package com.pioneercoders.webapplicationintroduction;

public class WebIntroduction {

/\*

\* By using java applications we can develop the following applications

\* j2se(java standard edition)

\* j2ee(java enterprise edition)

\* j2me(java micro addition)

\* java standard applications are meant for desktop applications

\* if anybody want to access the desktop applications

\* compulsory he has to install the corresponding related technologies softwares

\* in java if we have to access desktop applications communication can be done through only by sharing his project by extract as jar and user has to take the jar file and use in his computer

\* if any changes made by the product owner again he has to take the jar and remove the previous jar and he has to use ,this is some what difficult

\*

\*

\*we are moving to webapplications following are the advantages

\*Web applications avoid the burden in deploying in each client machine.

Don't have to enforce version check in client machine.

Updates are easier.

Makes bug fixes easier.

No administrator rights checking.

Can access from anywhere.

Platform independent.

Support and maintenance are easier.

Adaptability in mobile applications.

\*

\*

\*/

}

package com.pioneercoders.servletintrodction;

public class ServletIntroduction {

/\*

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

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

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

\*

\*/

}

package com.pioneercoders.servletintrodction;

public class ServletApi {

/\*

\* The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

Servlet

ServletRequest

ServletResponse

RequestDispatcher

ServletConfig

ServletContext

SingleThreadModel

Filter

FilterConfig

FilterChain

ServletRequestListener

ServletRequestAttributeListener

ServletContextListener

ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

GenericServlet

ServletInputStream

ServletOutputStream

ServletRequestWrapper

ServletResponseWrapper

ServletRequestEvent

ServletContextEvent

ServletRequestAttributeEvent

ServletContextAttributeEvent

ServletException

UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

HttpServletRequest

HttpServletResponse

HttpSession

HttpSessionListener

HttpSessionAttributeListener

HttpSessionBindingListener

HttpSessionActivationListener

HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

HttpServlet

Cookie

HttpServletRequestWrapper

HttpServletResponseWrapper

HttpSessionEvent

HttpSessionBindingEvent

HttpUtils (deprecated now)

\*/

}

package com.pioneercoders.servletrequestexample;

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;

/\*\*

\* Servlet implementation class ServletRequestEx

\*/

public class ServletRequestEx extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* Default constructor.

\*/

public ServletRequestEx() {

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

/\*

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

\* ServletRequest inteface provides methods to gather information about the client like getParameter,getContentType

// ServletRequest Interface method are extends to HttpServletRequest interface

\*let see how we are getting client information using request interface methods

\*

\*

\*--------------------------------------

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

\*After recieving request from the client using ServletRequest to generate response and agian send back to the client to send the response we have to use ServletResponse

\*ServletResponse methods inherit to HttpServletResponse

\*

\*/

//following are the methods of servletrequest used get information about the servlet

PrintWriter out=response.getWriter();

out.println("RequestURI :"+request.getRequestURI());

out.println("ContextPath :"+request.getContextPath());

out.println("ServletPath :"+request.getServletPath());

out.println("PathInfo :"+request.getPathInfo());

out.println("QueryString :"+request.getQueryString());

//how to get the user request and how to generate response

//to see the request and response refer ServletResponseInterface project how request and response is generating

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}

}

package com.pioneercoders.servletresponseexample;

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;

/\*\*

\* Servlet implementation class FormServlet

\*/

public class FormServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public FormServlet() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

//response.getWriter().append("Served at: ").append(request.getContextPath());

// TODO Auto-generated method stub

//user always sends with url pattern mapping only ,we are hiding internal details of our application

//see web.xml for url pattern

//<url-pattern>/FormServlet</url-pattern>

//url pattern may be anything

//form.jsp user hits the submit button action ="FormServlet" tomcat checks is there any urlpattern available with this name and sends the request to FormServlet ,FormServlet recieve the client data and generate response

//if url pattern is not matching we will get 404 error

//by request methods we recieve client data

//user always enters string data only

//taking the input field values based on name attribute in html

String user = request.getParameter("user");

String pass= request.getParameter("pass");

//after recieving client information we have to send the response using httpservletresponse

//im sending text as response

response.setContentType("text");

//to send the response to the user

//im sending the response to the user with his username and password

PrintWriter out = response.getWriter();

out.println("PathInfo :"+request.getPathInfo());

out.println("QueryString :"+request.getQueryString());

out.println("User credentials are "+" "+user+" "+pass);

String name= getServletName();

System.out.println(name);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}

package com.pioneercoders.servletconfigexample;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class ServletConfigEx

\*/

public class ServletConfigEx extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public ServletConfigEx() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

//response.getWriter().append("Served at: ").append(request.getContextPath());

/\*

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

\* With in the Servlet we can access Servlet initialization parameters by using ServletConfig object.

\* during servlet instantiation if any init parameters are there to give init parameters information to the servlet we have to use ServletConfig

\* String getInitParameter(String name): returns a String value initialized parameter, or NULL if the parameter does not exist.

Enumeration getInitParameterNames(): returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.

ServletContext getServletContext(): returns a reference to the ServletContext

String getServletName(): returns the name of the servlet instance

\*/

//by using servlet config we get the servlet information like servlet and init parameters

ServletConfig cfg= getServletConfig();

cfg.getInitParameter("email");

String servletname= getServletName();

PrintWriter out = response.getWriter();

out.println("Servletname"+servletname);

out.println(cfg.getInitParameter("email"));

System.out.println("Servlet name"+" "+servletname);

System.out.println("Init values "+" "+cfg.getInitParameter("email"));

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}

package com.pioneercoders.servletcontextexample;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class ServletContextEx

\*/

public class ServletContextEx extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public ServletContextEx() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

//response.getWriter().append("Served at: ").append(request.getContextPath());

//if we want to get any init parameters for any servlet while loading we should go for init parameters using servletconfig

//but we want any parameters common to all servlets and jsps we have to get from web.xml the solution is context-param using servletcontext

//For every Web application a ServletContext object is created by the web container. ServletContext object is used to get configuration information from Deployment Descriptor(web.xml) which will be available to any servlet or JSPs that are part of the web app.

//context parameters available to every servlet

//ex: jdbc details are required for common for every servlet so that we have to keep in context-param in web.xml

//we can context param values either by servletconfig or servletcontext

/\*ServletContext app = getServletContext();

OR

ServletContext app = getServletConfig().getServletContext();

\*/

/\*methods

\* Object getAttribute(String name) returns the container attribute with the given name, or NULL if there is no attribute by that name.

String getInitParameter(String name) returns parameter value for the specified parameter name, or NULL if the parameter does not exist

Enumeration getInitParameterNames() returns the names of the context's initialization parameters as an Enumeration of String objects

void setAttribute(String name,Object obj) set an object with the given attribute name in the application scope

void removeAttribute(String name) removes the attribute with the specified name from the application context

\*

\*/

ServletContext context = getServletContext();

PrintWriter out= response.getWriter();

out.println("ServletContext available parameters are "+ " "+context.getInitParameter("jdbc"));

System.out.println(context.getInitParameter("jdbc"));

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}

package com.pioneercoders.requestdispatcherexample;

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;

/\*\*

\* Servlet implementation class RequestDispatcherEx

\*/

public class RequestDispatcherEx extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public RequestDispatcherEx() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

//response.getWriter().append("Served at: ").append(request.getContextPath());

//if we dont mention any method in the form while submiting the form by default it goes to doGet method in servlet

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

Methods of RequestDispatcher

RequestDispatcher interface provides two important methods

Methods Description

void forward(ServletRequest request, ServletResponse response) forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server

void include(ServletRequest request, ServletResponse response) includes the content of a resource (servlet, JSP page, HTML file) in the response

How to get an Object of RequestDispatcher

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

RequestDispatcher rs = request.getRequestDispatcher("form.jsp");

rs.forward(request,response);

\*

\*/

/\*In this example, we will show you how RequestDispatcher is used to forward or include response of a resource in a Servlet. Here we are using form.jsp to get id and name from the user, RequestDispatcherEx Servlet will validate the name entered by the user, if the user has entered "srinadh" as password, then he will be forwarded to Welcome Servlet else the user will stay on the form.jsp page and an error message will be displayed.

\*

\*

\*/

//after submitting the form the request comes to RequestDispatcherEx get the id and name and validate if user enter the name srinadh ,name matches then he will be redirected WelcomeServlet and gives success response otherwise he will redirected to form.jsp with error message

String id= request.getParameter("id");

String name = request.getParameter("name");

if(name.equals("srinadh")){

//if name equal to srinadh then forwarding the resource or request to Welcome Servlet to give success response

request.getRequestDispatcher("WelcomeMessage").forward(request, response);

}else

{

//user entered does not match with srinadh then again forward to form.jsp and print error msg

PrintWriter out= response.getWriter();

out.println("<font color='red'><b>You have entered incorrect password</b></font>");

request.getRequestDispatcher("index.html");

}

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}

package com.servletintroduction;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

/\*\*

\* Servlet implementation class SessionTrackingEx

\*/

public class SessionTrackingEx extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public SessionTrackingEx() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

/\*

\* We all know that HTTP is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

Session Management is a mechanism used by the Web container to store session information for a particular user. There are four different techniques used by Servlet application for session management. They are as follows:

Cookies

Hidden form field

URL Rewriting

HttpSession

Session is used to store everything that we can get from the client from all the requests the client makes.

\* The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

\* we are discussing about httpsession

\*

\* HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object. Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object.

\*

\* On client's first request, the Web Container generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.

The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.

The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

\*/

//how to get a httpsession

//HttpSession session = request.getSession();

//getSession() method returns a session if the session already exist it return the existing session else create a new session

//u can see session example in below post method

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

System.out.println("hiiiiii");

String name = request.getParameter("user");

String pass= request.getParameter("pass");

//to store the user credentials http protocol doesn't remember the client data we have to store by using httpsession

//user clicks the submit button by submitting the form with details in session.jsp

//if password matches create the session and store user name

//and send session data available to other Servlet also

if(pass.equals("sri")){

//creating a session

HttpSession session = request.getSession();

session.setAttribute("user", name);

//if password matches, user name available with in the session until logged out of the application we have to show his username

response.sendRedirect("SessionEx");

}

}

}

package com.servletintroduction;

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;

import javax.servlet.http.HttpSession;

/\*\*

\* Servlet implementation class SessionEx

\*/

public class SessionEx extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public SessionEx() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession();

String user = (String)session.getAttribute("user");

System.out.println(user);

out.println("Hello "+user);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}

}

package com.pioneercoders.servlet.filters;

public class FilterTheory {

/\*

\* Filters are compontents that you can use and configure to perform some filtering tasks.

\* Filter is used for pre-processing of requests and post-processing of responses.

\* You can have any number of filters for pre-processing of a request and post-processing of a response.

\* Filters are configured in the deployment descriptor of a web application.

\*

\* How Filters Works?

When a request reaches the Web Container, it checks if any filter has URL patterns that matches the requested URL.

The Web Container locates the first filter with a matching URL pattern and filter's code is executed.

If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns left.

If no error occurs, the request passes to the target servlet. Hence we know, that the request will be passed to the target servlet only when all the related Filters are successfully executed.

The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.

At last the response will be passed to the Web Container which passes it to the client.

\*

\*

\* More about Filter API

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

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

void init(FilterConfig filterConfig): invoked by the web container to indicate to a filter that it is being placed into service.

void doFilter(ServletRequest request, ServletResponse response, FilterChain chain): invoked by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.

void destroy(): invoked by the web container to indicate to a filter that it is being taken out of service.

\*

\*

\*

\* First filter recieves the request and authenticate and based on the url pattern pass the request to Servlet here it is passing to LoginServlet

\*

\*

\*

\*/

}

package com.pioneercoders.listener;

public class ListenerTheory {

/\* What is the need of ServletContextListener?

Sometime we may have a requirement that some code should run before a web application starts.

For example, we need to create a database connection so that web application can use it whenever it performs some operations and when application shuts down, we can close database connection.

\*

\* We can write our own listeners also by implementing ServletContextListener and HttpListener

\*

\* ex: Java EE specification provides an interface named ServletContextListener which serves our purpose. ServletContextListener listens to the lifecycle events of a servlet context. This interface gets notified whenever an application with which listener is associated starts up and shuts down. Here is what javadoc says about it.

Implementations of this interface receive notifications about changes to the servlet context of the web application they are part of. To receive notification events, the implementation class must be configured in the deployment descriptor for the web application.

If you want to listen when web application starts, use contextInitialized(ServletContextEvent event) method.

Notification that the web application initialization process is starting. All ServletContextListeners are notified of context initialization before any filter or servlet in the web application is initialized.

If you want to listen when web application stops, use contextDestroyed(ServletContextEvent event) method.

Notification that the servlet context is about to be shut down. All servlets and filters have been destroyed() before any ServletContextListeners are notified of context destruction.

Create a listener as follows

MyServletContextListenerJava

package com.pioneercoders

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

public class MyServletContextListener implements ServletContextListener {

public void contextInitialized(ServletContextEvent event) {

System.out.println("context initialized");

}

public void contextDestroyed(ServletContextEvent event) {

}

}

package com.pioneercoders;

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

public class MyServletContextListener implements ServletContextListener {

public void contextInitialized(ServletContextEvent event) {

System.out.println("context initialized");

}

public void contextDestroyed(ServletContextEvent event) {

}

}

\* some of the important methods of ServletRequestListener

\*

\* Java EE specification provides an interface called ServletRequestListener which receives notifications whenever a new request is about to come to the web application. This is what javadoc says about it.

A ServletRequest is defined as coming into scope of a web application when it is about to enter the first servlet or filter of the web application, and as going out of scope as it exits the last servlet or the first filter in the chain.

requestInitialized(ServletRequestEvent event) :

Receives notification that a ServletRequest is about to come into scope of the web application.

requestDestroyed(ServletRequestEvent event) :

Receives notification that a ServletRequest is about to go out of scope of the web application.

\*

\*

\* How to configure listener in web.xml

\* <listener>

<listener-class>com.pioneercoders.MyRequestListener</listener-class>

</listener>

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}