How to Install Apache Tomcat 9 (on Windows, macOS, Ubuntu) and Get Started with Java Servlet Programming

TABLE OF CONTENTS (HIDE)

This practical can be completed in a 3-hour session.

This installation and configuration guide is applicable to Tomcat 9, and possibly the earlier versions. Take note that Tomcat 9 requires JDK 8 and later.

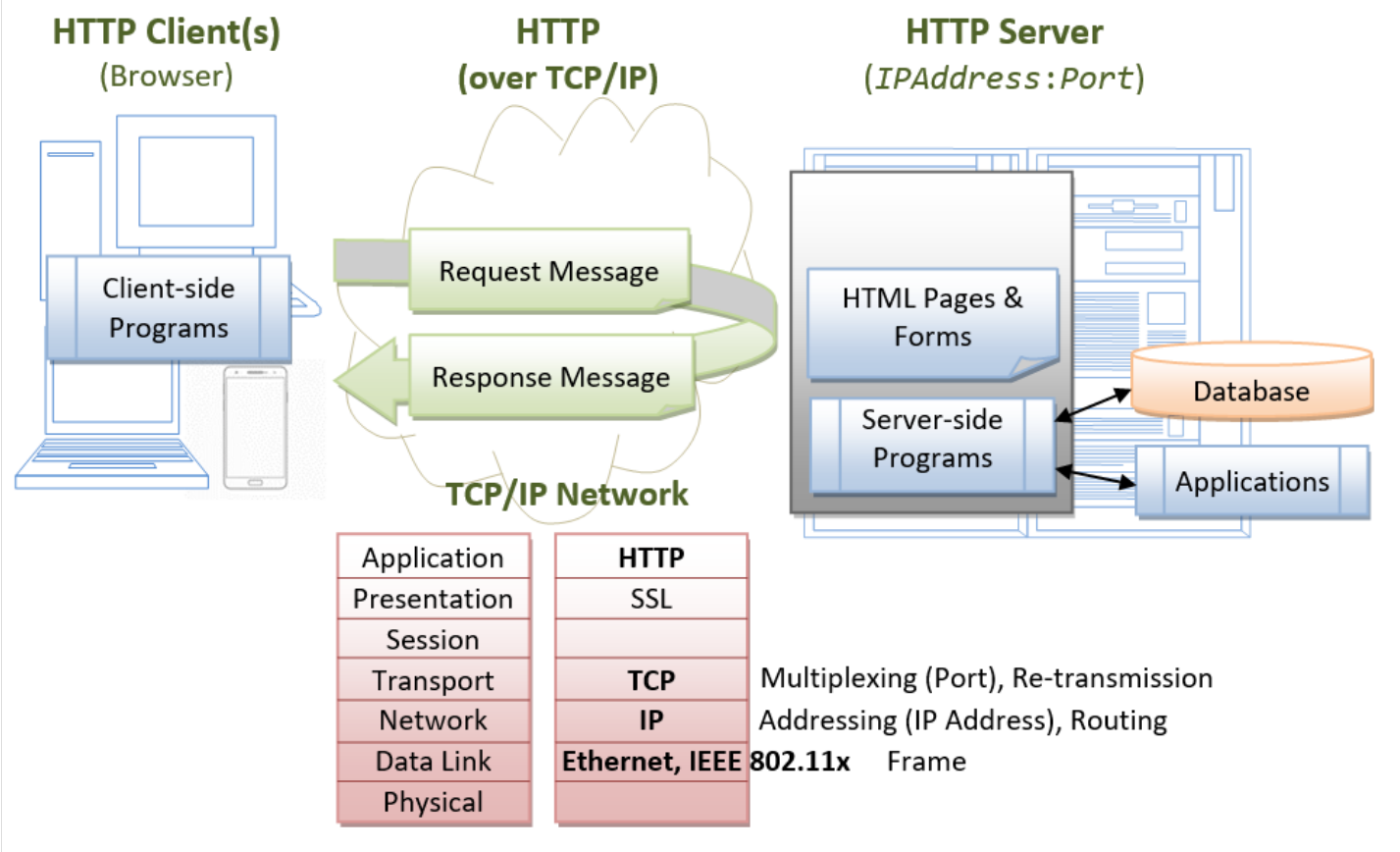
1. Introduction

1.1 Web Application (Webapp)

A *web application* (or *webapp*), unlike standalone application, runs over the Internet. Examples of webapps are google, amazon, facebook and twitter.

A webapp is typically a *3-tier* (or *multi-tier*) *client-server database application* run over the Internet as illustrated in the diagram below. It comprises five components:

1. **HTTP Server**: E.g., Apache HTTP Server, Apache Tomcat Server, Microsoft Internet Information Server (IIS), nginx, Google Web Server (GWS), and others.
2. **HTTP Client (or Web Browser)**: E.g., Internet Explorer (MSIE), FireFox, Chrome, Safari, and others.
3. **Database**: E.g., Open-source MySQL, PostgreSQL, Apache Derby, mSQL, SQLite, OpenOffice's Base; Commercial Oracle, IBM DB2, SAP SyBase, Microsoft SQL Server, Microsoft Access; and others.
4. **Client-Side Programs**: Could be written in HTML Form, JavaScript, and others.
5. **Server-Side Programs**: Could be written in Java Servlet/JSP, ASP, PHP, Perl, Python, JavaScript, and others.

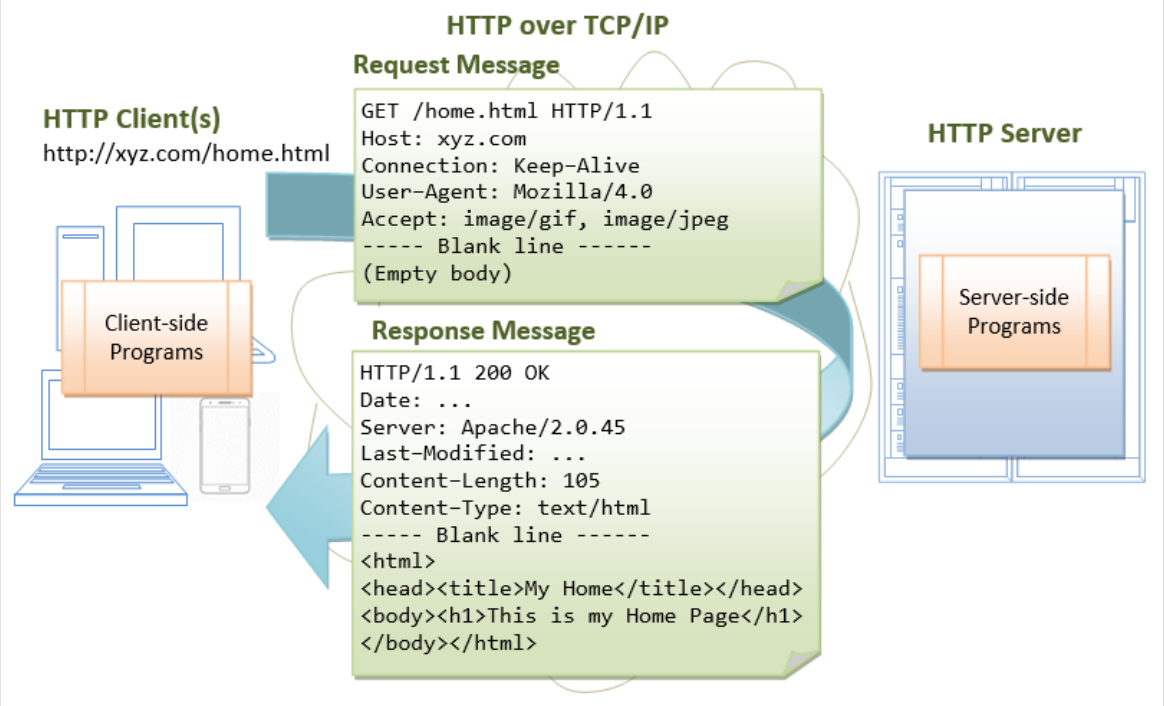


A typical *use case* is:

1. (client-to-server) A user, via a web browser (HTTP client), issues a URL request to an HTTP server to start a webapp.
2. (server-to-client) The HTTP server returns an HTML form (client-side program), which is loaded and rendered in the client's browser.
3. (client-to-server) The user fills up the query criteria inside the form and submits the form. This sends the query parameters to a server-side program.
4. (server-to-client) The server-side program receives the query parameters, queries the database based on these parameters, and returns the query result to the client-side program.
5. (client) The client-side program receives the query result and displays on the browser.
6. The process repeats for the next request-response.

1.2 Hypertext Transfer Protocol (HTTP)

- HTTP is an *application layer* protocol runs over TCP/IP. The IP provides support for routing and addressing (via a unique IP address for machines connected to the Internet); while TCP supports multiplexing via 64K ports from port number 0 to 65535. The default port number assigned to HTTP is TCP port 80. (Notes: TCP Port numbers below 1024 are reserved for popular protocols such as HTTP, FTP, SMTP; Port numbers 1024 and above could be used for applications.)
- HTTP is an *asynchronous request-response application-layer protocol*. A client sends a request message to the server. The server then returns a response message to the client. In other words, HTTP is a *pull* protocol, a client pulls a page from the server (instead of server pushes pages to the clients).
- The syntax of the message is defined in the HTTP specification.



1.3 Apache Tomcat HTTP Server

Apache Tomcat is a Java-capable HTTP server, which could execute special Java programs known as "Java Servlet" and "Java Server Pages (JSP)". Tomcat is an *open-source* project, under the "Apache Software Foundation" (which also provides the most use, open-source, industrial-strength Apache HTTP Server). The mother site for Tomcat is http://tomcat.apache.org. Alternatively, you can find tomcat via the Apache mother site @ http://www.apache.org.

Tomcat was originally written by James Duncan Davison (then working in Sun Microsystem) in 1998, based on an earlier Sun's server called Java Web Server (JWS). It began at version 3.0 after JWS 2.1 it replaced. Sun subsequently made Tomcat open-source and gave it to Apache.

The various Tomcat releases are:

1. Tomcat 3.0 (1999): Reference Implementation (RI) for Servlet 2.2 and JSP 1.1.
2. Tomcat 4.1 (Sep 2002): RI for Servlet 2.3 and JSP 1.2.
3. Tomcat 5.0 (Dec 2003): RI for Servlet 2.4 and JSP 2.0.
4. Tomcat 6.0 (Feb 2007): RI for Servlet 2.5 and JSP 2.1.
5. Tomcat 7.0 (Jan 2011): RI for Servlet 3.0, JSP 2.2 and EL 2.2.
6. Tomcat 8.0 (Jun 2014): RI for Servlet 3.1, JSP 2.3, EL 3.0 and WebSocket 1.0. Tomcat 8.5 (June 2016) supports HTTP/2, OpenSSL, TLS virtual hosting and JASPIC 1.1.
7. Tomcat 9.0 (Jan 2018): RI for Servlet 4.0, JSP 2.3, EL 3.0, WebSocket 1.0, JASPIC 1.1.
8. Tomcat 10.0 (???):

2. How to Install Tomcat and Get Started with Java Servlet Programming

2.1 STEP 0: Create a Directory to Keep all your Works

I shall assume that you have created a directory called "`c:\myWebProject`" (for Windows) or "`~\myWebProject`" (for macOS) in your earlier exercises. Do it otherwise. This step is important; otherwise, you will be out-of-sync with this article and will not be able to find your files later.

2.2 STEP 1: Download and Install Tomcat

For Windows

1. Goto http://tomcat.apache.org ⇒ Under "Tomcat 9.0.{xx} Released", where {xx} is the latest update number ⇒ Click "Download" ⇒ Under "9.0.{xx}" ⇒ Binary Distributions ⇒ Core ⇒ "**zip**" (e.g., "`apache-tomcat-9.0.{xx}.zip`", about 11 MB).
2. UNZIP (right-click ⇒ Extract All) the downloaded file into your project directory "`c:\myWebProject`". Tomcat shall be unzipped into directory "`c:\myWebProject\apache-tomcat-9.0.{xx}`".
3. For **EASE OF USE**, we shall shorten and rename this directory to "`c:\myWebProject\tomcat`".

Take note of Your Tomcat Installed Directory. Hereafter, I shall refer to the Tomcat installed directory as `<TOMCAT_HOME>`.

For macOS

1. Goto http://tomcat.apache.org ⇒ Under "Tomcat 9.0.{xx} Released", where {xx} is the latest update number ⇒ Click "Download" ⇒ Under "9.0.{xx}"⇒ Binary distribution ⇒ Core ⇒ "**`tar.gz`**" (e.g., "`apache-tomcat-9.0.{xx}.tar.gz`", about 10.5 MB).
2. To install Tomcat:
   a. Double-click the downloaded tarball (e.g., "`apache-tomcat-9.0.{xx}.tar.gz`") to expand it into a folder (e.g., "`apache-tomcat-9.0.{xx}`").
   b. Move the extracted folder (e.g., "`apache-tomcat-9.0.{xx}`") to your project directory "`~/myWebProject`".
   c. For **EASE OF USE**, we shall shorten and rename this folder to "tomcat", i.e., "`~/myWebProject/tomcat`".

Take note of Your Tomcat Installed Directory. Hereafter, I shall refer to the Tomcat installed directory as `<TOMCAT_HOME>`.

For Ubuntu

Read "How to Install Tomcat on Ubuntu". You need to switch between these two articles.

For academic learning, I recommend "`zip`" (or "`tar.gz`") package, as you could simply delete the entire directory when Tomcat is no longer needed (without running any un-installer). You are free to move or rename the Tomcat's installed directory. You can install (unzip) multiple copies of Tomcat in the same machine.

Tomcat's Sub-Directories

Take a quick look at the Tomcat installed directory. It contains the these sub-directories:

- **`bin`**: contains the *binaries* and *scripts* (e.g., `startup.bat` and `shutdown.bat` for Windows; `startup.sh` and `shutdown.sh` for Unixes and macOS).
- **`conf`**: contains the system-wide *configuration* files, such as `server.xml`, `web.xml`, and `context.xml`.
- **`webapps`**: contains the *webapps* to be deployed. You can also place the WAR (Webapp Archive) file for deployment here.
- **`lib`**: contains the Tomcat's system-wide library JAR files, accessible by all webapps. You could also place external JAR file (such as MySQL JDBC Driver) here.
- **`logs`**: contains Tomcat's log files. You may need to check for error messages here.
- **`work`**: Tomcat's working directory used by JSP, for JSP-to-Servlet conversion.

2.3 STEP 2: Create an Environment Variable `JAVA_HOME`

(For Windows)

You need to create an *environment variable* (system variable available to all applications) called "`JAVA_HOME`", and set it to your JDK installed directory.

Follow the steps HERE!

(For macOS)

Skip this step. No need to do anything.

2.4 STEP 3: Configure the Tomcat Server

The Tomcat configuration files, in XML format, are located in the "`conf`" sub-directory of your Tomcat installed directory, e.g. "`c:\myWebProject\tomcat\conf`" (for Windows) or "`~/myWebProject/tomcat/conf`" (for macOS). The important configuration files are:

1. `server.xml`
2. `web.xml`
3. `context.xml`

Make a BACKUP of the configuration files before you proceed!!!

Step 3(a) "`conf\server.xml`" - Set the TCP Port Number

Use a programming text editor (e.g., Sublime Text, Atom) to open the configuration file "`server.xml`".

The default TCP port number configured in Tomcat is 8080, you may choose any number between 1024 and 65535, which is not used by existing applications. We shall choose 9999 in this article. (For production server, you should use port 80, which is pre-assigned to HTTP server as the default port number.)

Locate the following lines (around Line 69) that define the HTTP connector, and change `port="8080"` to `port="9999"`.

```
<!-- A "Connector" represents an endpoint by which requests are received
      and responses are returned. Documentation at :
      Java HTTP Connector: /docs/config/http.html
      Java AJP  Connector: /docs/config/ajp.html
      APR (HTTP/AJP) Connector: /docs/apr.html
      Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="9999" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

Step 3(b) "conf\web.xml" - Enable Directory Listing

Again, use a programming text editor to open the configuration file "`web.xml`".

We shall enable directory listing by changing "`listings`" from "`false`" to "`true`" for the "`default`" servlet. This is handy for test system, but not for production system for security.

Locate the following lines (around Line 122) that define the "`default`" servlet; and change the "`listings`" from "`false`" to "`true`".

```
<servlet>
  <servlet-name>default</servlet-name>
  <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <init-param>
    <param-name>listings</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Step 3(c) "conf\context.xml" - Enabling Automatic Reload

We shall add the attribute `reloadable="true"` to the `<Context>` element to enable automatic reload after code changes. Again, this is handy for test system but not recommended for production, due to the overhead of detecting changes.

Locate the `<Context>` start element (around Line 19), and change it to:

```
<Context crossContext="true" reloadable="true">
   ......
   ......
</Context>
```

2.5 STEP 4: Start Tomcat Server

The Tomcat's executable programs and scripts are kept in the "`bin`" sub-directory of the Tomcat installed directory.

Step 4(a) Start Server

For Windows

I shall assume that Tomcat is installed in "`c:\myWebProject\tomcat`". Launch a CMD shell and issue:

```
c:                            // Change drive
cd \myWebProject\tomcat\bin   // Change directory to your Tomcat's binary directory
startup                       // Run startup.bat to start tomcat server
```

For macOS

I assume that Tomcat is installed in "`~/myWebProject/tomcat`". To start the Tomcat server, open a new "Terminal" and issue:

```
cd ~/myWebProject/tomcat/bin  // Change directory to your Tomcat's binary directory
./catalina.sh run             // Run catalina.sh to start tomcat server
```

A *new* Tomcat console window appears (with Java's coffee-cup logo as icon). Study the messages on the console. Look out for the Tomcat's port number. Double check that Tomcat is running on port 9999 as configured.

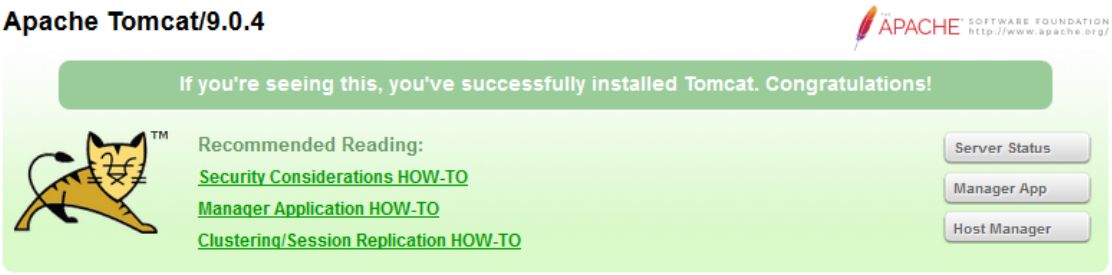
Error messages will be sent to this console. `System.out.println()` issued by your Java servlets will also be sent to this console.

```
.............
.............
xxxxx INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-9999"]
xxxxx INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
xxxxx INFO [main] org.apache.catalina.startup.Catalina.start Server startup in [1325] ms
```

(Skip Unless ...) Cannot Start Tomcat: Read "How to Debug".

Step 4(b) Start a Client to Access the Server

Start a browser (Firefox, Chrome) as an HTTP client. Issue URL "`http://localhost:9999`" to access the Tomcat server's welcome page. The hostname "`localhost`" (with IP address of `127.0.0.1`) is meant for local loop-back testing within the same machine. For users on the other machines over the net, they have to use the server's IP address or DNS domain name in the form of "`http://`*`serverHostnameOrIPAddress`*`:9999`".



(Optional) Try issuing URL `http://localhost:9999/examples` to view the servlet and JSP examples. Try running some of the servlet examples.

Step 4(c) Shutdown Server

For Windows

You can shutdown the tomcat server by either:

1. Press Ctrl-C on the Tomcat console; OR
2. Run "`<TOMCAT_HOME>\bin\`**`shutdown.bat`**" script. Open a new "cmd" and issue:

```
c:                          // Change the current drive
cd \myWebProject\tomcat\bin  // Change directory to your Tomcat's binary directory
shutdown                    // Run shutdown.bat to shutdown the server
```

For macOS

To shutdown the Tomcat server:

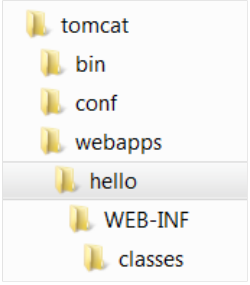
1. Press Control-C (NOT Command-C) on the Tomcat console; OR
2. Run the "`<TOMCAT_HOME>/bin/shutdown.sh`" script. Open a new "Terminal" and issue:

```
cd ~/myWebProject/tomcat/bin  // Change current directory to Tomcat's bin directory
./shutdown.sh                 // Run shutdown.sh to shutdown the server
```

WARNING: You MUST properly shutdown the Tomcat. DO NOT kill the CAT by pushing the window's "CLOSE" button.

2.6 STEP 5: Develop and Deploy a "Hello-world" WebApp

Step 5(a) Create the Directory Structure for your WebApp



Let's call our first webapp "`hello`". Goto Tomcat's "`webapps`" sub-directory and create the following directory structure for your webapp "`hello`" (as illustrated). The directory names are case-sensitive!!

1. Under Tomcat's "`webapps`", create your webapp's *root* directory "`hello`" (i.e., "`<TOMCAT_HOME>\webapps\hello`").
2. Under "`hello`", create a sub-directory "`WEB-INF`" (case sensitive, a "dash" not an underscore) (i.e., "`<TOMCAT_HOME>\webapps\hello\WEB-INF`").
3. Under "`WEB-INF`", create a sub-sub-directory "`classes`" (case sensitive, plural) (i.e., "`<TOMCAT_HOME>\webapps\hello\WEB-INF\classes`").

You need to keep your web resources (e.g., HTMLs, CSSs, images, scripts, servlets, JSPs) in the proper directories:

- "`hello`": The is called the *context root* (or *document base directory*) of your webapp. You should keep all your HTML files and resources visible to the web users (e.g., HTMLs, CSSs, images, scripts, JSPs) under this *context root*.
- "`hello/WEB-INF`": This directory, although under the context root, is *not visible* to the web users. This is where you keep your application's web descriptor file "`web.xml`".

- "`hello/WEB-INF/classes`": This is where you keep all the Java classes such as servlet class-files.

You need to RE-START your Tomcat server to pick up the `hello` webapp. Check the Tomcat's console to confirm that "`hello`" application has been properly deployed:

```
......
xxxxx INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory
Deploying web application directory [xxx\webapps\hello]
xxxxx INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory
Deployment of web application directory [xxx\webapps\hello] has finished in [38] ms
......
```

You can issue the following URL to access the web application "`hello`":

```
http://localhost:9999/hello
```

You should see the directory listing of the directory "`<TOMCAT_HOME>\webapps\hello`", which shall be empty at this point of time. Take note that we have earlier enabled directory listing in "`web.xml`". Otherwise, you will get an error "404 Not Found".

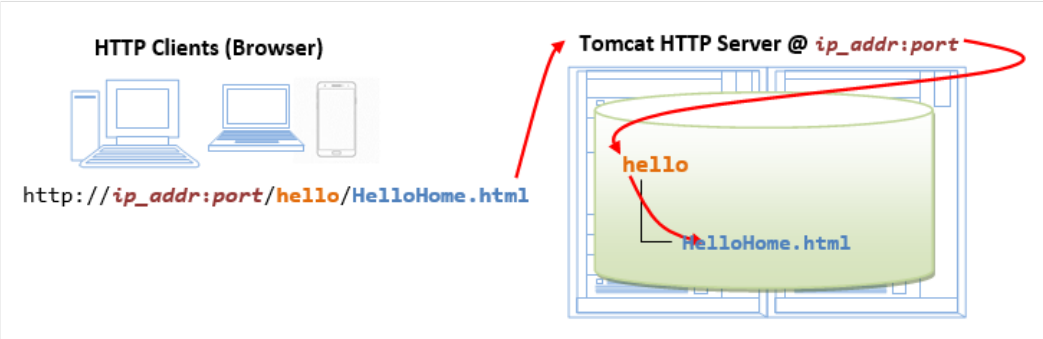
Step 5(b) Write a Welcome Page

Create the following HTML page and save as "`HelloHome.html`" in your webapp's root directory "`hello`".

```
<!DOCTYPE html>
<html>
  <head><title>My Home Page</title></head>
  <body>
    <h1>Hello, world!</h1>
    <p>My Name is so and so. This is my HOME.</p>
  </body>
</html>
```

You can browse this page by issuing this URL:

```
http://localhost:9999/hello/HelloHome.html
```



Alternatively, you can issue an URL to your webapp's root "`hello`":

```
http://localhost:9999/hello
```

The server will return the directory listing of your base directory. You can then click on "`HelloHome.html`".

Rename "`HelloHome.html`" to "`index.html`", and issue a directory request again:

```
http://localhost:9999/hello
```

Now, the server will redirect the directory request to "`index.html`", if the root directory contains an "`index.html`", instead of serving the directory listing.

Rename "`index.html`" back to "`HelloHome.html`", so that you could see get directory listing for convenience.

You can check out the home page of your peers by issuing:

```
http://YourPeerHostnameOrIPAddr:9999/hello
http://YourPeerHostnameOrIPAddr:9999/hello/HelloHome.html
```

with a valid "*YourPeerHostnameOrIPAddr*", provided that your peer has started his tomcat server and his firewall (and the network) does not block your access. You can use command such as "`ipconfig`" (Windows), "`ifconfig`" (macOS and Unix) to find your IP address.

(Skip Unless...) The likely errors are "Unable to Connect", "Internet Explorer cannot display the web page", and "404 File Not Found". Read "How to Debug" section.

2.7 STEP 6: Write a "Hello-world" Java Servlet

A *servlet* is Java program that runs inside a Java-capable HTTP Server, such as Apache Tomcat. A web user invokes a servlet by issuing an appropriate URL from a web browser (HTTP client).

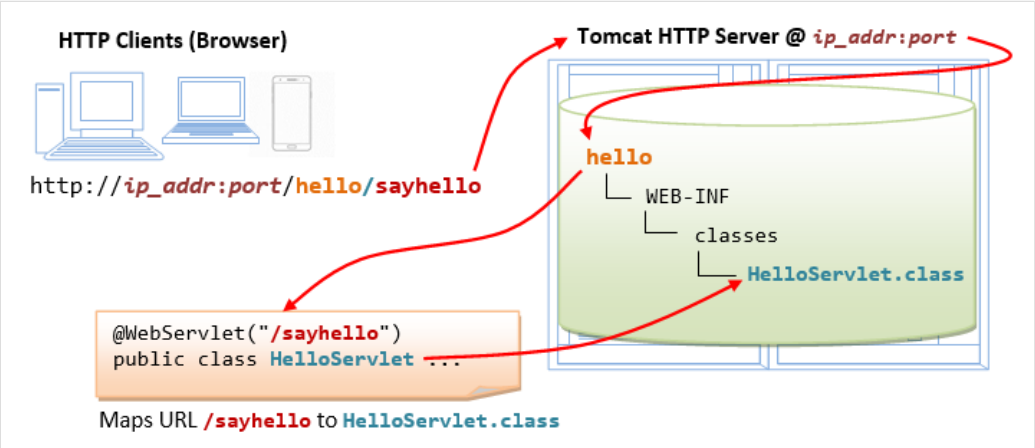
Before you proceed, I shall assume that you are familiar with Java Programming and have installed the followings:

1. JDK (Read "How to install JDK and Get Started").
2. A programming text editor, such as Sublime Text or Atom.

Step 6(a) Write a "Hello-world" Java Servlet

A Java servlet is a Java program that runs inside a HTTP server. A web user invokes a servlet by issuing a URL from a browser (or HTTP client).

In this example, we are going to write a Java servlet called `HelloServlet`, which says "Hello, world!". We will configure such that web users can invoke this servlet by issuing URL `http://ip_addr:port/hello/sayhello` from their browser, as illustrated:



Write the following source codes called "`HelloServlet.java`" and save it under your application "`classes`" directory (i.e., "`<TOMCAT_HOME>\webapps\hello\WEB-INF\classes\HelloServlet.java`"). This servlet says "Hello", echoes some request information, and prints a random number upon each request.

```
1  // To save as "<TOMCAT_HOME>\webapps\hello\WEB-INF\classes\HelloServlet.java"
2  import java.io.*;
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import javax.servlet.annotation.*;
6
7  @WebServlet("/sayhello")   // Configure the request URL for this servlet (Tomcat 7/Servlet 3.0 upwards)
8  public class HelloServlet extends HttpServlet {
9
10    // The doGet() runs once per HTTP GET request to this servlet.
11    @Override
12    public void doGet(HttpServletRequest request, HttpServletResponse response)
13          throws IOException, ServletException {
14
15       // Set the response MIME type of the response message
16       response.setContentType("text/html");
17       // Allocate a output writer to write the response message into the network socket
18       PrintWriter out = response.getWriter();
19
20       // Write the response message, in an HTML page
21       out.println("<!DOCTYPE html>");
22       out.println("<html>");
23       out.println("<head><title>Hello, World</title></head>");
24       out.println("<body>");
25       out.println("<h1>Hello, world!</h1>");  // says Hello
26       // Echo client's request information
27       out.println("<p>Request URI: " + request.getRequestURI() + "</p>");
28       out.println("<p>Protocol: " + request.getProtocol() + "</p>");
29       out.println("<p>PathInfo: " + request.getPathInfo() + "</p>");
30       out.println("<p>Remote Address: " + request.getRemoteAddr() + "</p>");
31       // Generate a random number upon each request
32       out.println("<p>A Random Number: <strong>" + Math.random() + "</strong></p>");
33       out.println("</body></html>");
34       out.close();  // Always close the output writer
35    }
36 }
```

Take note that in Line 7, we configure this `HelloServlet` to URL "`/sayhello`" via annotation `@WebServlet("/sayhello")`, which is applicable to Tomcat 7 onwards. In other words, the full URL shall be `http://ip_addr:port/hello/sayhello` to trigger this `HelloServlet`.

Step 6(b) Compiling the Servlet (DIFFICULT)

We need the Java Servlet API to compile the servlet. Servlet API is NOT part of JDK. Tomcat provides a copy in `<TOMCAT_HOME>/lib/servlet-api.jar`. We need to include this JAR file in the compilation via the `-cp` (classpath) option as follows:

(For Windows)

```
// Assume that Tomcat is installed in c:\myWebProject\tomcat
// Change directory to the Java source directory
c:
cd \myWebProject\tomcat\webapps\hello\WEB-INF\classes

// Compile with servlet API library
javac -cp .;c:\myWebProject\tomcat\lib\servlet-api.jar HelloServlet.java
```

(For macOS)

```
// Assume that Tomcat is installed in ~/myWebProject/tomcat
// Change directory to the Java source directory
cd ~/myWebProject/tomcat/webapps/hello/WEB-INF/classes

// Compile with Servlet API - Need to use $HOME instead of ~ in the "javac" command
javac -cp .:$HOME/myWebProject/tomcat/lib/servlet-api.jar HelloServlet.java
```

The output of the compilation is "`HelloServlet.class`".

Use your "File Explorer" to check the "`webapps/hello/WEB-INF/classes`" folder to make sure that "`HelloServlet.class`" has been created in the right place.

Step 6(c) Invoke the Servlet

Restart your Tomcat Server (no need but just in case ...).

To invoke this servlet, start a browser, and issue the request URL configured as follows:

```
http://localhost:9999/hello/sayhello
```

You shall see the output of the servlet displayed in your web browser.

Refresh the browser, you shall see a new random number upon each refresh. In other word, the `doGet()` method of the servlet runs once per request.

View Page Source

(For Firefox and Chrome) Right-click the page ⇒ "View Page Source" to look at the output received by the web browser (which is returned by the server). Take note that the web browser receives only the output of the servlet (generated via the `out.println()` statements). The client has no access to the servlet source codes (which may contain confidential information).

(For macOS's Safari browser) You need to enable "Developer Menu" under the "Preferences" to enable the "View Source" menu.

```
<!DOCTYPE html>
<html>
<head><title>Hello, World</title></head>
<body>
<h1>Hello, world!</h1>
<p>Request URI: /hello/sayhello</p>
<p>Protocol: HTTP/1.1</p>
<p>PathInfo: null</p>
<p>Remote Address: 127.0.0.1</p>
<p>A Random Number: <strong>0.3523682325749493</strong></p>
</body>
</html>
```

(Skip Unless...) The likely errors are "404 File Not Found" and "500 Internal Server Error". Read "How to debug" Section.

(Optional) Inspecting HTTP Request and Response Messages

When you enter a URL (e.g., `http://localhost:9999/hello/sayhello`) on a web browser, an HTTP GET *request message* is sent to the server; and the server returns a *response message* for display on the web browser. You can inspect the request and response messages via Web browser's Developer Tool.

For **Firefox/Chrome**, press F12 (called F12 debugger) to enable "Web Console" or "Developer Tool". Choose "Console" or "Network" pane. Enter URL `http://localhost:9999/hello/sayhello` (or refresh). Under "Net" or "Network". Expand the link `http://localhost:9999/hello/sayhello`. A HTTP

message consists of a header and a body. Inspect the request header and body; as well as the response header and body.

The request message *header* is as follows:

```
GET http://localhost:9999/hello/sayhello HTTP/1.1
Host: localhost:9999
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control:max-age=0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:52.0) Gecko/20100101 Firefox/52.0
```

For this request, there is no request message body.

The response message *header* is as follows:

```
HTTP/1.1 200 OK
Date: xxx, xx xxx xxxx xx:xx:xx xxx
Content-Length: 286
Content-Type: text/html;charset=ISO-8859-1
```

The response message *body* is as follows:

```
<!DOCTYPE html>
<html>
<head><title>Hello, World</title></head>
<body>
<h1>Hello, world!</h1>
<p>Request URI: /hello/sayhello</p>
<p>Protocol: HTTP/1.1</p>
<p>PathInfo: null</p>
<p>Remote Address: 0:0:0:0:0:0:0:1</p>
<p>A Random Number: <strong>0.4480280769255568</strong></p>
</body></html>
```

2.8 STEP 7: Write a Database Servlet

This section assumes that you are familiar with "Java database programming" and "MySQL database server". Otherwise, read "Java Database Program" and "How to Install MySQL and Get Started", respectively.

Step 7(a) Setup a Database on MySQL (Already done in the MySQL exercises)

Start your MySQL server. Take note of the server's port number. I shall assume that the MySQL server is running on port `3306`, whereas the Tomcat is running on port `9999`.

```
// For Windows: I shall assume that MySQL is installed in "c:\myWebProject\mysql"
c:
cd \myWebProject\mysql\bin
mysqld --console

// For macOS
// Use graphical control at "System Preferences" -> MySQL
```

Start a MySQL client. I shall assume that there is a user called "`myuser`" with password "`xxxx`".

```
// For Windows: I shall assume that MySQL is installed in "c:\myWebProject\mysql"
c:
cd \myWebProject\mysql\bin
mysql -u myuser -p

// For macOS: I shall assume that MySQL is installed in "/usr/local/mysql"
cd /usr/local/mysql/bin
./mysql -u myuser -p
```

Run the following SQL statements to create a database called "`ebookshop`", with a table called "`books`" with 5 columns: `id`, `title`, `author`, `price`, `qty`.

```
create database if not exists ebookshop;

use ebookshop;

drop table if exists books;
create table books (
   id     int,
   title  varchar(50),
   author varchar(50),
   price  float,
   qty    int,
```

```
   primary key (id));

insert into books values (1001, 'Java for dummies', 'Tan Ah Teck', 11.11, 11);
insert into books values (1002, 'More Java for dummies', 'Tan Ah Teck', 22.22, 22);
insert into books values (1003, 'More Java for more dummies', 'Mohammad Ali', 33.33, 33);
insert into books values (1004, 'A Cup of Java', 'Kumar', 55.55, 55);
insert into books values (1005, 'A Teaspoon of Java', 'Kevin Jones', 66.66, 66);

select * from books;
```

Step 7(b) Install MySQL JDBC Driver (Already done in the previous JDBC exercises)

You need to download MySQL JDBC driver if you have not done so. Read "Installing the MySQL JDBC Driver".

Step 7(c) Copy the MySQL JDBC Driver to Tomcat's "lib" (IMPORTANT!!! DON'T MISS THIS STEP!!!)

Copy the MySQL JDBC Driver **JAR** file "`mysql-connector-java-8.0.{xx}.jar`" into Tomcat's **`lib`** directory, i.e., "`c:\myWebProject\tomcat\lib`" (for Windows) or "`~\myWebProject\tomcat\lib`" (macOS).

Step 7(d) Write a Client-side HTML Form

Let's write an HTML script to create a *query form* with 3 checkboxes and a submit button, as illustrated below. Save the HTML file as "`querybook.html`" in your application root directory "`<TOMCAT_HOME>\webapps\hello`".


```
<!DOCTYPE html>
<html>
<head>
  <title>Yet Another Bookshop</title>
</head>
<body>
  <h2>Yet Another Bookshop</h2>
  <form method="get" action="http://localhost:9999/hello/query">
    <b>Choose an author:</b>
    <input type="checkbox" name="author" value="Tan Ah Teck">Ah Teck
    <input type="checkbox" name="author" value="Mohammad Ali">Ali
    <input type="checkbox" name="author" value="Kumar">Kumar
    <input type="submit" value="Search">
  </form>
</body>
</html>
```

You can browse the HTML page by issuing the following URL:

```
http://localhost:9999/hello/querybook.html
```

Check a box (e.g., "Tan Ah Teck") and click the "Search" button. You are expected to get an error "404 File Not Found", as you have yet to write the server-side program.

But observe the URL in the browser's navigation bar, reproduced as follows:

```
http://localhost:9999/hello/query?author=Tan+Ah+Teck
```

Query String and NAME=VALUE pairs

The above URL consists of two parts:

1. the URL extracted from the "`action`" attribute of the `<form>` tag.
2. a so-called *query string* begins with a '`?`', followed by "`name=value`" pair extracted from the selected `<input>` tag (i.e., `author=Tan+Ah+Teck`). Take note that blanks are replaced by '`+`' (or `%20` - a '`%`' followed by ASCII code of blank in hex), because blanks are not allowed in the URL.

If you check two boxes (e.g., "Tan Ah Teck" and "Mohammad Ali"), you will trigger a query string with two "name=value" pairs separated by an '`&`'.

```
http://localhost:9999/hello/query?author=Tan+Ah+Teck&author=Mohammad+Ali
```

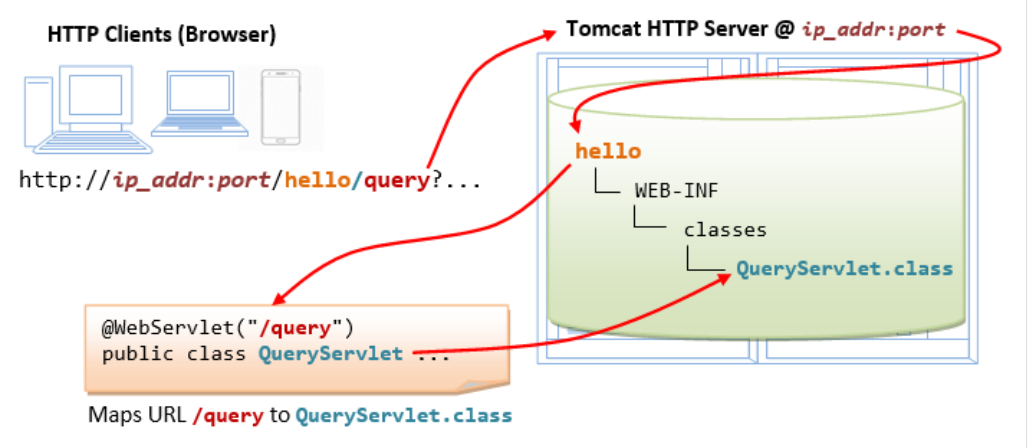
Step 7(e) Write the Server-side Database Query Servlet

The next step is to write a Java servlet, which responses to the client's request by querying the database and returns the query results.

```
// To save as "<TOMCAT_HOME>\webapps\hello\WEB-INF\classes\QueryServlet.java".
```

```
2  import java.io.*;
3  import java.sql.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import javax.servlet.annotation.*;
7
8  @WebServlet("/query")   // Configure the request URL for this servlet (Tomcat 7/Servlet 3.0 upwards)
9  public class QueryServlet extends HttpServlet {
10
11    // The doGet() runs once per HTTP GET request to this servlet.
12    @Override
13    public void doGet(HttpServletRequest request, HttpServletResponse response)
14                throws ServletException, IOException {
15       // Set the MIME type for the response message
16       response.setContentType("text/html");
17       // Get a output writer to write the response message into the network socket
18       PrintWriter out = response.getWriter();
19
20       // Print an HTML page as the output of the query
21       out.println("<!DOCTYPE html>");
22       out.println("<html>");
23       out.println("<head><title>Query Response</title></head>");
24       out.println("<body>");
25
26       try (
27          // Step 1: Allocate a database 'Connection' object
28          Connection conn = DriverManager.getConnection(
29                "jdbc:mysql://localhost:3306/ebookshop?allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC",
30                "myuser", "xxxx");   // For MySQL
31                // The format is: "jdbc:mysql://hostname:port/databaseName", "username", "password"
32
33          // Step 2: Allocate a 'Statement' object in the Connection
34          Statement stmt = conn.createStatement();
35       ) {
36          // Step 3: Execute a SQL SELECT query
37          String sqlStr = "select * from books where author = "
38                + "'" + request.getParameter("author") + "'"   // Single-quote SQL string
39                + " and qty > 0 order by price desc";
40
41          out.println("<h3>Thank you for your query.</h3>");
42          out.println("<p>Your SQL statement is: " + sqlStr + "</p>"); // Echo for debugging
43          ResultSet rset = stmt.executeQuery(sqlStr);  // Send the query to the server
44
45          // Step 4: Process the query result set
46          int count = 0;
47          while(rset.next()) {
48             // Print a paragraph <p>...</p> for each record
49             out.println("<p>" + rset.getString("author")
50                   + ", " + rset.getString("title")
51                   + ", $" + rset.getDouble("price") + "</p>");
52             count++;
53          }
54          out.println("<p>==== " + count + " records found =====</p>");
55       } catch(Exception ex) {
56          out.println("<p>Error: " + ex.getMessage() + "</p>");
57          out.println("<p>Check Tomcat console for details.</p>");
58          ex.printStackTrace();
59       }  // Step 5: Close conn and stmt - Done automatically by try-with-resources (JDK 7)
60
61       out.println("</body></html>");
62       out.close();
63    }
64 }
```

Take note that in Line 8, we configure this `QueryServlet` to URL "`/query`" via annotation `@WebServlet("/query")`. In other words, the full URL to trigger this `QueryServlet` is `http://ip_addr:port/hello/query`, which corresponds to the "`action`" attribute of the `<form>` tag of the "`querybook.html`" written earlier.



Compile "`QueryServlet.java`" with the Servlet API library as follows:

```
// Windows
c:
cd \myWebProject\tomcat\webapps\hello\WEB-INF\classes
javac -cp .;c:\myWebProject\tomcat\lib\servlet-api.jar QueryServlet.java

// macOS
cd ~/myWebProject/tomcat/webapps/hello/WEB-INF/classes
javac -cp .:$HOME/myWebProject/tomcat/lib/servlet-api.jar QueryServlet.java
```

Use a "File Explorer", verify that "`QueryServlet.class`" was generated in the "`classes`" directory.

Step 7(f) Invoke the Servlet from the Client-Side Form with Query String

Issue the following URL to browse the HMTL form "`querybook.html`" that you have created earlier:

```
http://localhost:9999/hello/querybook.html
```

Select an author (e.g., "Tan Ah Teck") and click the submit button, which activates the following URL coded in the `<form>`'s "`action`" attribute, together with a query string of NAME=VALUE pair:

```
http://localhost:9999/hello/query?author=Tan+Ah+Teck
```

The above URL "`/query`" maps to `QueryServlet`.

Using method `request.getParameter(NAME)` to handle NAME=VALUE pair

We can use the method `request.getParameter()` to handle the `NAME=VALUE` pair in the query string. Specifically, the method `request.getParameter(NAME)` returns the `VALUE`.

The above query string has a `NAME=VALUE` pair of `author=Tan+Ah+Teck`. In Line 38, the method `request.getParameter("author")` returns a string "`Tan Ah Teck`", which is inserted into the SQL `SELECT` command:

```
select * from books where author = 'Tan Ah Teck' and qty > 0 order by price desc
```

In this way, the SQL `SELECT` command is formed based on the user's selection in the form.

(Skip Unless...) If you see a blank screen or incorrect output, look for error messages from the Tomcat console!!! Check "How to debug" Database Servlet Errors.

2.9 (Obsolete and Don't Do)(Prior to Tomcat 7) Deploying Servlets using `web.xml`

Please skip this section. I keep it here just in case...

The annotation `@WebServlet("url")` for deploying servlet is supported from Tomcat 7/Servlet 3.0. Prior to Tomcat 7, you need to deploy servlets via *deployment descriptors* in the `web.xml` configuration file.

Create the following configuration file called "**`web.xml`**", and save it under "**`webapps\hello\WEB-INF`**" (i.e., "`<TOMCAT_HOME>\webapps\hello\WEB-INF\web.xml`").

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <web-app version="3.0"
3    xmlns="http://java.sun.com/xml/ns/javaee"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
6
7     <!-- To save as "hello\WEB-INF\web.xml" -->
```

```
8
9     <servlet>
10       <servlet-name>HelloWorld</servlet-name>
11       <servlet-class>HelloServlet</servlet-class>
12    </servlet>
13
14    <!-- Note: All <servlet> elements MUST be grouped together and
15      placed IN FRONT of the <servlet-mapping> elements -->
16
17    <servlet-mapping>
18       <servlet-name>HelloWorld</servlet-name>
19       <url-pattern>/sayhello</url-pattern>
20    </servlet-mapping>
21 </web-app>
```

In the above configuration, a servlet having a class file "`HelloServlet.class`" is mapped to request URL "`/sayhello`" (via an *arbitrary* `servlet-name` "`HelloWorld`"), under this web application "`hello`". In other words, the complete request URL for this servlet is "`http://hostname:port/hello/sayhello`".

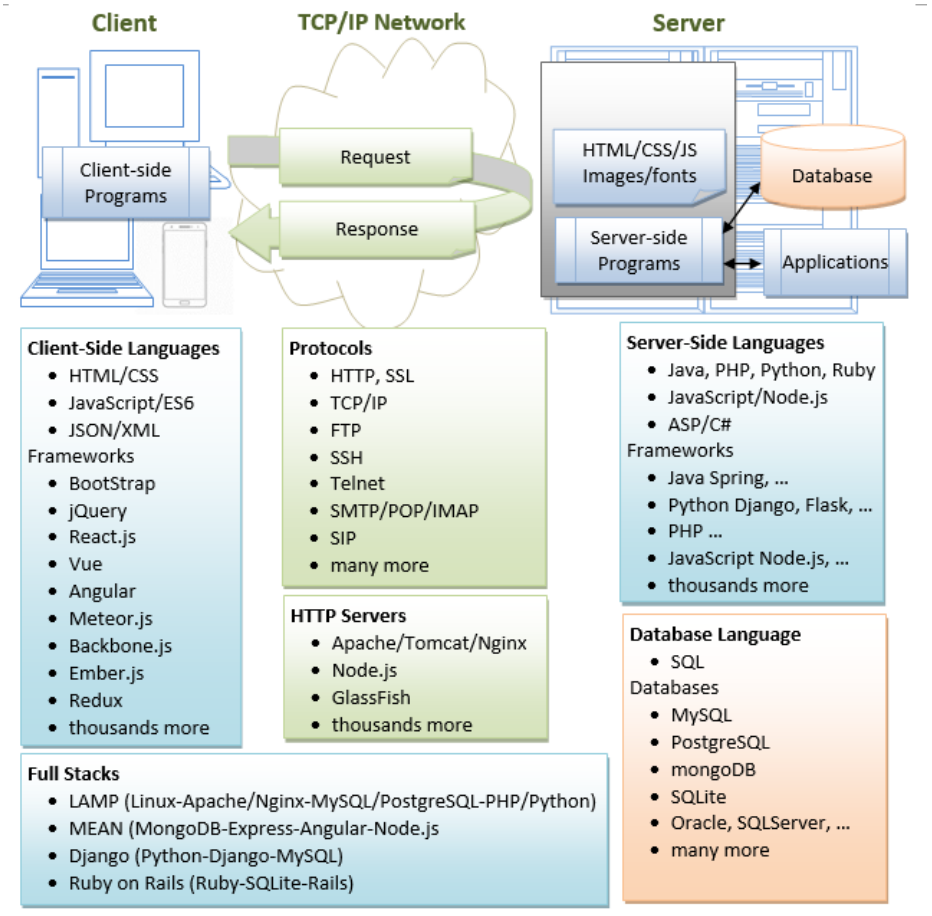
This configuration file, saved under your webapp "`hello`", is applicable only to this particular webapp "`hello`".

Restart your Tomcat server to refresh the "`web.xml`" file.

Note: For EACH servlet, you need to write a pair of `<servlet>` and `<servlet-mapping>` elements with a common but arbitrary `<servlet-name>`. Take note that all the `<servlet>` elements MUST be grouped together and placed IN FRONT of the `<servlet-mapping>` elements.

3. A Full-Stack Web Developer

A full-stack web developer is a person who can develop both client and server software. He/She needs to be familiar the client-side (front-end) programming, server-side (back-end) programming and the database. You are now a step closer.



4. (Skip Unless...) How to Debug?

"Everything that can possibly go wrong will go wrong." The most important thing to do is to find the **ERROR MESSAGES**!!!

Always...

1. **Re-start Tomcat (and Check the Tomcat's console for Error Messages)!!!**
2. **Re-start your browser!!!**
3. Refresh your browser using **Cntl-F5** (instead of refresh button or simply F5) to get a fresh copy, instead of from the cache.
4. Check your spelling! Always assume that all programs are case-sensitive. Don't type, copy and paste if possible!
5. and MOST IMPORTANTLY - Find the **ERROR MESSAGE**!!!
   a. Check the Error Messages on Tomcat's Console. Most of the error messages have a few screens of lines. You need to scroll up slowly from the last line to **look for the FIRST LINE of the error messages**.

```
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure  <== First line
The last packet sent successfully to the server was 0 milliseconds ago.
The driver has not received any packets from the server.
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:525)
        ......
        ......
        at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:305)
        at java.sql.DriverManager.getConnection(DriverManager.java:579)
        at java.sql.DriverManager.getConnection(DriverManager.java:221)
        at MySQLJdbcTestJDK7.main(MySQLJdbcTestJDK7.java:7)         <== Your program's line number here (line 7)
```

   b. Check the Tomcat's log files, located at "`<TOMCAT_HOME>\logs`". The "`catalina.yyyy-mm-dd.log`" shows the Tomcat's startup messages. Also check the "`localhost.yyyy-mm-dd.log`".
6. If things were running fine until the lightning strikes, ask yourself "What have I changed?"

4.1 Cannot Start Tomcat after Installation

```
SYMPTOM: Cannot start Tomcat after installation. The Tomcat console flashed and disappeared.
POSSIBLE SOLUTIONS:
  1. Run the script "configtest.bat" (for Windows) or "./configtest.sh" (for macOS/Linux) to check
     configuration files ("server.xml", "web.xml", "content.xml").
  2. Check the Tomcat's log files, located at "<TOMCAT_HOME>\logs".
     The "catalina.{yyyy-mm-dd}.log" shows the Tomcat's startup messages.
  3. Start the tomcat in the debugging mode by running "catalina debug" (or ./catalina.sh debug) and
     type "run" in the "jdb" prompt. Look for the error messages.
  4. If the error messages indicate that another Tomcat instance is running
     (java.net.BindException: Address already in use: JVM_Bind), kill the Tomcat process (See below).
  5. If the error messages indicate that another application is running on the Tomcat's port numbers,
     then you need to change the Tomcat's port number in "server.xml".
     You can issue command "netstat -an" to check the status of all the ports.
```

```
SYMPTOM: Cannot start Tomcat
ERROR MESSAGE:
  SEVERE: StandardServer.await: create[localhost:8005]
  java.net.BindException: Address already in use: JVM_Bind
POSSIBLE SOLUTIONS:
  1. Another Tomcat instance has been started. Kill it (see below).
  2. Another application is running on the Tomcat's port number.
     Change the Tomcat's port number in "server.xml".
     You can issue command "netstat -an" to check the status of all the ports.
```

```
SYMPTOM: Cannot start Tomcat after installation
ERROR MESSAGE:
  1. Neither the JAVA_HOME nor the JRE_HOME environment variable is defined
  2. JRE_HOME environment variable is not defined
POSSIBLE SOLUTIONS:
  1. Check if JAVA_HOME is properly defined, via command "set JAVA_HOME" (for Windows)
     or "echo $JAVA_HOME" (for macOS/Linux). Check the spelling carefully.
  2. Define environment variable JAVA_HOME according to "Step 2: Create an Environment Variable JAVA_HOME".
```

```
SYMPTOM: Cannot start Tomcat after installation (for macOS)
ERROR MESSAGE:
  "Permission Denied" running catalina.sh
POSSIBLE SOLUTIONS:
  Check if catalina.sh is executable by the current user!
```

Locating/Killing Tomcat's Process

- In windows, start "Task Manager", Tomcat run as a "process" named "`java.exe`". You may need to kill the process.
- In macOS, start "Activity Monitor". Select "All Processes" and look for "`java.exe`".
- In Linux/macOS, you may issue "`ps aux | grep tomcat`" to locate the Tomcat process. Note down the process ID (pid). You can kill the Tomcat process via "`kill -9 pid`".

4.2 Cannot Access the Tomcat Server From Browser

```
ERROR MESSAGE:
  (Firefox) Unable to Connect
  (IE) Internet Explorer cannot display the webpage
  (Chrome) Oops! Google Chrome could not connect to ...
  (Safari) Safari can't connect to the server
PROBABLE CAUSES: You are simply not connecting to your Tomcat.
POSSIBLE SOLUTION:
  1. Check if your Tomcat server has been started?
  2. Check the hostname and port number of your URL (http://localhost:9999/...)

ERROR MESSAGE: Error 404 File Not Found
PROBABLE CAUSES: You have connected to your Tomcat.
  But Tomcat server cannot find the HTML file or Servlet that your requested.
POSSIBLE SOLUTION:
  1. Check your spelling! The path is case-sensitive!
  2. For HTML file request with URL http://localhost:9999/xxxx/filename.html:
     a. Open Tomcat's "webapps" directory, check if sub-directory "xxxx" exists. It is case-sensitive.
     b. Open the "xxxx" directory, check if "filename.html" exists.
  3. For servlet request with URL http://localhost:9999/xxxx/servletURL:
     a. Check the Tomcat's console for error message.
     b. Check the Tomcat console to make sure that your application "xxxx" has been deployed.
     c. Open Tomcat's "webapps" directory, check if sub-directory "xxxx" exists.
     d. Open the "xxxx" directory, check if sub-directory "WEB-INF" (uppercase with a dash) exists.
     e. Open the "WEB-INF", check if sub-directory "classes" (lowercase, plural) exists.
     f. Open your servlet, check if the servlet is mapped to servletURL
     g. Check if you have compiled the servlet. That is, the .class exisits (NOT .java).

ERROR MESSAGE: Error 500 Internal Server Error
POSSIBLE SOLUTION:
  Error 500 should have triggered many error message in the Tomcat's console.
  Go to the Tomcat's console, find the error message.

ERROR MESSAGE: Error 505: GET (or POST) method not supported
POSSIBLE SOLUTION:
  Check you servlet to make sure that you have defined a doGet() (or doPost()) method.
```

4.3 Java Servlet Errors

```
SYMPTOM: Cannot compile Java Servlet
ERROR MESSAGE: class xxxx is public, should be declared in a file named xxxx.java
CAUSES/SOLUTION:
  In Java, the filename must be the same as the classname with extension of ".java".
```

```
SYMPTOM: Cannot compile Java Servlet
ERROR MESSAGE: package javax.servlet does not exist
CAUSES/SOLUTION:
  The Java Servlet library is missing. Read "Step 6(b) Compiling the Servlet"
  again, again and again....
```

```
SYMPTOM: A "new" servlet does not run
ERROR MESSAGE:
java.lang.IllegalStateException: Error starting child (in Tomcat console)
The servlets named [xxx] and [yyy] are both mapped to the url-pattern [zzz] which is not permitted
CAUSES/SOLUTION:
  Check to ensure that no two servlets are mapping to the SAME URL in @WebServlet("url")
```

4.4 Java Database Servlet Errors

```
ERROR MESSAGE: No suitable driver found
POSSIBLE SOLUTION:
  Check if you have done Step 7(c) Copy the MySQL JDBC Drive to Tomcat's "lib".

ERROR MESSAGE: Communications link failure
POSSIBLE SOLUTION:
```

```
  Check if you have started MySQL server.

ERROR MESSAGE: Access denied for user 'myuser'@'localhost' (using password: YES)
POSSIBLE SOLUTION:
  Wrong username or password in your servlet's getConnection()

ERROR MESSAGE: Public Key Retrieval is not allowed
POSSIBLE SOLUTION:
  Add jdbc:mysql://localhost:3306/ebookshop?allowPublicKeyRetrieval=true&useSSL=false to databaseURL.

ERROR MESSAGE: The server time zone value '...' is unrecognized or represents more than one time zone
POSSIBLE SOLUTION:
  Add jdbc:mysql://localhost:3306/ebookshop?serverTimezone=UTC to databaseURL.

ERROR MESSAGE: You have an error in your SQL syntax ...
POSSIBLE SOLUTION:
  SQL syntax error. Check you SQL statement.
```

REFERENCES & RESOURCES

1. Apache Tomcat mother site @ http://tomcat.apache.org.
2. Apache Tomcat Documentation @ "<TOMCAT_HOME>\webapps\docs".
3. "How to install MySQL and Get Started".
4. "Introduction to Java Database (JDBC) Programming".
5. Jason Brittain, Ian F. Darwin, "*Tomcat The Definitive Guide*", 2nd eds, OReilly, 2007.

Latest version tested: Tomcat 9.0.41, MySQL 8.0.23, JDK 15.0.2, Windows 10, macOS 10.15, Ubuntu 18.04LTS
Last modified: March, 2021

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | HOME