# [What's the purpose of the Apache authentication directive AuthName?](http://stackoverflow.com/questions/23446671/whats-the-purpose-of-the-apache-authentication-directive-authname)

# The realm name will make the user blink that he need to give particular user/passwd, because 'secret area' realm belong to protected 3 directories(say) docs, test1, test2 in webapps folder where as realm name 'Private' belong to 2 directories(say) dummy1, dummy2 in webapps folder.

<Directory protected-directories>

AuthType Basic

AuthName Private

AuthUserFile conf/basic.users

Require valid-user

</Directory>

----------------------------------------------------------------------------------------------------------------------------------

**Q1) What will be the Request Message body for the URL request submitted? (content)**

**Q2) What will be the Response Message body from the tomcat server? (content)**

A1) There will be no body. Only POST and PUT requests have a body.

A2) The response body contains 394 bytes, which are the bytes of the gif image requested.

-----------------------------------------------------------------------------------------------------------------------------------

[**How do multiple clients connect simultaneously to one port, say 80, on a server?**](http://stackoverflow.com/questions/3329641/how-do-multiple-clients-connect-simultaneously-to-one-port-say-80-on-a-server)

**First remember below two rules:**

1. Primary key of a socket: A socket is identified by {SRC-IP, SRC-PORT, DEST-IP, DEST-PORT, PROTOCOL} not by {SRC-IP, SRC-PORT, DEST-IP, DEST-PORT} - Protocol is an important part of a socket's definition.
2. OS Process & Socket mapping: A process can be associated with (can open/can listen to) multiple sockets which might be obvious to many readers but also remember that two processes CANNOT share same socket.

**Example 1:** Two clients connecting to same server port means: socket1 {SRC-A, 100, DEST-X,80, TCP} and socket2{SRC-B, 100, DEST-X,80, TCP}. This means host A connects to server X's port 80 and another host B also connects to same server X to the same port 80. Now how server handles these two sockets decides if a server is single threaded or multiple threaded (I'll explain this later), what is important that one server can listen to multiple sockets simultaneously.

**To answer the original question of the post:**

Irrespective of state-full or stateless protocols, two clients can connect to same server port because for each client we can assign a different socket (as client IP will definitely differ). Same client can also have two sockets connecting to same server port - since such sockets differ by SRC-PORT. With all fairness, "Borealid" essentially mentioned the same correct answer but the reference to state-less/full was kind of unnecessary/confusing.

To answer the second part of the question on how a server knows which socket to answer. First understand that for a single server process that is listening to same port, there could be more than one sockets (may be from same client or from different clients). Now as long as a server knows which request is associated with which socket, it can always respond to appropriate client using the same socket. Thus a server never needs to open another port in its own node than the original one on which client initially tried to bind. If any server allocates different server-ports after a socket is bound, then in my opinion the server is wasting its resource and it must be needing the client to bind again to the new port assigned.

**A bit more for completeness:**

**Example 2:** It's very interesting question that can a server's two different processes listen to same port. If you do not consider protocol as one of parameter defining socket then the answer is no. Initiatively this is so because we can say that in such case for a single client trying to connect to a server-port will not have any mechanism to mention which of the two listening process the client intends to. This is the same theme asserted by rule (2). However this is WRONG answer because 'protocol' is also a part of the socket definition. Thus two processes in same node can listen to same port only if they are using different protocol. For example two unrelated clients (say one is using TCP and another is using UDP) can bind and communicate to same server node and to the same port but they must be served by two different server-processes.

**Server Types - single & multiple:**

When a server's processes listening to a port that means multiple sockets can simultaneously connect and communicate with the same server-process. If a server uses only a single child-process to serve all the sockets then the server is called single-process/threaded and if the server uses many sub-processes to serve each socket by one sub-process then the server is called multi-process/threaded server. Note that irrespective of the server's type a server can/should always uses the same initial socket to respond back (no need to allocate another server-port).

Suggested [Books](http://rads.stackoverflow.com/amzn/click/0130183806) and rest of the two volumes if you can.

\*\* A Note on Parent/Child Process (in response to query/comment of 'Ioan Alexandru Cucu') \*\*

Wherever I mentioned any concept in relation to two processes say A and B, consider that they are not related by parent child relationship. OS's (especially UNIX) by design allow a child process to inherit all File-descriptors (FD) from parents. Thus all the sockets (in UNIX like OS are also part of FD) that a process A listening to, can be listened by many more processes A1, A2, .. as long as they are related by parent-child relation to A. But an independent process B (i.e. having no parent-child relation to A) cannot listen to same socket. In addition, also note that this rule of disallowing two independent processes to listen to same socket lies on an OS (or its network libraries) and by far it's obeyed by most OS's. However, one can create own OS which can very well violate this restrictions.

# [What are the difference between server-side and client-side programming?](http://programmers.stackexchange.com/questions/171203/what-are-the-difference-between-server-side-and-client-side-programming)

Web development is all about communication. In this case, communication between 2 parties, over the HTTP protocol:

* The **Server** - This party is responsible for **serving** pages.
* The **Client** - This party requests pages from the **Server**, and displays them to the user. On most cases, the client is a **web browser**.
  + The **User** - The user uses the **Client** in order to surf the web, fill in forms, watch videos online, etc.

Each side's programming, refers to code which runs at the specific machine, the server's or the client's.

# Basic Example

1. The **User** opens his web browser (the **Client**).
2. The **User** browses to [http://google.com](http://google.com/).
3. The **Client** (on the behalf of the **User**), sends a request to [http://google.com](http://google.com/) (the **Server**), for their home page.
4. The **Server** then acknowledges the request, and replies the client with some meta-data (calledheaders), followed by the page's source.
5. The **Client** then receives the page's source, and renders it into a human viewable website.
6. The **User** types Stack Overflow into the search bar, and presses Enter
7. The **Client** submits that data to the **Server**.
8. The **Server** processes that data, and replies with a page matching the search results.
9. The **Client**, once again, renders that page for the **User** to view.

# Programming

## Server-side Programming

Server-side programming, is the general name for the kinds of programs which are run on the **Server**.

### Uses

* Process user input.
* Display pages.
* Structure web applications.
* Interact with permanent storage (SQL, files).

### Example Languages

* PHP
* ASP.Net in C#, C++, or Visual Basic.
* Nearly any language (C++, C#, Java). These were not designed specifically for the task, but are now often used for application-level web services.

## Client-side programming

Much like the server-side, Client-side programming is the name for all of the programs which are run on the**Client**.

### Uses

* Make interactive webpages.
* Make stuff happen dynamically on the web page.
* Interact with temporary storage, and local storage (Cookies, localStorage).
* Send requests to the server, and retrieve data from it.
* Provide a remote service for client-side applications, such as software registration, content delivery, or remote multi-player gaming.

### Example languages

* JavaScript (primarily)
* HTML\*
* CSS\*
* Any language running on a client device that interacts with a remote service is a client-side language.

\*HTML and CSS aren't really "programming languages" per-se. They are markup syntax by which the**Client**renders the page for the**User**.

**In layman's words:**

Here I will talk only about web programming.

**Client side** programming has mostly to do with the user interface, with which the user interacts. In web developing it's the browser, in the user's machine, that runs this code, and is mainly done in **javascript, flash,** etc. This code must run in a variety of browsers.

**Its main tasks are:**

* validating input
* animation
* manipulating UI elements
* applying styles
* some calculations are done when you don't want the page to refresh so often

The **person in charge** of front end programming **must know**:

* javascript
* css
* HTML
* basic graphic design
* Ajax
* maybe Flash
* some 3rd party javascript libraries like JQuery
* UI design
* information design, etc.

**Server side** programming has to do with generating dynamic content. It runs on servers. Many of these servers are "headless". Most web pages are not static, they search a database in order to show the user updated personalized information. This sides interacts with the back end, like say, the database.

This programming can be done in a lot of languages:

* PHP
* Java and jsp
* asp
* Perl
* Python
* Ruby on Rails, etc.

This code has to do which:

* Querying the database
* Encode the data into html
* Insert and update information onto the database
* Business rules and calculations

The person in charge of server side programing must know:

* some of the languages mentioned above
* HTML
* SQL,
* linux/unix shell scripting
* OOP
* business rules, etc.

# Can't i just program a sample server like the one described in that simple tutorial, and leave it always running on a computer?

Using a socket is very simple. At first.

You create your server class, you have it bind to a port and away you go.

The first hurdle you will hit, covered in the comments, is multi-threading. But a simple producer/consumer pattern will solve that for you in no time.

The next problem you will hit is **protocol**.

* Who talks first?
* How do you respond?
* How do you deal with an invalid request?
* What happens if the stream collapses during a request?
* Do you open a new socket for each request or does a client hold onto a socket and write multiple requests?
* Maybe you want some sort of non-blocking IO?

This is where HTTP comes in, it is a protocol for communicating over TCP/IP (actually over anything, you could use bits of paper and a bike). It defines answers to all the above questions (and many more).

So, you run a webserver (tomcat, glassfish) and it deals with the raw sockets and sending the right information.

A servlet is an abstraction, when Tomcat has a connection and it has negotiated compression, encryption etc it will pass a request onto the servlet.

The servlet doesn't have to worry about the raw socket, it reads the request and writes a response.

It's worth pointing out that HTTP isn't the only protocol, it's just the one happens to be used for web-browsing. And so the one used by web-servers.

# -------------------------------------------------------------------------------

# Points from Brian Will (code school)

Apache HTTP web server has 60% of market share.

IIS(Inter Information Services) web server which runs only on windows has 15% market share.

Nginx is another open source webserver which has 8-10% of market share.

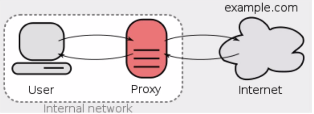
Lighttpd is another open source webserver which has 5% of market share.

Nginx & Lighttpd became popular because they serve many concurrent requests quickly. With this quality, both of these servers are used as **forward proxy** or **reverse proxy**.

# What is forward proxy & Reverse proxy?

In normal case, Webserver receive request from web browser and returns an http Response to that user. When acting as a proxy however, the webserver will take the request and actually pass it on to something else. It doesn’t really process on the request itself. When proxy gets its response for the request forwarded, proxy returns the http Response to the original webbrowser.

Forward proxy is immediately positioned in the network of requesting user agent itself.



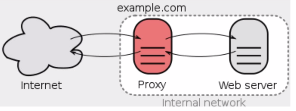
Say, you work at a company and company has it’s own local network that is connected to the Internet, but they configure their network such each host request uses forward proxy and then go on to the web. If you look into you OS network configuration, you see that option for specifying a proxy. That’s where you configure the system to use a forward proxy.

But what is the point in using the forward proxy?

There are many different reasons for an organization like Business, you don’t want users to go to certain websites, proxy can act as a filter on what websites users can visit. Forward proxy can keep the log of record about what the each user do on the web for security reasons.

So, Mainly the purpose is about Security & Controlling access.

Reverse proxy is immediately positioned in the network where webserver is hosted.



This is between the Webserver and requests coming from Internet.

Reverse proxy might improve performance by load balancing where the request can be diverted to another webserver, if one webserver is highly loaded.

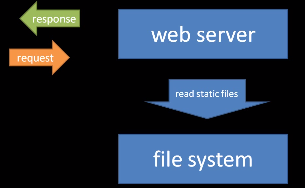
Another way, Reverse proxy might improve performance, if it acts as a cache. So, you get frequent repetitive requests coming from Internet, where proxy can intelligently return cached responses for such requests, instead of forwarding the request to webserver. So, Webserver does not touch such requests at all.

# Evolution of Webserver

In the simplest possible setup of a webserver, Webserver is simply configured to read static files from the filesystem and then serve those files back as responses.

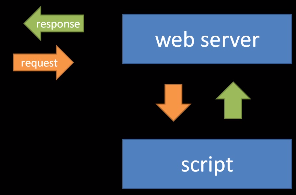
In this, Webserver is pointed to some particular directory and for each request, Webserver interprets the path of the URL as relative path of the file in the directory.

If the path simply reads index.html webserver will return the content of index.html as response from the directory to which webserver is configured.



But Now a days, Almost all webserver are configured to take the incoming request and pass it along to another program.

One flavor of it is to pass it along to a script and that script responses back to webserver and the webserver actually returns the http Response.



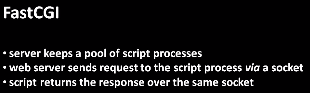
To facilitate this pattern, a standard was developed specifying how a script and webserver should communicate. This standard is called CGI.

**CGI(Common Gateway Interface)**

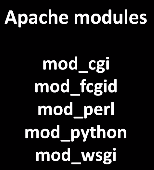
* Each request runs a script as a new, separate process
* Script receives headers and GET variables in environment (by communication of data areas of both parent/child process)
* script reads POST body from standard input(using pipes).
* script writes response body to standard output(using pipes).

But this approach was inherently inefficient due to overhead in launching whole new process for every single request.

To overcome these problems, couple of alternatives was introduced.



But Most commonly used approach on these lines, is taking the interpreter for most commonly used scripting languages(like python) and embed in the web server itself. In the case of Apache,



these interpreter’s come in the apache modules. These webserver’s are written in a modular style.

Tomcat = ( Web Server + Servlet container + JSP environment )

----------------------------------------------------------------------------------------------------------------------------------

# [Java Servlets threading model](http://stackoverflow.com/questions/2095418/java-servlets-threading-model)

If requests were handled serially by servlets, then web applications would be very slow. It's actually the case that servlets need to be thread-safe, because a single instance of a servlet can be responsible for handling multiple requests simultaneously.

Usually a Servlet container will maintain a thread pool for handling requests, with incoming requests being assigned to threads on an on-demand basis.

The servlet container is an extension of a Web server in the same way CGI, ASP, and PHP are.

The Servlet container is often written in Java, since it eventually runs inside a JVM. However, some vendors implement their containers in different languages (they aren’t essentially bound to Java). The point here is the fact that, all we need is a servlet container that can read and execute our servlets. The language in which it is implemented is not necessarily important for us.

**Flow of Control**

Below is a simple illustration of the flow of control that starts and ends at the Web Browser. Control starts at the browser, goes to the server and then to the container which in turn invokes the Servlet. Our Servlet happily does the processing (which might involve hitting a database) and returns the data to the container which in turn passes it on to the web server which finally displays the contents on the web browser.

# 

# --------------------------------------------------------------------------------

**Q) How Servlet Matching Procedure happens?**

A request may match more than one servlet-mapping in a given context. The servlet container uses a straightforward matching procedure to determine the best match.

**The matching procedure has four simple rules.**

* First, the container prefers an **exact path match** over a wildcard path match.
* Second, the container prefers to match the **longest pattern**.
* Third, the container prefers **path matches over filetype** matches.
* Finally, the pattern <url-pattern>/</url-pattern> always matches any request that no other pattern matches.

For example, a context web.xml file can map the home page for an online catalog to one pattern and the search page for the catalog to a different pattern, as shown below:

<servlet-mapping>

<servlet-name>catalogBrowse</servlet-name>

<url-pattern>/Catalog/\*</url-pattern>

</servlet-mapping>

<servlet-mapping>

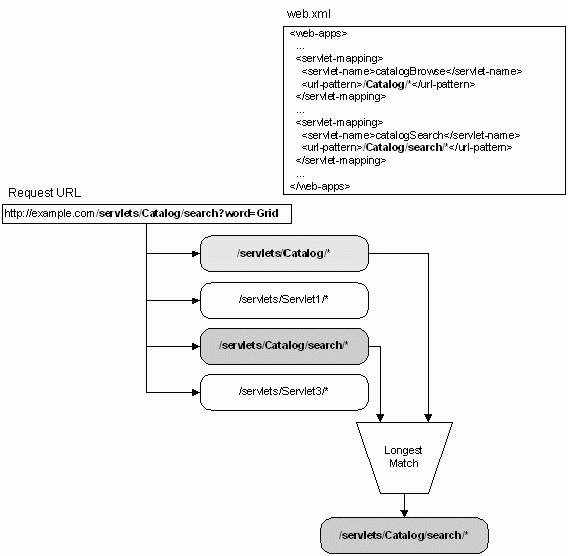
<servlet-name>catalogSearch</servlet-name>

<url-pattern>/Catalog/search/\*</url-pattern>

</servlet-mapping>

Below figure illustrates the matching process for a context. Since the container prefers to match the **longest pattern**, a URL that includes /Catalog/search/ always matches the mapping for catalogSearch rather than the mapping for catalogBrowse.

**URL pattern matching**



Jsp :

jsp is interpreted on the tomcat server side  and not on browser

Browser’s only understands (html + javascript + css)

tomcat = (webserver-http + servlet\_container + jasper)

jasper is a template engine that compiles the jsp into a java class at runtime and then that class is run on the jvm.

jasper converts jsp -> .class

.class runs on the jvm

So the http response will have (html + javascript + css)

so you have two options here , implement a servlet class that returns the response (html + javascript)

write a jsp , which automagically gets compiled to the servelet class by jasper and returns html+ javascript

ServletContext:

Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

There is one context per "web application" per Java Virtual Machine. (A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace such as /catalog and possibly installed via a.war file.)

In the case of a web application marked "distributed" in its deployment descriptor, there will be one context instance for each virtual machine. In this situation, the context cannot be used as a location to share global information (because the information won't be truly global). Use an external resource like a database instead.

The ServletContext object is contained within the [ServletConfig](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletConfig.html" \o "interface in javax.servlet) object, which the Web server provides the servlet when the servlet is initialized.

Q) What does this code do?

this.getServletContext().getRequestDispatcher("/index.jsp").forward(request, response);

A) With the call to [getRequestDispatcher](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html" \l "getRequestDispatcher%28java.lang.String%29) the container

Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.

The resource located at the given path, in this case, is a JSP (another Servlet). With your call to forward, you are, down the line, invoking service on the JSP servlet. That renders the content of your JSP and sends it as the response body.

## ServletContext

When the servletcontainer (like [Apache Tomcat](http://tomcat.apache.org/)) starts up, it will deploy and load all webapplications. When a webapplication get loaded, the servletcontainer will create the [ServletContext](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html) once and keep in server's memory. The webapp's web.xml will be parsed and every Servlet, Filter and Listener found in web.xml or annotated with respectively @WebServlet, @WebFilter and @WebListener will be created once and kept in server's memory as well. When the servletcontainer shuts down, it will unload all webapplications and the ServletContext and all Servlet, Filter and Listener instances will be trashed.

## HttpServletRequest and HttpServletResponse

The servletcontainer is attached to a webserver which listens on HTTP requests on a certain port number, which is usually 8080 in development and 80 in production. When a client (user with a webbrowser) sends a HTTP request, the servletcontainer will create new [HttpServletRequest](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html) and [HttpServletResponse](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html) objects and pass it through the methods of the already-created Filter and Servlet instances whose url-pattern matches the request URL, all in the same thread.

The request object provides access to all information of the HTTP request, such as the request headers and the request body. The response object provides facility to control and send the HTTP response the way you want, such as setting headers and the body (usually with HTML content from a JSP file). When the HTTP response is committed and finished, then both the request and response objects will be trashed.

## HttpSession

When a client visits the webapp for the first time and/or the [HttpSession](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html) is to be obtained for the first time by request.getSession(), then the servletcontainer will create it, generate a long and unique ID (which you can get by session.getId()) and store it in server's memory. The servletcontainer will also set a [Cookie](http://docs.oracle.com/javaee/7/api/javax/servlet/http/Cookie.html) in the Set-Cookie header of the HTTP response with JSESSIONID as cookie name and the unique session ID as cookie value.

As per the [HTTP cookie specification](http://www.faqs.org/rfcs/rfc2965.html) (a contract a decent webbrowser and webserver has to adhere), the client (the webbrowser) is required to send this cookie back in the subsequent requests in the Cookie header as long as the cookie is valid. Using browser builtin HTTP traffic monitor you can check them (press F12 in Chrome / Firefox23+ / IE9+ and check Net/Network tab). The servletcontainer will determine the Cookie header of every incoming HTTP request for the presence of the cookie with the name JSESSIONID and use its value (the session ID) to get the associated HttpSession from server's memory.

The HttpSession lives until it has not been used for more than the <session-timeout> time, a setting you can specify in web.xml, which defaults to 30 minutes. So when the client doesn't visit the webapp anymore for over 30 minutes, then the servletcontainer will trash the session. Every subsequent request, even though with the cookie specified, will not have access to the same session anymore. The servletcontainer will create a new one.

On the other hand, the session cookie on the client side has a default lifetime which is as long as the browser instance is running. So when the client closes the browser instance (all tabs/windows), then the session will be trashed at the client side. In a new browser instance the cookie associated with the session won't be sent anymore. A new request.getSession() would return a brand new HttpSession and set a cookie with a brand new session ID.

## In a nutshell

* The ServletContext lives as long as the webapp lives. It's been shared among all requests in allsessions.
* The HttpSession lives as long as the client is interacting with the webapp with the same browser instance and the session hasn't timed out at the server side yet. It's been shared amongall requests in the same session.
* The HttpServletRequest and HttpServletResponse lives as long as the client has sent it until the complete response (the webpage) is arrived. It is not being shared elsewhere.
* Any Servlet, Filter and Listener lives as long as the webapp lives. They are being shared among all requests in all sessions.
* Any attribute which you set in ServletContext, HttpServletRequest and HttpSession will live as long as the object in question lives.

## Threadsafety

That said, your major concern is possibly threadsafety. You should now have learnt that Servlets and filters are shared among all requests. That's the nice thing of Java, it's multithreaded and different threads (read: HTTP requests) can make use of the same instance. It would otherwise have been too expensive to recreate it on every request.

But you should also realize that you should **never** assign any request or session scoped data as aninstance variable of a servlet or filter. It will be shared among all other requests in other sessions. That's threadunsafe! The below example illustrates that:

public class ExampleServlet extends HttpServlet {

private Object thisIsNOTThreadSafe;

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

Object thisIsThreadSafe;

thisIsNOTThreadSafe = request.getParameter("foo"); // BAD!! Shared among all requests!

thisIsThreadSafe = request.getParameter("foo"); // OK, this is thread safe.

}

}

The session management (client identification, cookie handling, saving session scoped data and so on) is basically already done by the appserver itself. You don't need to worry about it at all. You can just set/get Java objects in the session by [HttpSession#setAttribute()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html" \l "setAttribute%28java.lang.String,%20java.lang.Object%29) and [#getAttribute()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html#getAttribute%28java.lang.String%29). Only thing what you really need to take care with is the **URL rewriting** for the case that the client doesn't support cookies. It will then append a jsessionid identifier to the URL. In the JSP you can use the JSTL's [c:url](http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/c/url.html) for this. In the Servlet you can use [HttpServletResponse#encodeURL()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html" \l "encodeURL%28java.lang.String%29) for this. This way the server can identify the client by reading the new request URL.

Your new question shall probably be "But how are cookies related to this? How does the server do it all?". Well, the answer is this: if the server receives a request from a client and the server side code (your code) is trying to get the [HttpSession](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html) by [HttpServletRequest#getSession()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html" \l "getSession%28%29) while there's no one created yet (first request in a fresh session), the server will create a new one itself. The server will generate an long, unique and hard-to-guess ID (the one which you can get by [HttpSession#getId()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html" \l "getId%28%29)) and set this ID as a value of the cookie with the name jsessionid. Under the hood the server uses [HttpServletResponse#addCookie()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html" \l "addCookie%28javax.servlet.http.Cookie%29) for this. Finally the server will store all sessions in some kind of Map with the session ID as key and the HttpSession as value.

According to the [HTTP cookie spec](http://tools.ietf.org/html/rfc6265) the client is required to send the same cookies back in the headers of the subsequent request. Under the hood the server will search the jsessionid cookie by [HttpServletRequest#getCookies()](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html" \l "getCookies%28%29) and determine its value. This way the server is able to obtain the associated HttpSession and give it back by every call on HttpServletRequest#getSession().

To the point: the only which is stored in the client side is the session ID (in flavor of a cookie) and the HttpSession object (including all of its attributes) is stored in the server side (in Java's memory). You don't need to worry about session management youself and you also don't need to worry about the security.

Q) Are there multiple instances of servlet class? As I hear "each instance of servlet" Can anybody elaborate on this?

A) When the Servlet container starts, it:

1. reads web.xml;
2. finds the declared Servlets in the classpath; and
3. loads and instantiates each Servlet **only once**.

Roughly, like this:

String urlPattern = parseWebXmlAndRetrieveServletUrlPattern();

String servletClass = parseWebXmlAndRetrieveServletClass();

HttpServlet servlet = (HttpServlet) Class.forName(servletClass).newInstance();

servlet.init();

servlets.put(urlPattern, servlet); // Similar to a map interface.

Those Servlets are stored in memory and reused every time the request URL matches the Servlet's associated url-pattern. The servlet container then executes code similar to:

for (Entry<String, HttpServlet> entry : servlets.entrySet()) {

String urlPattern = entry.getKey();

HttpServlet servlet = entry.getValue();

if (request.getRequestURL().matches(urlPattern)) {

servlet.service(request, response);

break;

}

}

The [GenericServlet#service()](http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServlet.html" \l "service%28javax.servlet.http.HttpServletRequest,%20javax.servlet.http.HttpServletResponse%29) on its turn decides which of the doGet(), doPost(), etc.. to invoke based on [HttpServletRequest#getMethod()](http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServletRequest.html" \l "getMethod%28%29).

You see, the servletcontainer reuses the **same servlet instance** for every request. In other words: the servlets are shared among *every request*. That's why it's extremely important to write servlet code the threadsafe manner --which is actually simple: just do **not** assign request or session scoped data as servlet instance variables, but just as method local variables. E.g.

public class MyServlet extends HttpServlet {

private Object thisIsNOTThreadSafe;

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

Object thisIsThreadSafe;

thisIsNOTThreadSafe = request.getParameter("foo"); // BAD!! Shared among all requests!

thisIsThreadSafe = request.getParameter("foo"); // OK, this is thread safe.

}

}

# Q) [What is the difference between JSF, Servlet and JSP?](http://stackoverflow.com/questions/2095397/what-is-the-difference-between-jsf-servlet-and-jsp)

JSP is a **Java view technology** running on the server machine which allows you to write template text in (the client side languages like HTML, CSS, JavaScript and so on). JSP supports [taglibs](http://docs.oracle.com/javaee/5/tutorial/doc/bnann.html), which are backed by pieces of Java code that let you control the page flow or output dynamically. A well known taglib is [JSTL](http://stackoverflow.com/tags/jstl/info). JSP also supports [Expression Language](http://stackoverflow.com/tags/el/info), which can be used to access backend data (via attributes available in page, request, session and application scopes), mostly in combination with taglibs.

When a JSP is requested for the first time or when the webapp starts up, the servlet container will compile it into a class extending [HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html) and use it during the webapp's lifetime. You can find the generated source code in the server's work directory. In for example [Tomcat](http://tomcat.apache.org/), it's the /workdirectory. On a JSP request, the servlet container will execute the compiled JSP class and send the generated output (usually just HTML/CSS/JS) through the webserver over network to the client side, which in turn displays it in the web browser.

### [Servlets](http://stackoverflow.com/tags/servlets/info)

Servlet is an **Java application programming interface (API)** running on the server machine, which intercepts requests made by the client and generates/sends a response. A well known example is theHttpServlet which provides methods to hook on [HTTP](http://www.w3.org/Protocols/rfc2616/rfc2616.html) requests using the popular [HTTP methods](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html)such as GET and POST. You can configure HttpServlets to listen on a certain HTTP URL pattern, which is configurable in web.xml, or more recently with [Java EE 6](http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html), with @WebServlet annotation.

When a Servlet is first requested or during webapp startup, the servlet container will create an instance of it and keep it in memory during the webapp's lifetime. The same instance will be reused for every incoming request whose URL matches the servlet's URL pattern. You can access the request data by [HttpServletRequest](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html) and handle the response by [HttpServletResponse](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html). Both objects are available as method arguments inside any of the overridden methods of HttpServlet, such as doGet() and doPost().

### [JSF (JavaServer Faces)](http://stackoverflow.com/tags/jsf/info)

JSF is a **component based MVC framework** which is built on top of the Servlet API, and provides[components](http://docs.oracle.com/javaee/6/tutorial/doc/bnarf.html) via taglibs which can be used in JSP or any other Java based view technology such as[Facelets](http://docs.oracle.com/javaee/6/tutorial/doc/giepx.html). Facelets is much more suited to JSF than JSP. It namely provides great [templating capabilities](http://docs.oracle.com/javaee/6/tutorial/doc/giqxp.html) such as [composite components](http://docs.oracle.com/javaee/6/tutorial/doc/giqzr.html), while JSP basically only offers the [<jsp:include>](http://java.sun.com/products/jsp/syntax/2.0/syntaxref2020.html#8828) for templating, so that you're forced to create custom components with raw Java code (which is a bit opaque and a lot of tedious work in JSF) when you want to replace a repeated group of components with a single component. Since JSF 2.0, JSP has been deprecated as view technology in favor of Facelets.

As being a MVC ([Model-View-Controller](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)) framework, JSF provides the [FacesServlet](http://docs.oracle.com/javaee/6/api/javax/faces/webapp/FacesServlet.html) as the sole request-response Controller. It takes all the standard and tedious HTTP request/response work from your hands, such as gathering user input, validating/converting them, putting them in model objects, invoking actions and rendering the response. This way you end up with basically a JSP or Facelets (XHTML) page for View and a Javabean class as Model. The JSF components are been used to bind the view with the model (such as your ASP.NET web control does) and the FacesServlet uses theJSF component tree to do all the work.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Q) Where are attributes stored?

A) There are three scopes:

1. Application Scope (I.e. SevletContext)
2. Session Scope (I.e. HttpSession)
3. Request Scope (I.e. HttpServletRequest)

Getting ServletContext Object:

1. getServletContext().set attribute("name","value"); //now the name attribute will be accessible from any Servlet within the application.

Getting HttpSession Object:

1. request.getSession(true).set attribute("name2","value"); //now the name2 attribute will be accessible from current session
2. request.set attribute("name3","value");

//now the name3 attribute will be accessible anywhere in Servlet or jsp before sending response back to the client.

Ques:where attributes are stored?

Ans: Attributes are stored in a Map(in name-value pair) of respective Scope. i.e. Session Map, Request Map and ServletContext Map.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Q) What is an artifact in maven world?

A) In Maven terminology, the artifact is the resulting output of the maven build, generally a jar or war or other executable file. Artifacts in maven are identified by a coordinate system of groupId, artifactId, and version. Maven uses the groupId, artifactId, and version to identify dependencies (usually other jar files) needed to build and run your code.

By default, Maven will download from the Maven Central Repository, which is located at this URL:<http://search.maven.org/>

Maven will download the artifacts when it needs them. So doing an mvn update or mvn install would trigger Maven to go to the repository if it doesn't already have the necessary JARs locally. And the local folder where the JAR files gets stored is C:\Users\your\_windows\_user\.m2\repository by default.

Maven operates by managing dependencies, which are individual JAR files. So if you need to use a class, Maven will pull in the entire JAR file containing that class. The main reason for this is that Java libraries typically ship as JAR files, not individual classes.

The add dependency actually gets its list of dependencies from the Central Repository. I would assume that if you select a dependency and you already have the JAR file locally, then it will not download it again.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Q) What is Spring Project Nature?

A) The "nature" has nothing to do with your code or spring. In eclipse, a "project nature" creates an association between the project and a tool, plug-in, or feature set. By adding a nature to an eclipse project, you tell an eclipse plug-in that it is configured to use that project. By adding the "Spring Project Nature" to your project, you are enabling eclipse's spring plugin to work with your project.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++