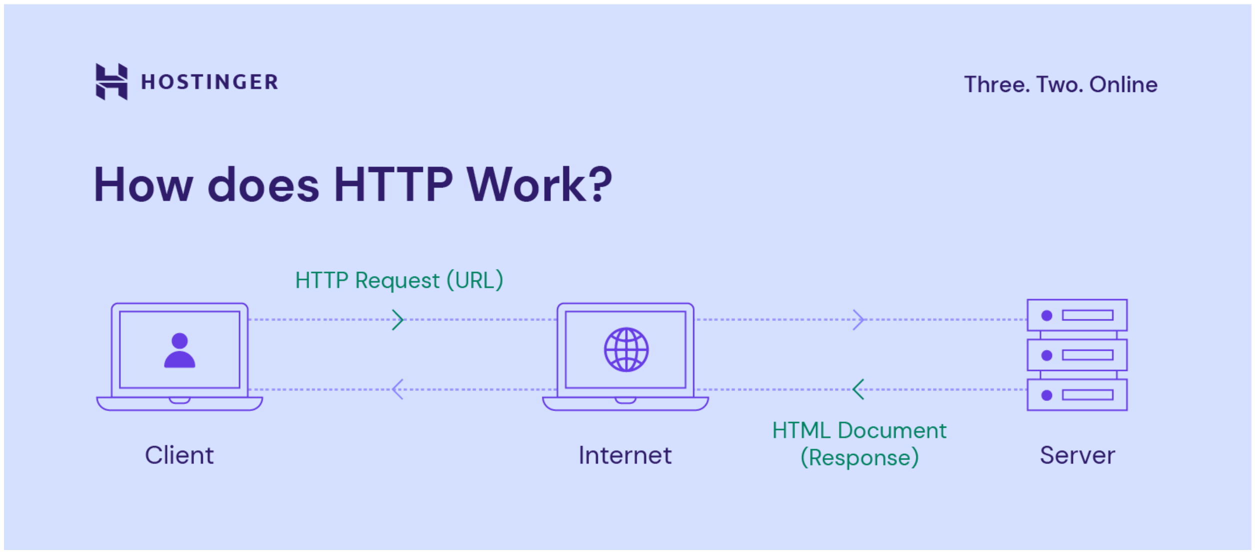
Introduction to Servlet:

Web is a request-response model.
Http is the protocol of the web world.
We have other protocols too like https,ftp,smtp…



When a client computer sends a request to a server, in simple terms, when you type in a URL on your browser,
the request ultimately reaches another computer, the 'server', which serves the particular web page you requested.
Servlet is a program that runs on that web server that can process your request and send the right content back.
These programs have a life cycle from creation to destruction.The servlet container like Tomcat manages the life cycle of servlets.We can see servlet as an extension of a server.

Servlets have two characteristics: the function on the server side and the ability to handle sophisticated requests from the web server.

In execution, Java servlets run in a container, which is a type of operating environment for a component that delivers a variety of services. The Web server or application server generally provides this container.

The container also handles component security, limiting component access to the local operating environment. Also if desired, it provides authentication services to authenticate a user's identity.

Servlets are secure in three different ways.

1. since they are written in Java, direct memory access calls and other violations are not

possible.

2. Servlets use the security manager of the server for enforcement of certain security policies.

The security manager can restrict access of files to untrusted servlets.

3. Lastly, servlets combine with SSL to give a full fledged secure communication over the net.

Client/Browser: The client through the browser sends HTTP requests to the webserver.

Web Server: Includes several components to control access to files hosted on the server.
A basic HTTP server will comprehend URLs and follow HTTP protocol to deliver the contents of hosted websites.

Web Container: This is a web server component that interacts with Java servlets.
It manages the servlets' life cycle, including loading, unloading, managing request and response objects,
and URL mapping on the server-side. Ex- Tomcat,JBoss,Weblogic Server....

There are two main packages that make up the servlet architecture.
1. javax.servlet
2. javax.servlet.http
the javax.servlet package contains generic interfaces and classes that are implemented and

extended by all the servlets. The javax.servlet.http package contains the classes that are
extended when creating HTTP-specific servlets.
The starting point is the interface javax.servlet.Servlet.
This interface provides the framework for all the servlets. It defines five methods out of which the
three most important methods are
init() method that initialises a servlet
service() method that receives and responds to the client requests.

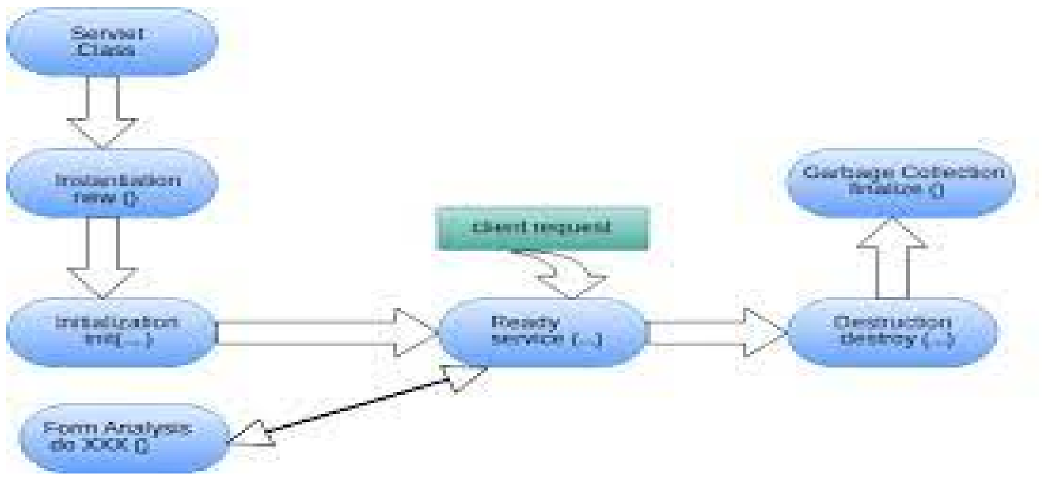
destroy() method that performs clean up.

The two main classes are GenericServlet and HttpServlet. The HttpServlet class has been

extended from GenericServlet. During the development of our own servlets, we will be extending
from one of these classes. The GenericServlet is an abstract class and if we extend this we have
to override the service() method. The service() method takes two arguments viz., ServletRequest
and ServletResponse. The ServletRequest object holds the information that is being sent to the
servlet whereas the ServletResponse object is where we place the data to be sent to the client.
But when we extends the HttpServlet we will not extends the service() method. The HTTP
protocol uses either a GET method or POST method to invoke a servlet. So the HttpServlet has
two methods doGet() and doPost() which takes two arguments viz., HttpServletRequest and
HttpServletResponse. Internally the service() method of the HttpServlet has been overridden to
understand whether the request from the client has come by either the GET/POST method and

fire the appropriate method. So we have to override either the doGet() or the doPost() method
only in this case.

LIFE CYCLE OF A SERVLET



The life cycle of a servlet is very simple. Once a servlet is constructed and initialised, it then
services zero or more requests until the service it extends shuts down. At this point, the servlet is
destroyed and garbage collected.
init()
This is where the servlet's life cycle begins. It is called by the server immediately after the servlet
has been instantiated. This is called only once during the lifetime of a servlet. In this init() method, a servlet creates and initialises resources it will be using while handling the client
requests. The syntax of this method is :
public void init(ServletConfig sc) throws ServletException
The init method takes a ServletConfig object as an argument. This object has to be saved so that

it can be referenced later. Normally it is achieved by placing a super.init(sc) call as the first line
inside the init() method.
The init() method also throws a ServletException. This will be raised if for some reason the
servlet cannot initialise the necessary resources.
service()
This handles all the requests sent by the client. This cannot fire unless and until the init() method
has fired successfully. If we use HttpServlet as the parent class, we usually do not override
method instead provide a doGet() or a doPost() method.

public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
These methods will be using a PrintWriter object to output the results to the client. Hence the
IOException is being declared in the throws clause.
destroy()
this signifies the end of a servlet's life. When a server is being shut down, it calls the servlet's
shutdown method. At this juncture, any resources help up by the servlet like any database
connections etc., have to be closed.

----

doGet and doPost are methods of the javax.servlet.http.HttpServlet class that are used to handle
HTTP GET and POST requests, respectively.

The doGet method is called by the server (via the service method) when the client requests a GET request.
It is used to retrieve information from the server.
Tge httpmethodis used by Google Search is get.We could see the url appended with the query strings.

The doPost method is called by the server (via the service method) when the client requests a POST request. It is used to send information to the server.

Difference Between Doget and Dopost

Get method main job is asking the server for the resources. Get is one of the simplest HTTP method. Get method has a size limitation of 1024 characters.It is faster and of limited size.If we dont provide the method in html form ,then doGet is assumed.It is the default method.

What is the Post method?

Post method is Provide information. by using post we can send as well as request data to the server.
it is slowr but has no limitation on size.

What is Difference Between **Doget() and Dopost() :**

doGet()
doPost()
In doget() Parameters not encrypted. In dopost() Parameters encrypted.
doget() allows bookmark. dopost() disallows bookmark.
doget() method is idempotent.(It could be changed with different values. dopost() method does not idempotent..
In doget() not change anything on the server. In dopost() server is expected to remember.
doget() is request information. dopost() is provide information.
In doget() parameters are appended to URL and sent with header information. In dopost(), on the other hand, will send the information through socket back to the webservers and it won’t show in the URL bar.

doget() is Faster. dopost() Slower.
In doget() only 1024 characters limit. In dopost() doest not have limit.

ServletContext and ServletCofig:

Both are interfaces inj ServletAPI.
ServletConfig and ServletContext both are import interfaces in ServletAPI.
The Key Difference between ServletConfig and ServletContext is that ServletConfig is used by only a single servlet to get configuration information whereas ServletContext is used by multiple objects to get configuration information.

ServletConfig
ServletConfig available in javax.servlet.*; package
ServletConfig is used to get configuration information from the web.xml file.
If configuration information is modified from the web.xml file no need to change the servlet.
Example: String str = config.getInitParameter("name")
ServletConfig Advantage
ServletConfig is that you don't need to edit the servlet file if the information is modified from the web.xml file.

ServletContext
ServletContext available in javax.servlet.*; package
ServletContext is created by the web container at the time of deploying the project.
It can be used to get configuration information from a web.xml file.
There is only one object to the entire web application.
ServletContext Advantage

Differences:

ServletConfig object is one per servlet class. ServletContext object is global to the entire web application.
Object of ServletConfig will be created during the initialization process of the servlet.
Object of ServletContext will be created at the time of web application deployment
We have to give the request explicitly in order to create the ServletConfig object for the first time ServletContext object can be available even before giving the first request

Scope: As long as a servlet is executing, the ServletConfig object will be available, it will be destroyed once the servlet execution is completed   Scope: As long as a web application is executing, the ServletContext object will be available, and it will be destroyed once the application is removed from the server
ServletConfig object is used while only one servlet requires information shared by it.   ServletContext object is used while application requires information shared by it
getServletConfig() method is used to obtain Servletconfig object   getServletContext() method is used to obtain ServletContext object
In web.xml — <init-param> tag will be appear under <servlet-class> tag.   In web.xml — <context-param> tag will be appear under <web-app> tag.
Easy to maintain

Session handling:

A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the users computer.

A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the users computer rather session is stored in server.
When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc.). By default, session variables last until the user closes the browser. So; Session variables hold information about one single user, and are available to all pages in one application.

Fig : For every client session data is stored separately

Uses of session

- **Carrying information as a client travels between pages.**
- **One page data can be stored in session variable and that data can be accessed from other pages.**
- **Session can help to uniquely identify each client from another**
- **Session is mostly used in ecommerce sites where there is shopping cart system.**
- **It helps maintain user state and data all over the application.**

- **It is easy to implement and we can store any kind of object.**
- **Stores client data separately.**
- **Session is secure and transparent from the user.**

Prepare more on your own.

—-------------

a cookie is a small piece of data stored on the client-side which servers use when communicating with clients.

They're used to identify a client when sending a subsequent request. They can also be used for passing some data from one servlet to another.

Persistant Cookies   Non-Persistant Cookies
If we set max age for the Cookie , such type of Cookies are called "Persistant Cookies".
These cookies will be stored permanently in the local file system.
Once the time expires , these Cookies will be disabled automatically.
If we are not setting max age for the Cookie , such type of Cookies are called Non-Persistant/temporary Cookies.
These will be stored in the browsers cache and disabled automatically Once browser will be closed.

Create a Cookie

The *Cookie* class is defined in the ***javax.servlet.http*** package.

To send it to the client, we need to **create one and add it to the response**:

```
Cookie uiColorCookie = new Cookie("color", "red");
response.addCookie(uiColorCookie);
```

Set the Cookie Expiration Date

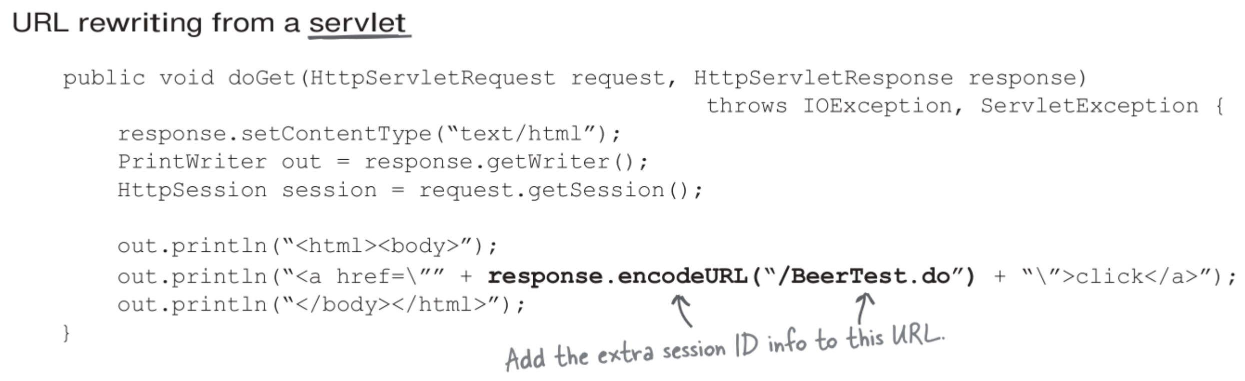
We can set the max age (with a method *maxAge(int)*) which defines how many seconds a given cookie should be valid for:

```
uiColorCookie.setMaxAge(60*60);
```

We set a max age to one hour. After this time, the cookie cannot be used by a client (browser) when sending a request and it also should be removed from the browser cache.

Advantages of Cookies :
It is very easy to implement.
Persist across browser shutdowns, server shutdowns and application redeployments.
If very less Session information is available or if huge no. of end-users are available then the best suitable mechanism is Cookies.
Limitations of Cookie :
To meet security constraints there may be a chance of disabling cookies at client side. In this case Session management by Cookies wan't work .
The maximum no. of Cookies supported by the browser is fixed.(maximum of 30, based on browser)
The size of Cookie is also fixed. Hence we can't store huge amount of information by using Cookies.
The Cookies should be travelled every time across the network. Hence there may be a chance of network over heads. i.e., it impact performance.

URL - Rewriting :



URL rewriting from a servlet

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                          throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();

    out.println("<html><body>");
    out.println("<a href=\"" + response.encodeURL("/BeerTest.do") + "\">click</a>");
    out.println("</body></html>");
}
```

Add the extra session ID info to this URL.

It is another technique followed to mainatian session.Ciikies could be disable or deleted at client side.
Whenever Cookies are disabled at client side , browser unable to see "set-cookie" response header and hence browser unable to get session id and cookies send by the Server .
Due to this browser can't send session id & Cookies to the server and hence server is unable to remember client information across multiple requests . So that Session management fails.
To resolve this we should go for url rewriting technique.
The central idea in this technique is append required Session information to the url , instead of appending to "set-cookie" response header.
Whenever client clicks url for further communication server can get required Session information with the url . So that Server can able to remember client information across multiple requests.
URL Re-writing = URL + session information
OR
URL Re-writing = URL ; jsessionid=123456789

login.html

```
<form action="./redirectone">
<table>
<tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
<tr><td><input type="submit" value="submit"></td></tr>
</table>
</form>
```

UrlRedirectServletOne.java

```
public class UrlRedirectServletOne extends HttpServlet {

@SuppressWarnings("deprecation")
public void doGet(HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException {

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

  String name=request.getParameter("uname");
  HttpSession session=request.getSession();
  session.setAttribute("uname", name);
  out.println("Welcome to Aksahay");

 // out.println("<br> <a href=./redirecttwo?name="
        // +name+ "> click here to get User </a>");

  out.println("<br> <a href= "+response.encodeUrl("./redirecttwo") +
      " >  click here to get User </a> ");


}

}
```

UrlRedirectServletTwo.java

```
public class UrlRedirectServletTwo extends HttpServlet {

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

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

 HttpSession session=request.getSession(false);
 String name=(String)session.getAttribute("uname");

 //String name=request.getParameter("name");
 out.println("Good Morning :"+name);

}

}
```

web.xml

```
<web-app>
  <servlet>
    <servlet-name>UrlRedirectServletOne</servlet-name>
    <servlet-class>session.UrlRedirectServletOne</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>UrlRedirectServletTwo</servlet-name>
    <servlet-class>session.UrlRedirectServletTwo</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>UrlRedirectServletOne</servlet-name>
    <url-pattern>/redirectone</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>UrlRedirectServletTwo</servlet-name>
    <url-pattern>/redirecttwo</url-pattern>
  </servlet-mapping>
</web-app>
```

----

he *HttpSession* is another option for storing user-related data across different requests. A session is a server-side storage holding contextual data.

Data isn't shared between different session objects (client can access data from its session only). It also contains key-value pairs, but in comparison to a cookie, a session can contain object as a value. The storage implementation mechanism is server-dependent.

A session is matched with a client by a cookie or request parameters.

We can obtain an *HttpSession* straight from a request:

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

The above code will create a new session in case it doesn't exist. We can achieve the same by calling:

```
request.getSession(true)
```

In case we just want to obtain existing session and not create a new one, we need to use:

```
request.getSession(false)
```

If we access the JSP page for the first time, then a new session gets created by default. We can disable this behavior by setting the *session* attribute to *false:*

```
<%@ page contentType="text/html;charset=UTF-8" session="false" %>
```

In most cases, a web server uses cookies for session management. When a session object is created, then a server creates a cookie with *JSESSIONID* key and value which identifies a session.

3.2. *Session* Attributes

The session object provides a bunch of methods for accessing (create, read, modify, remove) attributes created for a given user session:

- *setAttribute(String, Object)* which creates or replaces a session attribute with a key and a new value
- *getAttribute(String)* which reads an attribute value with a given name (key)

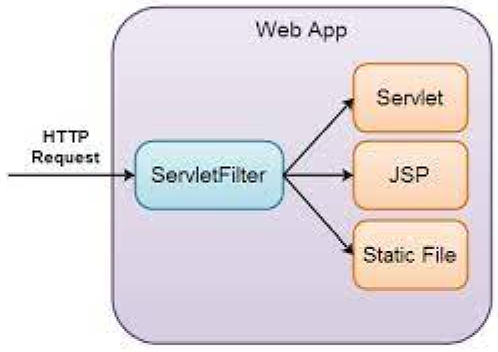
- ***removeAttribute(String)*** **which removes an attribute with a given name**

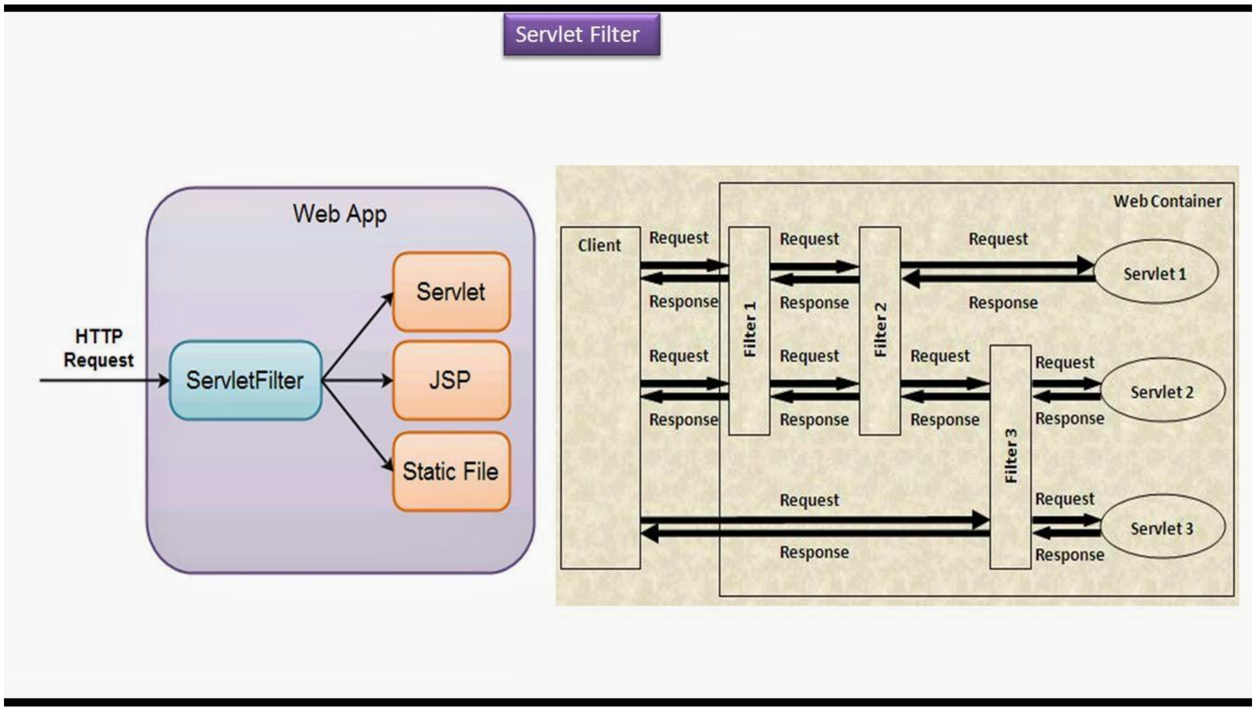
We can also easily check already existing session attributes by calling *getAttributeNames()*.

(ref my code)

Servlet Filters:

Java Servlet Filter is used to intercept the client request and do some pre-processing. It can also intercept the response and do post-processing before sending to the client in web application.





we learned how we can manage session in web application and if we want to make sure that a resource is accessible only when the user session is valid, we can achieve this using servlet session attributes. The approach is simple but if we have a lot of servlets and jsps, then it will become hard to maintain because of redundant code. If we want to change the attribute name in the future, we will have to change all the places where we have session authentication. That’s why we have a servlet filter. Servlet Filters are pluggable java components that we can use to intercept and process requests before they are sent to servlets and response after servlet code is finished and before container sends the response back to the client.

Some common tasks that we can do with servlet filters are:

Logging request parameters to log files.
Authentication and autherization of request for resources.
Formatting of request body or header before sending it to servlet.
Compressing the response data sent to the client.
Alter response by adding some cookies, header information etc.

servlet filters are pluggable and configured in deployment descriptor (web.xml) file. Servlets and filters both are unaware of each other and we can add or remove a servlet filter just by editing web.xml. We can have multiple filters for a single resource and we can create a chain of filters for a single resource in web.xml. We can create a Servlet Filter by implementing javax.servlet.Filter interface.

1.index.html ---entry point

2.Authentication.java -filter bind with the HelloLogin servlet
3.HelloLogin servlet

index.html

```
------
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="hello" method="GET">

  UserName :  <input type="text" name="username" /><br />

  Password :&nbsp; &nbsp; <input type="password"
name="password" /><br />

  <input type="submit" value="login" />

 </form>
</body>
</html>
----
```

2.AuthenticationFilter1.java

```
---------------
```

```
package filtertest;

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

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class AuthenticationFilter1 implements Filter
{

 public void init(FilterConfig config) throws ServletException
 {
  System.out
        .println("-----------------------------------------------------");
  System.out.println(" init method is called in :cosole print "
        + this.getClass().getName());
  System.out
        .println("-----------------------------------------------------");
 }

 public void doFilter(ServletRequest request, ServletResponse
response,
       FilterChain chain) throws IOException, ServletException
```

```
{

 System.out.println(" doFilter method is called in "
         + this.getClass().getName() + "\n\n");

 PrintWriter out = response.getWriter();

 String username = request.getParameter("username");
 String password = request.getParameter("password");
 System.out.println(username);
 if (username.equals("admin") && password.equals("admin"))
 {
  // sends request to next resource
  chain.doFilter(request, response);
 }
 else
 {
  out.print("username or password is not correct!");

 }
}

public void destroy()
{
 // add code to release any resource
 System.out
         .println("---------------------------------------------------");
 System.out.println(" destroy method is called in "
        + this.getClass().getName());
  System.out
        .println("----------------------------------------------------");
 }
}
---
3.HelloLogin.java
-----------
package filtertest;
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 HelloLogin extends HttpServlet{

                private static final long serialVersionUID = 1L;

                public void init() throws ServletException
                {
                 System.out

.println("----------------------------------------------------");
                 System.out.println(" Init method is called in:to print
in console for debug "
                        + this.getClass().getName());
                 System.out

.println("------------------------------------------------------");
                }

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

                 System.out.println(" doGet method is called in "
                        + this.getClass().getName() + "\n\n");
                 System.out

.println("------------------------------------------------------");

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

                 out.print("<br>welcome to helloservlet<br>");

                }

                public void destroy()
                {
                 System.out

.println("------------------------------------------------------");
```

```
                System.out.println(" destroy method is called in "
                        + this.getClass().getName());
                System.out

.println("----------------------------------------------------");
               }

              }
```

4.web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
 metadata-complete="true" version="3.0">
 <display-name>AuthenticationFilterDemo</display-name>
 <description>
        This is a simple web application with a source code
organization
        based on the recommendations of the Application
Developer's Guide.
    </description>
 <servlet>
  <servlet-name>HelloLogin</servlet-name>
  <servlet-class>filtertest.HelloLogin</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>HelloLogin</servlet-name>
  <url-pattern>/hello</url-pattern>
 </servlet-mapping>
 <filter>
  <filter-name>AuthFilter</filter-name>
  <filter-class>filtertest.AuthenticationFilter1</filter-class>
 </filter>
 <filter-mapping>
  <filter-name>AuthFilter</filter-name>
  <url-pattern>/hello</url-pattern>
 </filter-mapping>
</web-app>
```