionListener` allows you to catch the moment of creation and destruction of the session;
    - `HttpSessionAttributeListener` used when listening to events occurring with attributes in the session;
    - `HttpSessionActivationListener` used if session migration occurs between different JVMs in distributed applications;
    - `HttpSessionBindingListener` It is also used to listen for events occurring with attributes in the session. The difference between `HttpSessionAttributeListener` and `HttpSessionBindingListenerlisteners`: the first is declared in `web.xml`; an instance of the class is created automatically by the container in the singular and applies to all sessions; second: an instance of the class must be created and assigned to a certain session “manually”, the number of instances is also independently regulated.

Connecting listeners:

```
<web-app>
    ...
    <listener>
        <listener-class> xyz.company.ExampleListener </listener-class>
    </listener>
    ...
</web-app>
```

`HttpSessionBindingListener` It is connected as an attribute directly to the session, i.e. to connect it you need to:

- create an instance of a class implementing this interface;
- put the created instance into the session with `setAttribute(String, Object)`.

Q. When to use servlet filters, and when to listeners?

Filters should be used if it is necessary to process incoming or outgoing data (for example: for authentication, format conversion, compression, encryption, etc.), if it is necessary to respond to events, it is better to use listeners.

Q. How to implement servlet launch simultaneously with application launch?

A servlet container typically loads a servlet at the first request of a client.

If you need to load a servlet right at the start of the application (for example, if the servlet has been loading for a long time), you should use an element <load-on-startup>in the descriptor or an annotation @loadOnStartupin the servlet code, which will indicate the need to load the servlet at startup.

If the integer value of this parameter is negative, then the servlet will be loaded when the client requests. Otherwise, it will load at the start of the application, and the smaller the number, the earlier it will be in the download queue.

```
<servlet>
    <servlet-name> ExampleServlet </servlet-name>
    <servlet-class> xyz.company.ExampleServlet </servlet-class>
    <load-on-startup> 1 </load-on-startup>
</servlet>
```

Q. How to handle exceptions thrown by another servlet in an application?

When the application throws an exception, the servlet container processes it and creates an HTML response. This is similar to what happens with error codes like 404, 403, etc.

In addition to this, it is possible to write your own servlets to handle exceptions and errors with their indication in the deployment descriptor:

```
<error-page>
    <error-code> 404 </error-code>
    <location> / AppExceptionHandler </location>
</error-page>

<error-page>
    <exception-type> javax.servlet.ServletException </exception-type>
    <location> / AppExceptionHandler </location>
</error-page>
```

The main task of these servlets is to handle the error / exception and generate an understandable response to the user. For example, provide a link to the main page or a description of the error.

Q. Why do we have servlet filters?

A servlet filter is reusable Java code that converts the contents of HTTP requests, HTTP responses, and the information contained in HTML headers. The servlet filter pre-processes the request before it gets to the servlet, and / or subsequently processes the response coming from the servlet.

Servlet filters can:

- intercept servlet initiation before servlet initiation;
- determine the contents of the request before the servlet is triggered;
- modify the headers and data of the request into which the incoming request is packaged;
- modify the headers and response data into which the received response is packaged;
- intercept servlet initiation after accessing the servlet.

A servlet filter can be configured to work with a single servlet or a group of servlets. The basis for the formation of filters is an interface javax.servlet.Filterthat implements three methods:

- `void init(FilterConfig config) throws ServletException;`
- `void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException;`
- `void destroy();`

Q. What is difference between sendRedirect() and forward() in Servlet?

- **SendRedirect()**: This method is declared in **HttpServletResponse** Interface. It is used to redirect client request to some other location for further processing, the new location is available on different server or different context.our web container handle this and transfer the request using browser, and this request is visible in browser as a new request.

Signature:

```
void sendRedirect(String url)
```

- **Forward()**: This method is declared in **RequestDispatcher** Interface. It is used to pass the request to another resource for further processing within the same server, another resource could be any servlet, jsp page any kind of file.

Signature:

```
forward(ServletRequest request, ServletResponse response)
```

Title	Forward()	SendRediret()
Untitled	The forward() method is executed in the server side.	The sendRedirect() method is executed in the client side.
Untitled	The request is transfer to other resource within same server.	The request is transfer to other resource to different server.

Title	Forward()	SendRediret()
Untitled	It does not depend on the client’s request protocol since the forward ( ) method is provided by the servlet container.	The sendRedirect() method is provided under HTTP so it can be used only with HTTP clients.
Untitled	The request is shared by the target resource.	New request is created for the destination resource.
Untitled	Only one call is consumed in this method.	Two request and response calls are consumed.
Untitled	It can be used within server.	It can be used within and outside the server.
Untitled	The forward() method is faster than sendRedirect() method.	The sendRedirect() method is slower because when new request is created old request object is lost.
Untitled	It is declared in RequestDispatcher interface.	It is declared in HttpServletResponse.
Untitled	Signature: *forward(ServletRequest request, ServletResponse response)*	Signature: *void sendRedirect(String url)*

Example: sendRedirect() method

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class RedirectServlet extends HttpServlet {

  protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    res.sendRedirect("<https://www.java.com/en/>");
    out.close();
  }
}
```

Example: forward() method

```
//index.html
```

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="ISO-8859-1">
    <title>Insert title here</title>
 </head>
<body>
    <form action="Simple" method="get">
        Name: <input type="text" name="username">
        password: <input type="password" name="password"><br />
    <input type="submit" value="Submit" />
    </form>
</body>
</html>
```

SimpleServlet.java

```
package javaexample.net.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SimpleServlet extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String str = request.getParameter("username");
    String st = request.getParameter("password");

    if (st.equals("javaexample")) {
      RequestDispatcher rd = request.getRequestDispatcher("Welcome");
      rd.forward(request, response);
    } else {
      out.print("Sorry username or password incorrect!");
      RequestDispatcher rd = request.getRequestDispatcher("/index.html");
      rd.include(request, response);
    }
  }
}
```

Some more questions you can prepare.

Q. What is a Server Side Include (SSI)?

Q. What are the differences between ServletContext vs ServletConfig?

Q. What is MIME Type?

Q. Why HttpServlet class is declared abstract?

Q. How to notify an object in session when session is invalidated or timed-out?

Q. How to make sure a servlet is loaded at the application startup?

Q. Write a servlet to upload file on server?

Q. How do we go with database connection and log4j integration in servlet?

Q. What is the effective way to make sure all the servlets are accessible only when user has a valid session?

Q. Why do we have servlet listeners?

Q. What are Scriptlets?

Q. What is different between web server and application server?

Q. Which HTTP method is non-idempotent?

Q. What is difference between PrintWriter and ServletOutputStream?

Q. How can we create deadlock situation in servlet?

Q. What is SingleThreadModel interface?

Q. Do we need to override service() method?

Q. Is it good idea to create servlet constructor?

Q. Are Servlets Thread Safe? How to achieve thread safety in servlets?

Q. why we should override only no-airs init() method.

Q. What are different ways for servlet authentication?

Q. What is Servlet Chaining?

Q. How do you find out what client machine is making a request to your servlet?