Java EE

web application

- Web Applications are used to create dynamic websites.
- Java provides support for web application through **Servlets and JSPs**.
- We can create a website with static HTML pages but when we want information to be dynamic, we need web application.

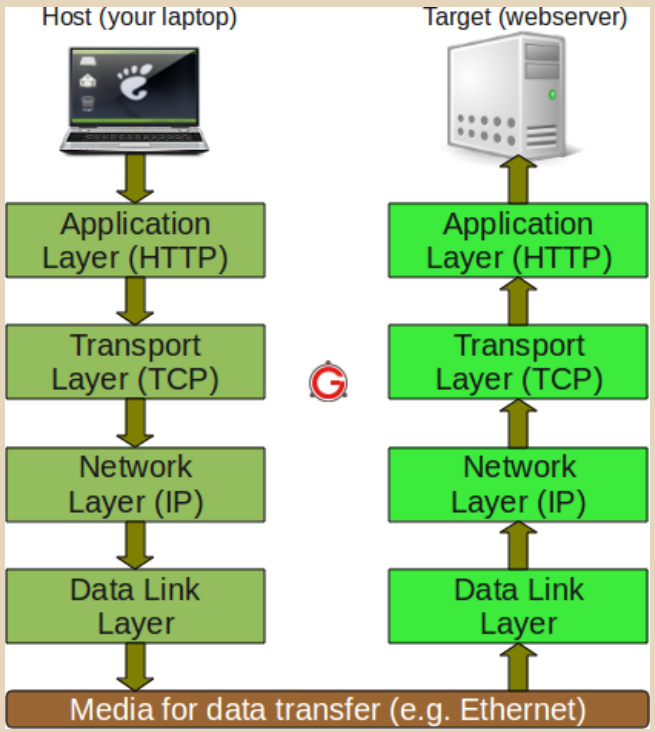
web server and client

- **Web Server** is a software that can process the client request and send the response back to the client.
- For example, **Apache** is one of the most widely used web server.
- Web Server runs on some physical machine and listens to client request on specific port.
- A **web client** is a software that helps in communicating with the server.
- Some of the most widely used web clients are **Firefox, Google Chrome, Safari** etc.
- When we request something from server (through URL), web client takes care of creating a request and sending it to server and then parsing the server response and present it to the user.

html and http

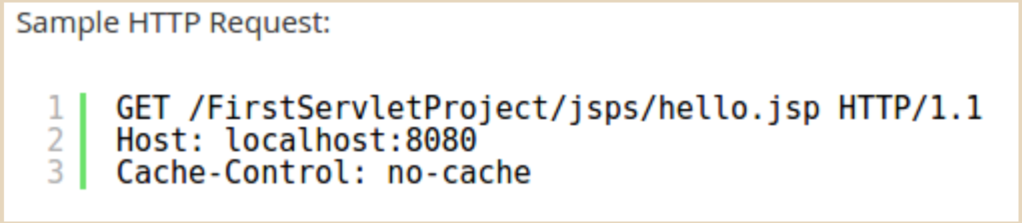
- Web Server and Web Client are two separate softwares, so there should be some common language for communication.
- **HTML** is the common language between server and client and stands for **HyperText Markup Language.**
- Web server and client needs a common communication protocol, **HTTP (HyperText Transfer Protocol)** is the communication protocol between server and client.
- HTTP runs on top of **TCP/IP** communication protocol.
- Some of the important parts of **HTTP Request** are:
    - **HTTP Method** - action to be performed, usually GET, POST, PUT etc.
    - **URL** - Page to access
    - **Form Parameters** - similar to arguments in a java method, for example user, password details from login page.

tcp/ip



img source: http://static.thegeekstuff.com/wp-content/uploads/2011/10/tcp-ip.png

html and http

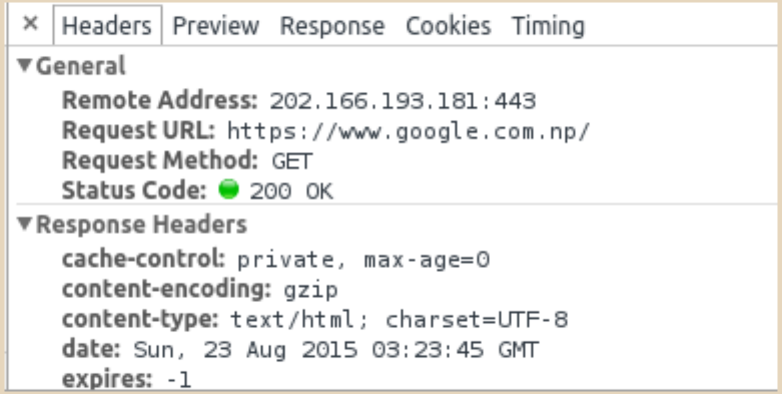

Sample HTTP Request:

```
GET /FirstServletProject/jsps/hello.jsp HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache
```

- Some of the important parts of **HTTP Response** are:
  - **Status Code** - an integer to indicate the state of the request.
    - 1xx Informational
    - 2xx Success
    - 3xx Redirection
    - 4xx Client Error
    - 5xx Server Error
  - **Content Type** - text, html, image, pdf etc. Also known as **MIME type.**
  - **Content** - actual data that is rendered by client and shown to user.

mime type or content type

- If you see sample HTTP response header, it contains tag “**Content-Type**”.
- It’s also called **MIME type** and server sends it to client to let them know the kind of data it’s sending.
- It helps client in rendering the data for user. Some of the mostly used mime types are **text/html, text/xml, application/xml** etc.



```
▼General
   Remote Address: 202.166.193.181:443
   Request URL: https://www.google.com.np/
   Request Method: GET
   Status Code: ● 200 OK
▼Response Headers
   cache-control: private, max-age=0
   content-encoding: gzip
   content-type: text/html; charset=UTF-8
   date: Sun, 23 Aug 2015 03:23:45 GMT
   expires: -1
```

understanding URL

- URL is acronym of **Universal Resource Locator** and it's used to locate the server and resource.
- Every resource on the web has it's own unique address.
- https://www.google.com.np/services/solutions/advertise.html
    - **http://** - This is the first part of URL and provides the communication protocol to be used in server-client communication.
    - **www.google.com.np** – 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 takes care of sending request to the particular server instance.

understanding url

- **80** - 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, 25 for SMTP etc.
- /services/solutions/advertise.html - Resource requested from server. It can be static html, pdf, JSP, servlets, PHP, ASP etc.

Servlet and JSPs

- Web servers are good for **static contents HTML** pages but they don’t know how to generate dynamic content or how to save data into databases, so we need another tool that we can use to generate dynamic content.
- There are several programming languages for dynamic content like PHP, Python, Ruby on Rails, **Java Servlets and JSPs**.
- Java Servlet and JSPs are server side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.
- Since servlet is a server side technology, we will need a web container that supports Servlet technology, so we will use **Apache Tomcat** server.
- Create servlet and run it on tomcat server.

Servlet

- At the lowest level it is nothing but a Java class.
- Servlet is a server-side technology in Java platform stack.
- Servlet responds to an incoming request from a browser client with a response.
- Though initially meant for generating HTML content in web application, nowadays used mostly at the controller layers.
- Before Servlet 3, we need to provide the url pattern information in **web application deployment descriptor** but **servlet 3.0** uses java annotations that is easy to understand and chances of errors are less.

web container

- Tomcat is a **web container**, when a request is made from Client to web server, it passes the request to web container and it’s web container job to find the correct resource to handle the request (servlet or JSP) and then use the response from the resource to generate the response and provide it to web server. Then web server sends the response back to the client.
- When web container gets the request and if it’s for servlet container creates two Objects **HTTPServletRequest** and **HTTPServletResponse**.
- Then it finds the correct servlet based on the URL and creates a thread for the request.
- Then it invokes the servlet **service**() method and based on the HTTP method service() method invokes **doGet**() or **doPost**() methods.
- Servlet methods generate the dynamic page and write it to response.
- Once servlet thread is complete, container converts the response to HTTP response and send it back to client.

web container

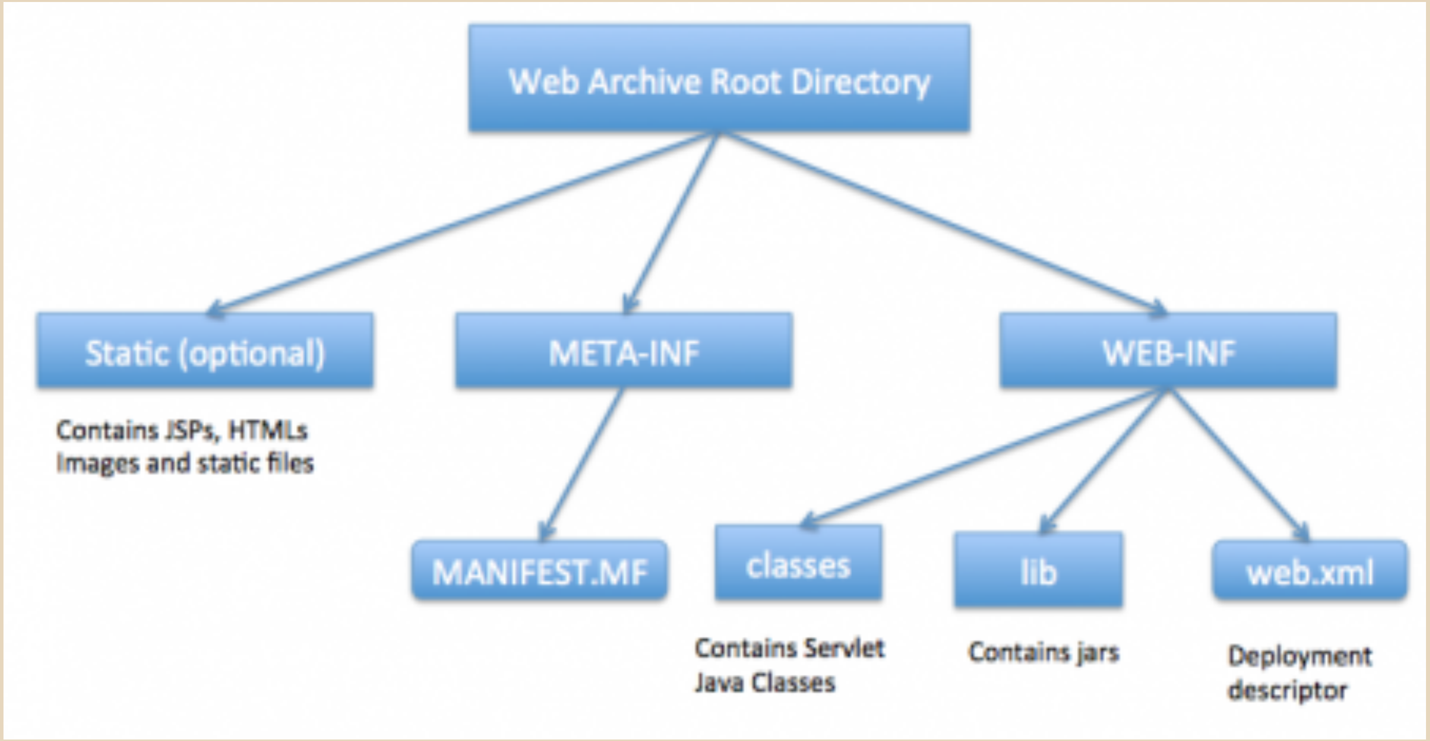
- **Communication Support -** Container provides easy way of communication between web server and the servlets and JSPs.
- **Lifecycle and Resource Management -** Container takes care of managing the life cycle of servlet. Container takes care of loading the servlets into memory, initializing servlets, invoking servlet methods and destroying them. Container also provides utility like JNDI for resource pooling and management.
- **Multithreading Support -** Container creates new thread for every request to the servlet and when it’s processed the thread dies. So servlets are not initialized for each request and saves time and memory.[CGI]

web container

- **JSP Support -** JSPs doesn’t look like normal java classes and web container provides support for JSP. Every JSP in the application is compiled by container and converted to Servlet and then container manages them like other servlets.
- **Miscellaneous Task** - Web container manages the resource pool, does memory optimizations, run garbage collector, provides security configurations, support for multiple applications, hot deployment and several other tasks behind the scene that makes our life easier.

- **Deployment Descriptor: web.xml** file is the deployment descriptor of the web application and contains mapping for servlets (prior to 3.0), welcome pages, security configurations, session timeout settings etc.

web directory structure



Servlet

- Servlet is JEE server driven technology to create web applications in java.
- **The javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing our own servlets.
- All servlets must implement the **javax.servlet.Servlet** interface, which defines servlet lifecycle methods.
- When implementing a generic service, we can extend the **GenericServlet** class provided with the Java Servlet API.
- **HttpServlet** class provides methods, such as **doGet()** and **doPost()**, for handling HTTP-specific services.
- Most of the times, web applications are accessed using HTTP protocol and thats why we mostly extend **HttpServlet** class.

Common Gateway Interface

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

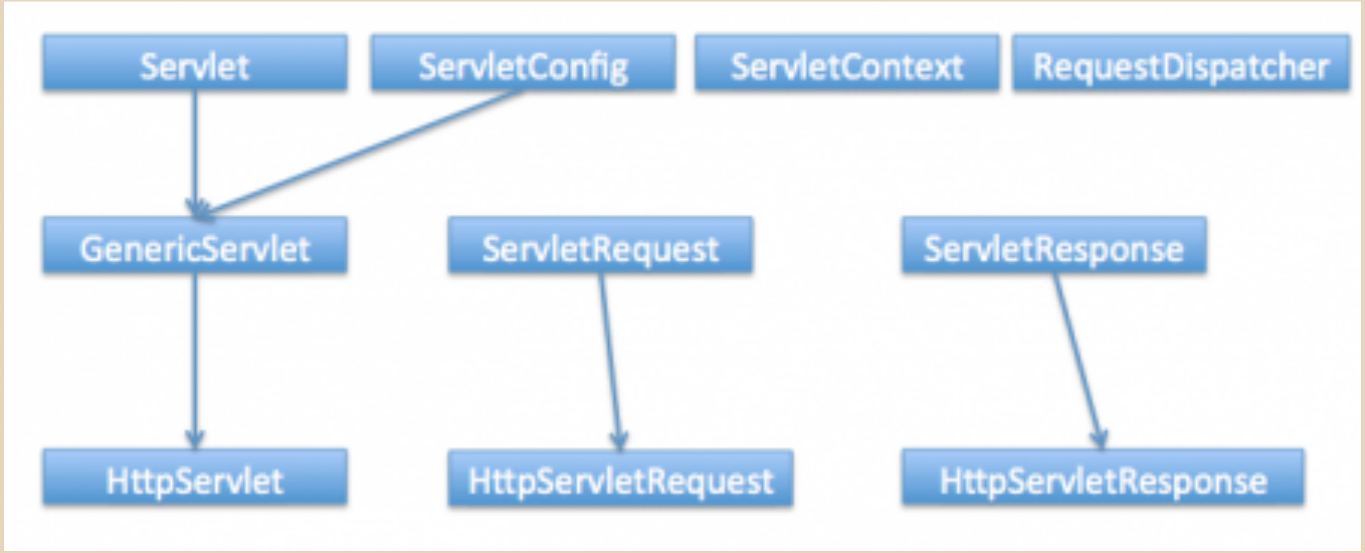
Servlet vs CGI

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

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

Servlet hierarchy

- **javax.servlet.Servlet** is the base interface of Servlet API.
- With Servlet 3.0 specs, servlet API introduced use of annotations rather than having all the servlet configuration in deployment descriptor.



Servlet interface

- **javax.servlet.Servlet** is the base interface of Java Servlet API.
- Servlet interface declares the **life cycle methods** of servlet.
- The methods declared in this interface are:
  - public abstract void **init**(ServletConfig paramServletConfig) throws ServletException
    - This is the very important method that is invoked by servlet container to initialize the servlet and ServletConfig parameters.
    - The servlet is not ready to process client request until unless init() method is finished executing.
    - This method is called only once in servlet lifecycle and make Servlet class different from normal java objects.
    - We can extend this method in our servlet classes to initialize resources such as DB Connection, Socket connection etc.

Servlet interface

- public abstract ServletConfig **getServletConfig()**
  - This method returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet.
  - We can use this method to get the init parameters of servlet defines in deployment descriptor (**web.xml**) or through **annotation** in Servlet 3.
- public abstract void **service**(ServletRequest req, ServletResponse res) throws ServletException, IOException
  - This method is responsible for processing the client request.
  - Whenever servlet container receives any request, it creates a new thread and execute the service() method by passing request and response as argument.
  - Servlets usually run in multi-threaded environment, so it’s developer responsibility to keep shared resources thread-safe using synchronization.

Servlet interface

- public abstract String **getServletInfo**()
  - This method returns string containing information about the servlet, such as its author, version, and copyright.
  - The string returned should be plain text and can’t have markups.
- public abstract void **destroy**()
  - This method can be called only once in servlet life cycle and used to close any open resources.
  - This is like finalize method of a java class.

ServletConfig interface

- **javax.servlet.ServletConfig** is used to pass configuration information to Servlet.
- Every servlet has it's own ServletConfig object and servlet container is responsible for instantiating this object.
- We can provide servlet init parameters in **web.xml** file or through use of **WebInitParam** annotation.
- We can use **getServletConfig**() method to get the ServletConfig object of the servlet.
- The important methods of ServletConfig interface are:
    - public abstract ServletContext **getServletContext**()
        - This method returns the ServletContext object for the servlet.
    - public abstract Enumeration **getInitParameterNames**()
        - This method returns the Enumeration of name of init parameters defined for the servlet.
        - If there are no init parameters defined, this method returns empty enumeration.

ServletConfig interface

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

ServletContext interface

- javax.servlet.**ServletContext** interface provides access to web application variables to the servlet.
- The **ServletContext** is unique object and available to all the servlets in the web application.
- When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using <context-param> element.
- We can get the ServletContext object via the **getServletContext**() method of ServletConfig.
- Servlet engines may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the URL path namespace of the host.
- Some of the important methods of ServletContext are:
  - public abstract ServletContext **getContext**(String uripath)
    - This method returns ServletContext object for a particular uripath or null if not available or not visible to the servlet.

ServletContext interface

- public abstract URL **getResource**(String path) throws MalformedURLException
  - This method return URL object allowing access to any content resource requested.
  - We can access items whether they reside on the local file system, a remote file system, a database, or a remote network site without knowing the specific details of how to obtain the resources.
- public abstract InputStream **getResourceAsStream**(String path)
  - This method returns an inputstream to the given resource path or null if not found.
- public abstract RequestDispatcher **getRequestDispatcher**(String urlpath)
  - This method is mostly used to obtain a reference to another servlet.
  - After obtaining a RequestDispatcher, the servlet programmer forward a request to the target component or include content from it.
- public abstract void **log**(String msg)
  - This method is used to write given message string to the servlet log file.

ServletContext interface

- public abstract Object **getAttribute**(String name)
  - Return the object attribute for the given name.
  - We can get enumeration of all the attributes using public abstract Enumeration getAttributeNames() method.
- public abstract void **setAttribute**(String paramString, Object paramObject)
  - This method is used to set the attribute with application scope.
  - The attribute will be accessible to all the other servlets having access to this ServletContext.
  - We can remove an attribute using public abstract void removeAttribute(String paramString) method.

ServletContext interface

- String **getInitParameter**(String name)
    - This method returns the String value for the init parameter defined with name in web.xml, returns null if parameter name doesn’t exist.
    - We can use Enumeration getInitParameterNames() to get enumeration of all the init parameter names.
- boolean **setInitParameter**(String paramString1, String paramString2)
    - We can use this method to set init parameters to the application.

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

ServletRequest interface

- Protocol agnostic interfaces - GenericServlet and ServletRequest, ServletResponse
- ServletRequest interface is used to provide client request information to the servlet.
- Servlet container creates **ServletRequest** object from client request and pass it to the servlet **service()** method for processing.
- Some of the important methods of ServletRequest interface are:
    - Object **getAttribute**(String name)
        - This method returns the value of named attribute as Object and null if it’s not present.
        - We can use **getAttributeNames**() method to get the enumeration of attribute names for the request.
        - This interface also provide methods for setting and removing attributes.

ServletRequest interface

- String **getParameter**(String name)
  - This method returns the request parameter as String. We can use getParameterNames() method to get the enumeration of parameter names for the request.
- String **getServerName**()
  - Returns the hostname of the server.
- int **getServerPort**()
  - Returns the port number of the server on which it’s listening.
- The child interface of ServletRequest is HttpServletRequest that contains some other methods for session management, cookies and authorization of request.

ServletResponse interface

- ServletResponse interface is used by servlet in sending response to the client.
- Servlet container creates the ServletResponse object and pass it to servlet service() method and later use the response object to generate the HTML response for client.
- Some of the important methods in HttpServletResponse are:
  - void **addCookie**(Cookie cookie)
    - Used to add cookie to the response.
  - void **addHeader**(String name, String value)
    - Used to add a response header with the given name and value.
  - String **encodeURL**(java.lang.String url)
    - Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

ServletResponse interface

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

ServletDispatcher interface

- RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in the same context.
- We can also use this to include the content of another resource to the response.
- This interface is used for servlet communication within the same context.
- There are two methods defined in this interface:
  - void **forward**(ServletRequest request, ServletResponse response)
    - forwards the request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
  - void **include**(ServletRequest request, ServletResponse response)
    - includes the content of a resource (servlet, JSP page, HTML file) in the response.
- We can get RequestDispatcher in a servlet using ServletContext getRequestDispatcher(String path) method. The path must begin with a / and is interpreted as relative to the current context root.

GenericServlet class

- GenericServlet is an abstract class that implements Servlet, ServletConfig and Serializable interface.
- GenericServlet provide default implementation of all the Servlet life cycle methods and ServletConfig methods and makes our life easier when we extend this class, we need to override only the methods we want and rest of them we can work with the default implementation.
- Most of the methods defined in this class are only for easy access to common methods defined in Servlet and ServletConfig interfaces.
- One of the important method in GenericServlet class is no-argument init() method and we should override this method in our servlet program if we have to initialize some resources before processing any request from servlet.

HTTPServlet class

- Protocol Specific Class [Cast to HTTP]
- HTTPServlet is an abstract class that extends GenericServlet and provides base for creating HTTP based web applications.
- There are methods defined to be overridden by subclasses for different HTTP methods:
    - doGet(), for HTTP GET requests
    - doPost(), for HTTP POST requests
    - doPut(), for HTTP PUT requests
    - doDelete(), for HTTP DELETE requests

Servlet Attributes

- Servlet attributes are used for **inter-servlet communication**, we can set, get and remove attributes in web application.
- There are three scopes for servlet attributes
  - request scope,
  - session scope and
  - application scope.
- ServletRequest, HttpSession and ServletContext interfaces provide methods to get/set/remove attributes from request, session and application scope respectively.
- Servlet attributes are different from init parameters defined in web.xml for ServletConfig or ServletContext.

Servlet Annotations

- Prior to Servlet 3, all the servlet mapping and it’s init parameters were used to defined in web.xml file, this was not convenient and more error prone when number of servlets are huge in an application.
- Servlet 3 introduced use of java annotations to define a servlet, filter and listener servlets and init parameters. Some of the important Servlet API annotations are:
  - **WebServlet**
    - We can use this annotation with Servlet classes to define init parameters, loadOnStartup value, description and url patterns etc.
    - At least one URL pattern MUST be declared in either the value or urlPattern attribute of the annotation, but not both.
    - The class on which this annotation is declared MUST extend HttpServlet.

Servlet Annotations

- **WebInitParam**
  - This annotation is used to define init parameters for servlet or filter, it contains name, value pair and we can provide description also.
  - This annotation can be used within a WebFilter or WebServlet annotation.
- **WebFilter**
  - This annotation is used to declare a servlet filter. This annotation is processed by the container during deployment, the Filter class in which it is found will be created as per the configuration and applied to the URL patterns, Servlets and DispatcherTypes.
  - The annotated class MUST implement javax.servlet.Filter interface.
- **WebListener**
  - The annotation used to declare a listener for various types of event, in a given web application context.

Read Form Data

- **GET** and **POST** methods are used to pass information from a form to a Java Servlet.
- While using GET method the form data is passed in the url as query parameters.
- GET method is the default method used. It looks like,

    http://localhost:8080/servlet?**key1**=**value1**&**key2**=**value2**

- If **POST** method is used then the form data is passed in the HTTP request message body.
- It cannot be seen in the URL as seen in GET method.
- URL length has a limitation and so if we are passing large volume of data we should use POST.
- Similarly sensitive data like passwords should also be passed using POST method.

session management

- **HTTP protocol and Web Servers are stateless**, what it means is that for web server every request is a new request to process and they can’t identify if it’s coming from client that has been sending request previously.
- **session** is a conversion between a server and a client.
- The only way to maintain a session is when some unique information about the session **(session id)** is passed between server and client in every request and response.
- Servlet API will use one of the underlying traditional mechanisms like **cookies**, **URL rewriting**, but that will happen behind the scenes and you need not worry about it!

Session ID

- There are several ways we can provide unique identifier in request and response:
- **User Authentication**
  - This is not very effective method because it won't work if the same user is logged in from different browsers.
- **HTML Hidden Field**
  - This method can’t be used with links because it needs the form to be submitted every time request is made from client to server with the hidden field.
  - **It’s not secure.**
- **URL Rewriting**
  - We can append a session identifier parameter with every request and response to keep track of the session.
  - This is very tedious because we need to keep track of this parameter in every response and make sure it’s not clashing with other parameters.

Session ID

- **Cookies**
  - Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies.
  - When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session.
  - **We can maintain a session with cookies but if the client disables the cookies, then it won’t work.**
- **Session Management API**
  - Session Management API is built on top of above methods for session tracking. Some of the major disadvantages of all the above methods are:
  - Most of the time we don’t want to only track the session, we have to store some data into the session that we can use in future requests. This will require a lot of effort if we try to implement this.

Session ID

- All the above methods are not complete in themselves, all of them won’t work in a particular scenario.
- So we need a solution that can utilize these methods of session tracking to provide session management in all cases.
- **That’s why we need Session Management API and J2EE Servlet technology comes with session management API that we can use.**

JSESSIONID

- **JSESSIONID** cookie is created/sent when session is created.
- Session is created when your code calls **request.getSession()** or **request.getSession (true)** for the first time.
- If you just want get session, but not create it if it doesn't exists, use **request.getSession (false)** - this will return you a session or null. In this case, new session is not created, and JSESSIONID cookie is not sent.

Note: If you look at the generated Java code corresponding to a JSP in the work directory under Tomcat, It appears that, whether you like it or not, if you invoke a JSP from a servlet, JSESSIONID will get created!

- Adding the following JSP directive, you can disable the setting of JSESSIONID by a JSP.

```
<%@ page session="false" %>
```

- Session information is scoped only to the current web application (ServletContext), so information stored in one context will not be directly visible in another.

Filter

- We learned how we can manage session in web application and if we want to make sure that a resource is accessible only when user session is valid, we can achieve this using servlet session attributes.
- The approach is simple but if we have a lot of servlets and jsps, then it will become hard to maintain because of redundant code.
- If we want to change the attribute name in future, we will have to change all the places where we have session authentication.
- That’s why we have **servlet filter**.
- Servlet Filters are pluggable java components that we can use to intercept and process requests before they are sent to servlets and response after servlet code is finished and before container sends the response back to the client.

Filter - importance

- Logging request parameters to log files.
- Authentication and authorization of request for resources.
- Formatting of request body or header before sending it to servlet.
- Compressing the response data sent to the client.
- Alter response by adding some cookies, header information etc.
- As I mentioned earlier, servlet filters are pluggable and configured in deployment descriptor (web.xml) file.
- Servlets and filters both are unaware of each other and we can add or remove a filter just by editing web.xml.
- We can have multiple filters for a single resource and we can create a chain of filters for a single resource in web.xml.
- We can create a Servlet Filter by implementing **javax.servlet.Filter interface**.

Filter - interface

- **Filter** interface is similar to Servlet interface and we need to implement it to create our own servlet filter.
- Filter interface contains lifecycle methods of a Filter and it’s managed by servlet container.
- void **init**(FilterConfig paramFilterConfig)
- **doFilter**(ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain)
    - FilterChain is used to invoke the next filter in the chain. This is a great example of Chain of Responsibility Pattern.
- void **destroy**()
    - When container offloads the Filter instance, it invokes the destroy() method.
    - This is the method where we can close any resources opened by filter.

WebFilter - annotation

- javax.servlet.annotation.**WebFilter** was introduced in Servlet 3.0 and we can use this annotation to declare a servlet filter.
- We can use this annotation to define init parameters, filter name and description, servlets, url patterns and dispatcher types to apply the filter.

Servlet Listener

- Servlet API provides Listener interfaces that we can implement and configure to listen to an event and do certain operations.
- **Event** is occurrence of something, in web application world an event can be initialization of application, destroying an application, request from client, creating/destroying a session, attribute modification in session etc.
- **Servlet** API provides different types of Listener interfaces that we can implement and configure in web.xml to process something when a particular event occurs.

Servlet Listener Events

- javax.servlet.**AsyncEvent -** startAsync or stopAsync
- javax.servlet.http.**HttpSessionBindingEvent** - HttpSession.setAttribute HttpSession.removeAttribute.
- javax.servlet.http.**HttpSessionEvent** - changes to session
- javax.servlet.**ServletContextAttributeEvent** - changes to the attributes of the ServletContex javax.servlet.**ServletContextEvent** - changes to the servlet context of a web application.
- javax.servlet.**ServletRequestEvent** - Events of this kind indicate lifecycle events for a ServletRequest. The source of the event is the ServletContext of this web application.
- javax.servlet.**ServletRequestAttributeEvent** - changes to the attributes of the servlet request in an application.

Servlet Listeners

- javax.servlet.**AsyncListener**
- javax.servlet.**ServletContextListener**
- javax.servlet.**ServletContextAttributeListener**
- javax.servlet.**ServletRequestListener**
- javax.servlet.**ServletRequestAttributeListener**
- javax.servlet.http.**HttpSessionListener**
- javax.servlet.http.**HttpSessionBindingListener**
- javax.servlet.http.**HttpSessionAttributeListener**
- javax.servlet.http.**HttpSessionActivationListener**