Java Servlet Tutorial with Examples for Beginners

In the last article, we learned about **Java Web Application** and looked into core concepts of Web Applications such as **Web Server**, **Web Client**, **HTTP** and **HTML**, **Web Container** and how we can use **Servlets** and **JSPs** to create web application. We also created our first Servlet and JSP web application and executed it on tomcat server.

This tutorial is aimed to provide more details about servlet, core interfaces in Servlet API, Servlet 3.0 annotations, life cycle of Servlet and at the end we will create a simple login servlet application.

1. Servlet Overview

Servlet is J2EE server driven technology to create web applications in java. The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing our own servlets.

All servlets must implement the `javax.servlet.Servlet` interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the `GenericServlet` class provided with the Java Servlet API. The `HttpServlet` class provides methods, such as `doGet()` and `doPost()`, for handling HTTP-specific services.

Most of the times, web applications are accessed using HTTP protocol and thats why we mostly extend HttpServlet class.

2. Common Gateway Interface (CGI)

Before introduction of Servlet API, CGI technology was used to create dynamic web applications. CGI technology has many drawbacks such as creating separate process for each request, platform dependent code (C, C++), high memory usage and slow performance.

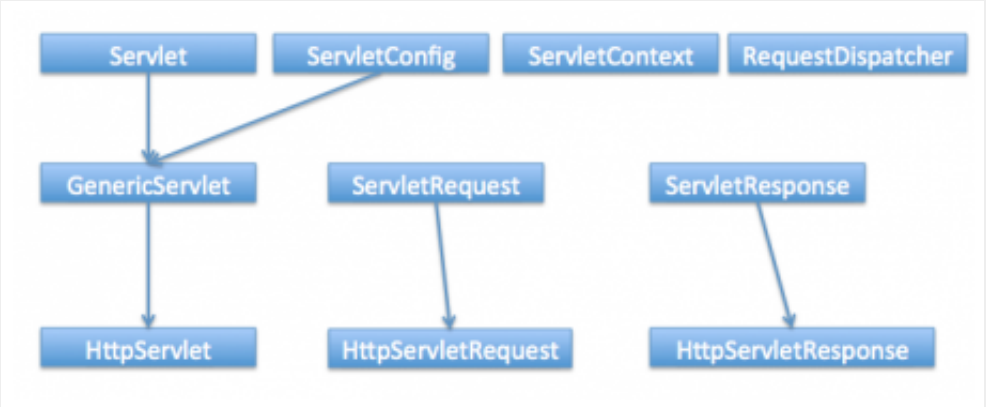
3. CGI vs Servlet

Servlet technology was introduced to overcome the shortcomings of CGI technology.

- Servlets provide better performance that CGI in terms of processing time, memory utilization because servlets uses benefits of multithreading and for each request a new thread is created, that is faster than loading creating new Object for each request with CGI.
- Servlets and platform and system independent, the web application developed with Servlet can be run on any standard web container such as Tomcat, JBoss, Glassfish servers and on operating systems such as Windows, Linux, Unix, Solaris, Mac etc.
- Servlets are robust because container takes care of life cycle of servlet and we don't need to worry about memory leaks, security, garbage collection etc.
- Servlets are maintainable and learning curve is small because all we need to take care is business logic for our application.

4. Servlet API Hierarchy

`javax.servlet.Servlet` is the base **interface** of Servlet API. There are some other interfaces and classes that we should be aware of when working with Servlets. Also with Servlet 3.0 specs, servlet API introduced use of annotations rather than having all the servlet configuration in deployment descriptor. In this section, we will look into important Servlet API interfaces, classes and annotations that we will use further in developing our application. The below diagram shows servlet API hierarchy.



A. Servlet Interface

`javax.servlet.Servlet` is the base interface of **Java Servlet API**. Servlet interface declares the life cycle methods of servlet. All the servlet classes are required to implement this interface. The methods declared in this interface are:

i. **public abstract void init(ServletConfig paramServletConfig) throws ServletException** – This is the very important method that is invoked by servlet container to initialized the servlet and ServletConfig parameters. The servlet is not ready to process client request until unless **init()** method is finished executing. This method is called only once in servlet lifecycle and make Servlet class different from normal java objects. We can extend this method in our servlet classes to initialize resources such as DB Connection, Socket connection etc.
ii. **public abstract ServletConfig getServletConfig()** – This method returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet. We can use this method to get the init parameters of servlet defines in deployment descriptor (web.xml) or through annotation in Servlet 3. We will look into ServletConfig interface later on.
iii. **public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException** – This method is responsible for processing the client request. Whenever servlet container receives any request, it creates a new thread and execute the `service()` method by passing request and response as argument. Servlets usually run in multi-threaded environment, so it's developer responsibility to keep shared resources thread-safe using **synchronization**.
iv. **public abstract String getServletInfo()** – This method returns string containing information about the servlet, such as its author, version, and copyright. The string returned should be plain text and can't have markups.
v. **public abstract void destroy()** – This method can be called only once in servlet life cycle and used to close any open resources. This is like finalize method of a java class.

B. ServletConfig Interface

`javax.servlet.ServletConfig` is used to pass configuration information to Servlet. Every servlet has it's own ServletConfig object and servlet container is responsible for instantiating this object. We can provide servlet init parameters in **web.xml** file or through use of `WebInitParam` annotation. We can use **getServletConfig()** method to get the ServletConfig object of the servlet.

The important methods of ServletConfig interface are:

i. **public abstract ServletContext getServletContext()** – This method returns the ServletContext object for the servlet. We will look into ServletContext interface in next section.
ii. **public abstract Enumeration getInitParameterNames()** – This method returns the Enumeration of name of init parameters defined for the servlet. If there are no init

parameters defined, this method returns empty enumeration.

iii. **public abstract String getInitParameter(String paramString)** – This method can be used to get the specific init parameter value by name. If parameter is not present with the name, it returns null.

c. ServletContext interface

`javax.servlet.ServletContext` interface provides access to web application variables to the servlet. The ServletContext is unique object and available to all the servlets in the web application. When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using **<context-param>** element. We can get the ServletContext object via the getServletContext() method of ServletConfig. Servlet engines may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the URL path namespace of the host.

Some of the important methods of ServletContext are:

i. **public abstract ServletContext getContext(String uripath)** – This method returns ServletContext object for a particular uripath or null if not available or not visible to the servlet.

ii. **public abstract URL getResource(String path) throws MalformedURLException** – This method return URL object allowing access to any content resource requested. We can access items whether they reside on the local file system, a remote file system, a database, or a remote network site without knowing the specific details of how to obtain the resources.

iii. **public abstract InputStream getResourceAsStream(String path)** – This method returns an input stream to the given resource path or null if not found.

iv. **public abstract RequestDispatcher getRequestDispatcher(String urlpath)** – This method is mostly used to obtain a reference to another servlet. After obtaining a RequestDispatcher, the servlet programmer forward a request to the target component or include content from it.

v. **public abstract void log(String msg)** – This method is used to write given message string to the servlet log file.

vi. **public abstract Object getAttribute(String name)** – Return the object attribute for the given name. We can get enumeration of all the attributes using **public abstract Enumeration getAttributeNames()** method.

vii. **public abstract void setAttribute(String paramString, Object paramObject)** – This method is used to set the attribute with application scope. The attribute will be accessible to all the other servlets having access to this ServletContext. We can remove an attribute using **public abstract void removeAttribute(String paramString)** method.

viii. **String getInitParameter(String name)** – This method returns the String value for the

init parameter defined with name in web.xml, returns null if parameter name doesn't exist. We can use **Enumeration getInitParameterNames()** to get enumeration of all the init parameter names.

ix. **boolean setInitParameter(String paramString1, String paramString2)** – We can use this method to set init parameters to the application.

Note: Ideally the name of this interface should be ApplicationContext because it's for the application and not specific to any servlet. Also don't get confused it with the servlet context passed in the URL to access the web application.

D. ServletRequest interface

ServletRequest interface is used to provide client request information to the servlet. Servlet container creates ServletRequest object from client request and pass it to the servlet service() method for processing.

Some of the important methods of ServletRequest interface are:

i. **Object getAttribute(String name)** – This method returns the value of named attribute as Object and null if it's not present. We can use `getAttributeNames()` method to get the enumeration of attribute names for the request. This interface also provide methods for setting and removing attributes.

ii. **String getParameter(String name)** – This method returns the request parameter as String. We can use `getParameterNames()` method to get the enumeration of parameter names for the request.

iii. **String getServerName()** – returns the hostname of the server.

iv. **int getServerPort()** – returns the port number of the server on which it's listening.

The child interface of `ServletRequest` is `HttpServletRequest` that contains some other methods for session management, cookies and authorization of request.

E. ServletResponse interface

ServletResponse interface is used by servlet in sending response to the client. Servlet container creates the ServletResponse object and pass it to servlet service() method and later use the response object to generate the HTML response for client.

Some of the important methods in HttpServletResponse are:

i. **void addCookie(Cookie cookie)** – Used to add cookie to the response.

ii. **void addHeader(String name, String value)** – used to add a response header with the given name and value.

iii. **String encodeURL(java.lang.String url)** – encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

iv. **String getHeader(String name)** – return the value for the specified header, or null if this header has not been set.
v. **void sendRedirect(String location)** – used to send a temporary redirect response to the client using the specified redirect location URL.
vi. **void setStatus(int sc)** – used to set the status code for the response.

F. RequestDispatcher interface

RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in the same context. We can also use this to include the content of another resource to the response. This interface is used for servlet communication within the same context.

There are two methods defined in this interface:

i. **void forward(ServletRequest request, ServletResponse response)** – forwards the request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
ii. **void include(ServletRequest request, ServletResponse response)** – includes the content of a resource (servlet, JSP page, HTML file) in the response.

We can get RequestDispatcher in a servlet using ServletContext *getRequestDispatcher(String path)* method. The path must begin with a / and is interpreted as relative to the current context root.

G. GenericServlet class

GenericServlet is an **abstract class** that implements Servlet, ServletConfig and Serializable interface. GenericServlet provide default implementation of all the Servlet life cycle methods and ServletConfig methods and makes our life easier when we extend this class, we need to override only the methods we want and rest of them we can work with the default implementation. Most of the methods defined in this class are only for easy access to common methods defined in Servlet and ServletConfig interfaces.

One of the important method in GenericServlet class is no-argument init() method and we should override this method in our servlet program if we have to initialize some resources before processing any request from servlet.

H. HTTPServlet class

HTTPServlet is an abstract class that extends GenericServlet and provides base for creating HTTP based web applications. There are methods defined to be overridden by subclasses for different HTTP methods.

i. doGet(), for HTTP GET requests
ii. doPost(), for HTTP POST requests
iii. doPut(), for HTTP PUT requests
iv. doDelete(), for HTTP DELETE requests

5. Servlet Attributes

Servlet attributes are used for inter-servlet communication, we can set, get and remove attributes in web application. There are three scopes for servlet attributes – **request scope**, **session scope** and **application scope**.

ServletRequest, **HttpSession** and **ServletContext** interfaces provide methods to get/set/remove attributes from request, session and application scope respectively.

Servlet attributes are different from init parameters defined in **web.xml** for ServletConfig or ServletContext.

6. Annotations in Servlet 3

Prior to Servlet 3, all the servlet mapping and it's init parameters were used to defined in web.xml, this was not convenient and more error prone when number of servlets are huge in an application.

Servlet 3 introduced use of **java annotations** to define a servlet, filter and listener servlets and init parameters.

Some of the important Servlet API annotations are:

A. **WebServlet** – We can use this annotation with Servlet classes to define init parameters, loadOnStartup value, description and url patterns etc. At least one URL pattern MUST be declared in either the value or urlPattern attribute of the annotation, but not both. The class on which this annotation is declared MUST extend HttpServlet.
B. **WebInitParam** – This annotation is used to define init parameters for servlet or filter, it contains name, value pair and we can provide description also. This annotation can be used within a WebFilter or WebServlet annotation.
C. **WebFilter** – This annotation is used to declare a servlet filter. This annotation is processed by the container during deployment, the Filter class in which it is found will be created as per the configuration and applied to the URL patterns, Servlets and DispatcherTypes. The annotated class MUST implement javax.servlet.Filter interface.
D. **WebListener** – The annotation used to declare a listener for various types of event, in a given web application context.

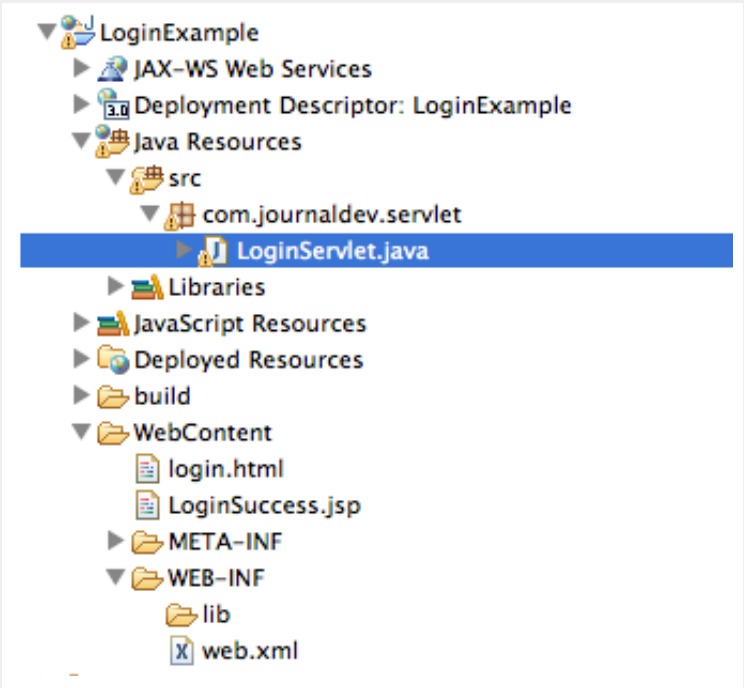
Note: We will look into Servlet Filters and Listeners in future articles, in this article our focus is

to learn about base interfaces and classes of Servlet API.

7. Servlet Login Example

Now we are ready to create our login servlet example, in this example I will use simple HTML, JSP and servlet that will authenticate the user credentials. We will also see the use of ServletContext init parameters, attributes, ServletConfig init parameters and RequestDispatcher include() and response sendRedirect() usage.

Our Final Dynamic Web Project will look like below image. I am using Eclipse and Tomcat for the application, the process to create dynamic web project is provided in Java Web Applications tutorial.



Here is our login HTML page, we will put it in the welcome files list in the web.xml so that when we launch the application it will open the login page.

login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Login Page</title>
</head>
<body>

<form action="LoginServlet" method="post">

Username: <input type="text" name="user">
<br>
Password: <input type="password" name="pwd">
<br>
<input type="submit" value="Login">
</form>
```

```
</body>
</html>
```

If the login will be successful, the user will be presented with new JSP page with login successful message. JSP page code is like below.

LoginSuccess.jsp

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Login Success Page</title>
</head>
<body>
<h3>Hi Pankaj, Login successful.</h3>
<a href="login.html">Login Page</a>
</body>
</html>
```

Here is the web.xml deployment descriptor file where we have defined servlet context init parameters and welcome page.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http
  <display-name>LoginExample</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <context-param>
  <param-name>dbURL</param-name>
  <param-value>jdbc:mysql://localhost/mysql_db</param-value>
  </context-param>
  <context-param>
  <param-name>dbUser</param-name>
  <param-value>mysql_user</param-value>
  </context-param>
  <context-param>
  <param-name>dbUserPwd</param-name>
  <param-value>mysql_pwd</param-value>
  </context-param>
</web-app>
```

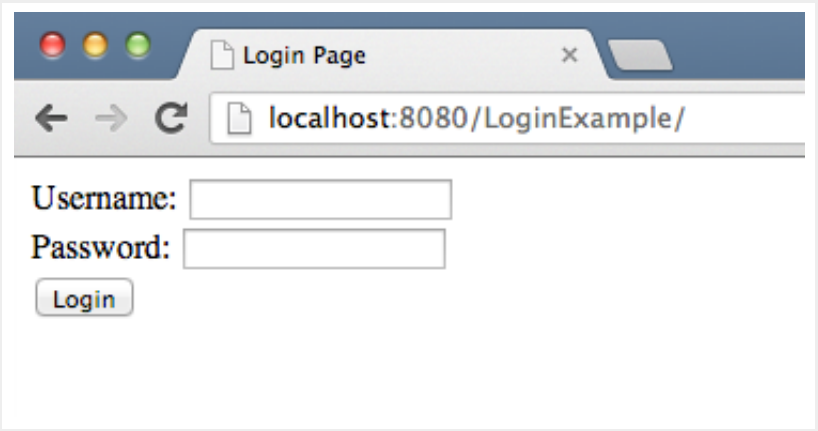
Here is our final Servlet class for authenticating the user credentials, notice the use of annotations for Servlet configuration and ServletConfig init parameters.

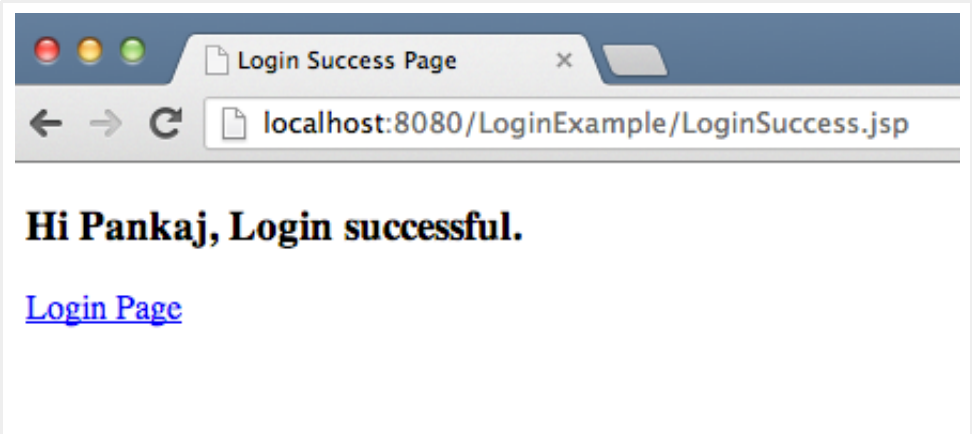
LoginServlet.java

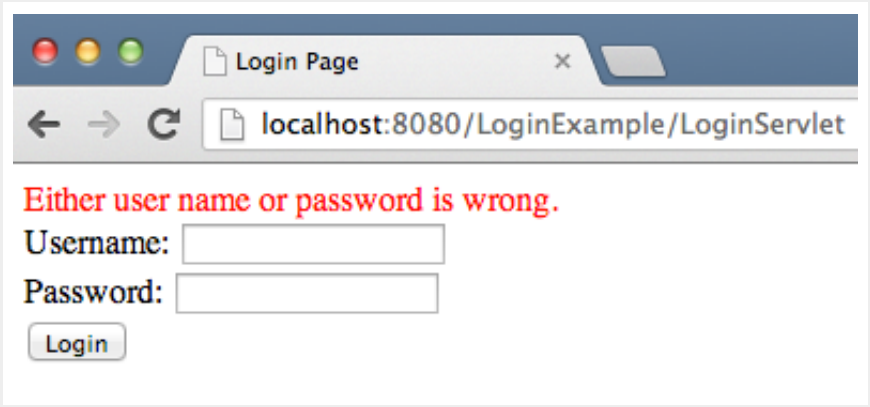
```
package com.journaldev.servlet;

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

import javax.servlet.RequestDispatcher;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class LoginServlet
 */
@WebServlet(
        description = "Login Servlet",
        urlPatterns = { "/LoginServlet" },
        initParams = {
                @WebInitParam(name = "user", value = "Pankaj"),
                @WebInitParam(name = "password", value = "journaldev")
        })
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;


    public void init() throws ServletException {
        //we can create DB connection resource here and set it to Servlet
        if(getServletContext().getInitParameter("dbURL").equals("jdbc:mysq
                getServletContext().getInitParameter("dbUser").equals("mys
                getServletContext().getInitParameter("dbUserPwd").equals("
        getServletContext().setAttribute("DB_Success", "True");
        else throw new ServletException("DB Connection error");
    }


    protected void doPost(HttpServletRequest request, HttpServletResponse

        //get request parameters for userID and password
        String user = request.getParameter("user");
        String pwd = request.getParameter("pwd");

        //get servlet config init params
        String userID = getServletConfig().getInitParameter("user");
        String password = getServletConfig().getInitParameter("password");
        //logging example
        log("User="+user+"::password="+pwd);

        if(userID.equals(user) && password.equals(pwd)){
            response.sendRedirect("LoginSuccess.jsp");
        }else{
            RequestDispatcher rd = getServletContext().getRequestDispatche
            PrintWriter out= response.getWriter();
            out.println("<font color=red>Either user name or password is w
            rd.include(request, response);

        }

    }

}
```

Below screenshots shows the different pages based on the user password combinations for successful login and failed logins.







Download Servlet Login Example Project
3246 downloads

Thats all for Servlet tutorial for beginners, in next tutorial we will look into Session Management, Servlet Filters and Listeners.

Update: Check out next article in series, Session in Servlet.