# WEB

|  |  |
| --- | --- |
| **TYPE** | **DESCRIPTION** |
| Web Server | It refers to infrastructure that handles HTTP [requests] and [responses]. Web Server serves static pages HTML. For example, Apache Tomcat, Jetty, Web Logic, JBoss |
| Application server | It refers to framework for handling Web applications. Application server is responsible for generating dynamic content like JSO, Servlet/EJB |
| Web container | Is part of application server that implement and manages servlets. For example, JSP. It handles all communications tasks. We don’t need to build any server socket to listen for any requests from web server. It’s handled by Web Container |
| Servlet container | It is a component that handles lifecycle of servlet. |
|  |  |
|  |  |

Note: Apache Tomcat serves as both

* web server-static web pages
* application server (Servlets, JSP)

# SERVLET

## History

Before JSP, CGI (**C**ommand **G**ateway **I**nterface) was used. GCI Technology has many drawbacks such as creating separate process for each [request]. Platform dependent code (C, C++), high memory usage

**JSP vs CGI**

JSP pros:

* Platform independent
* It does not create a new process for each [request]. As a result JSP has better performance
* JSP release 3 supports annotations. Before release 3 all settings was supposed to be set up in web.xml

## Overview

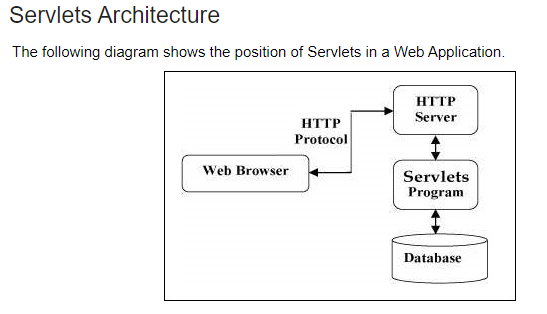
*Web servers are good with static content of HTML. But they don’t know how generate dynamic content or how save data into databases*. There are several programming languages for dynamic content like PHP, Ruby, Java Servlets and JSP

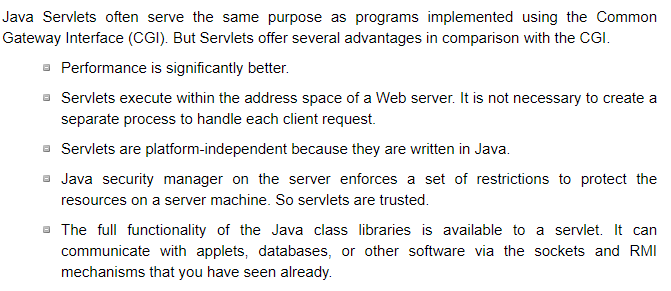
*Servlets and JSP* are server-side technologies to extend capability of web servers

**Servlet** is a Java class that extends HttpServlet and file definitions for build web application for use on the JVM. Servlet is responsible for generate html and send it in [response]. If [response] has a lot of data

it’s error – prone and hard read to use it. **JSP (Java Server Page)** come to solve this problem

Servlet are programs that run on a Web server and act as a middle layer between Web browser and application (database/ application on HTTP server)



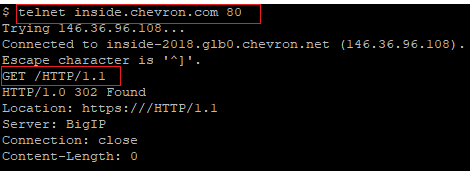


**Servlet tasks:**

* Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page.
* Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
* Process the data and generate the results. This process may require talking to a database, invoking a Web service, or computing the response directly.
* Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
* Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

# HTTP





What does [GET / HTTP/1.1] mean

* **GET** – you instruct that you send GET request
* **/** - use this page
* **1.1** -use 1.1 version of HTTP protocol

**HTTP Methods** are instructions to the web server what to do with the resource you are requesting.

<https://www.javatpoint.com/get-vs-post>

There are 5

* **GET** – is not secured. It’s limited by length, because data is sent in header. But it’s fast. GET does not change the state of resource.

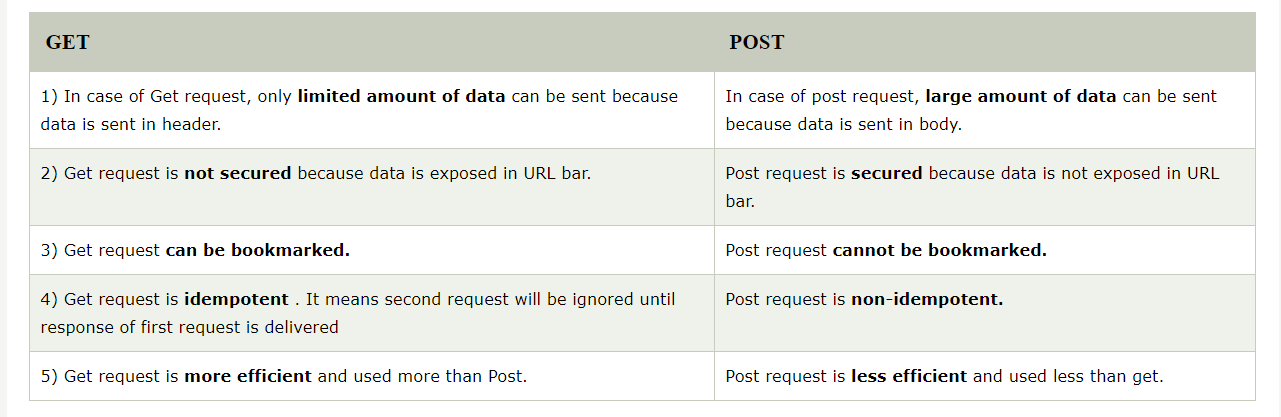
The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.



* **POST** – update the resources. It is slow, compared to [get], because data is sent in body. But it’s secured

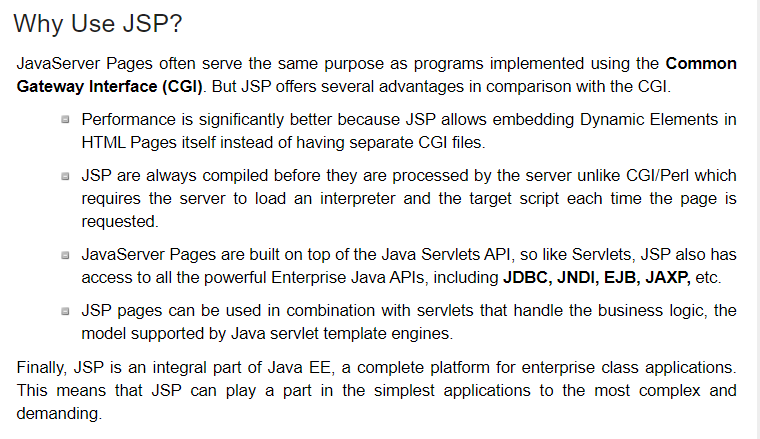
A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

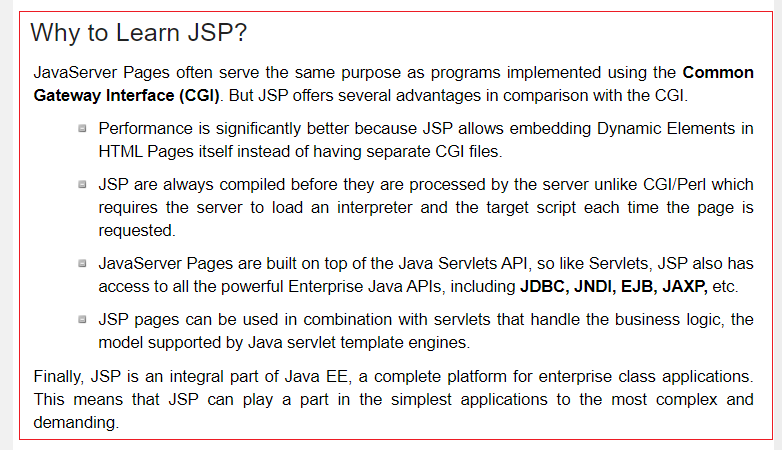
* **PUT** -set the content. It’s confusing when use PUT and POST. A rule is – when update existing content need to use POST when a new then PUT
* **DELETE** - remove the requested resources
* **HEAD** - is similar to GET. A difference is that it’s just returns code and headers. It can be used to verify that big resource exists without the need to download it. If the client previously downloaded the resource it can check the [last-modifier] header, which will allow the client to see it has most up to date copy



# JSP

**JSP (Java Server Page)** – is server side programming technologies that helps to create HTML pages with dynamic content



 twofold

**Scriplet** -is elements of JSP .<% for each %>

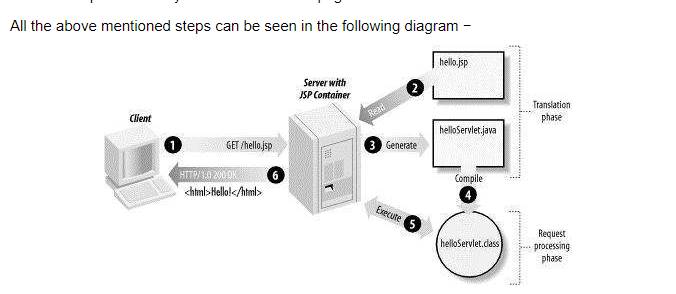
# HOW WORKS JSP

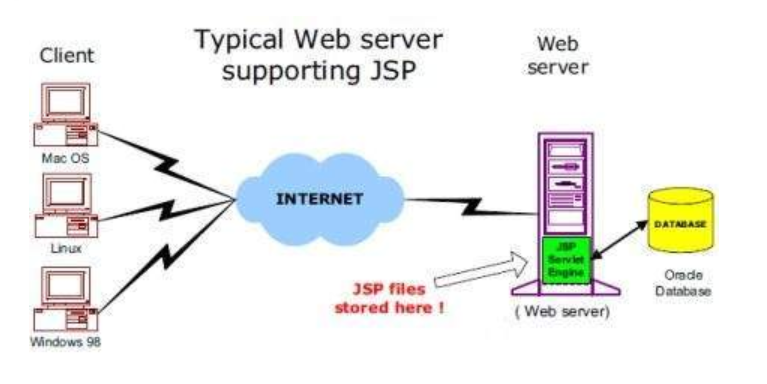
**HOW WORKS JSP:**

* **1 step.** Your browser sends an HTTP request to the web server

WebServer recognizes that HTTP request is for JSP page and forwards it to JSP engine. This done by using URL or JSP page that ends by [.jsp] extension instead of [.html]

* **2 step.** The JSP engine loads the JSP page from disk
* **3 step.**  Converts it to servlet content. This code also implements the corresponding dynamic behavior of the page.
* **4 step.** JSP engine compiles the servlet into executable class and forwards the original request to a servlet engine.
* **5 step** Server engine loads the Servlet class and executes it. During execution the Servlet produces output in HTML format. The output is passed on web server by servlet engine inside HTTP response.
* **6 step** Web Server forwards HTTP response in terms of static HTML contents





Servlet defines that web application uses a ***deployment descriptor*** that is called **web.xml**.

**Web.xml** is the deployment descriptor of the web application and contains mapping for servlets and some settings

Web.xml is located under path [webapp/WEB-INF/web.xml]

Deployment descriptor(web.xml) defines:

* Servlets
* And how they are configured and used by servlet container

Deployment descriptor(web.xml) has 2 parts

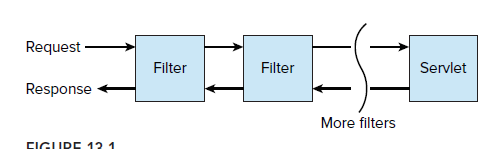
* Filters and mapping
* Servlet and mapping

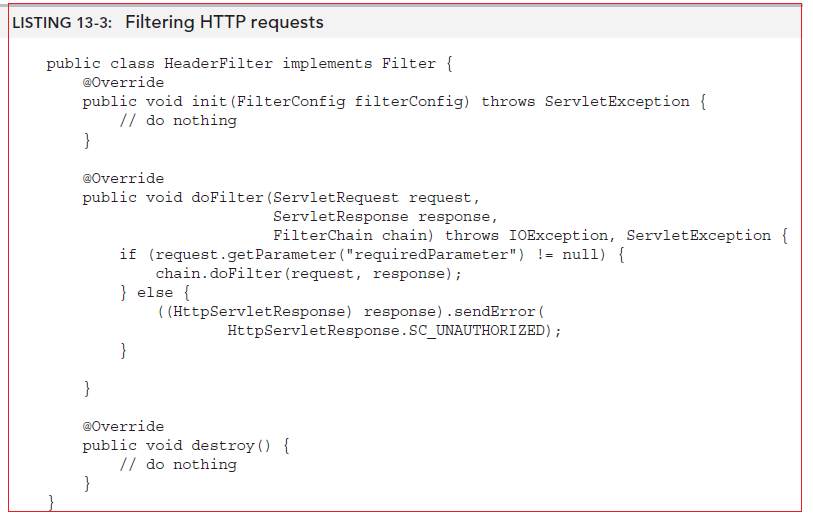
# FILTER INTERFACE

Filter interface define 3 methods

* init()
* doFilter(…) – it gives the filter opportunity to inspect request and decide whether reject request or allow it to go forward
* destroy()

note: you can have a few filters





# WAR FILE

**WAR file** – is **W**eb **Ar**chive file. It’s specifically used for creating web app

# WEBSERVER

**Webserver** – is a software that can process the client request and send back response to the client.

**HTTP** (HyperText Transfer Protocol) - is the communication protocol between server and client. HTTP runs on the top of TCP/IP protocol

**HTTP [Request]** is instance of  **javax.servlet.http.HttpServletRequest** object. Each time client request page JSP engine creates new object to represent that request

Parts of **HTTP Request**:

* **HTTP Methods** - Get, Post, Put, Delete
* **URL** - Page to access
* **Form Parameters** – similar to arguments in java method

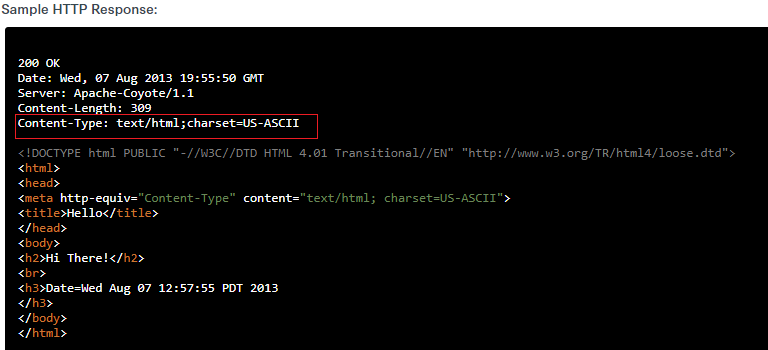
**HTTP [Response]** is instance of  **javax.servlet.http.HttpServletRequest** object. Just as the server creates the request object, it also creates an object to represent the response to the client.

Parts of **HTTP Response**:

* **Status code** – integer value. 200 – ok, 404 – page not found, 403 – access forbidden
* **Content type** – text, html, image, pdf
* **Content** – actual data
* **MIME type (M**ultipurpose **I**nternet **M**ail **E**xtensions) - **or Content Type**: If you see above sample HTTP response header, it contains tag “Content-Type”. It’s also called MIME type and server sends it to the client to let them know the kind of data it’s sending. It helps the client in rendering the data for the user. Some of the most used mime types are ***text/html, text/xml, application/xml*** etc.

**Web server** – is a software that can process the client requests and send requests back

When a Web server responds to a HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document. A typical response looks like this



There are a few popular Web Servers:

* *Tomcat*- lightweight
* *Jetty* – lightweight. You can create and run servers with no configuration (just use default) for [GET] request. For other, like [POST],[PUT] you need to config it
* *Play Framework* –is a new web server that does not use servlet technologies
* *WebLogic* – for big projects

# URL

**URL** (**Universal Resource Locator**) - it’s used to locate the server and resource. Every resource on the web has its own unique address. Let’s see parts of the URL with an example.

**http://localhost:8080/FirstServletProject/jsps/hello.jsp**

**http://** – This is the first part of URL and provides the communication protocol to be used in server-client communication.

**localhost** – The unique address of the server, most of the times it’s the hostname of the server that maps to unique IP address. Sometimes multiple hostnames point to same IP addresses and [web server virtual host](https://www.journaldev.com/1456/how-to-install-apache-php-and-mysql-on-mac-os-x) takes care of sending a request to the particular server instance.

**8080** – This is the port on which server is listening, it’s optional and if we don’t provide it in URL then request goes to the default port of the protocol. Port numbers 0 to 1023 are reserved ports for well-known services, for example, 80 for HTTP, 443 for HTTPS, 21 for FTP, etc.

**FirstServletProject/jsps/hello.jsp** – Resource requested from server. It can be static html, pdf, JSP, servlets, PHP etc.

# TOMCAT

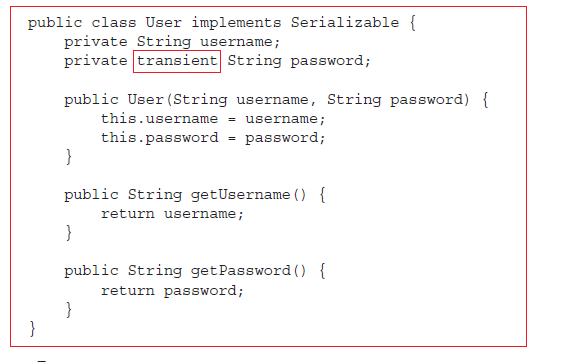
Tomcat is built from several components, including one for servlet container and one for processing JSP.

* Servlet container is called **Catalina**
* JSP is called **Jasper**

# SERIALIAZATION

Serialization – is a way of “exporting” Java objects from the JVM. The serialized data can be written to disk or another I/O interface rather than to the network

**Transient** is a key word. It means if you have serialized object, but have fields that do not want to be serialized when writing data to a stream, you can apply [transient] modifier to the field declaration. When a filed is a [deserialized], the field will be **null**.



**JSON** is a JavaScript Object Notation. It’s another serialization approach, similar to using XML and XSD. It uses a human readable approach which can be parsed by number of languages

**Jackson** -is a library for Java to parse JSON files