1. **What is the Servlet?**

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request- response programming model.

**2. What are the new features added to Servlet 2.5?**

Following are the changes introduced in Servlet 2.5:

* A new dependency on J2SE 5.0
* Support for annotations
* Loading the class
* Several web.xml conveniences
* A handful of removed restrictions
* Some edge case clarifications

**3. What are the uses of Servlet?**

Typical uses for HTTP Servlets include:

* Processing and/or storing data submitted by an HTML form.
* Providing dynamic content, e.g. returning the results of a database query to the client.
* A Servlet can handle multiple request concurrently and be used to develop high performance system
* Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

**4. What are the advantages of Servlet over CGI?**

Servlets have several advantages over CGI:

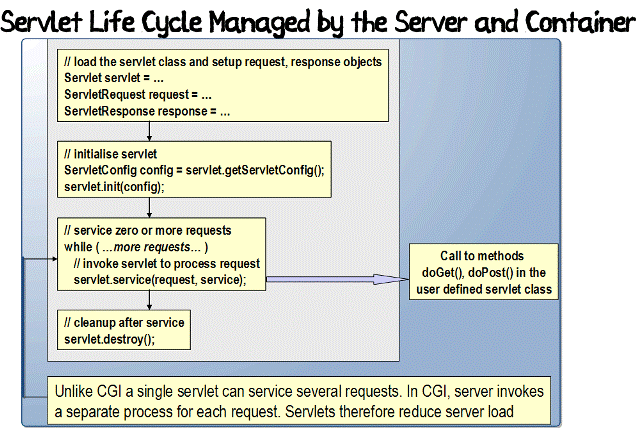
* A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.
* A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.
* There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
* Several web.xml conveniences
* A handful of removed restrictions
* Some edge case clarifications

**5. What are the phases of the servlet life cycle?**

The life cycle of a servlet consists of the following phases:

* **Servlet class loading** : For each servlet defined in the deployment descriptor of the Web application, the servlet container locates and loads a class of the type of the servlet. This can happen when the servlet engine itself is started, or later when a client request is actually delegated to the servlet.
* **Servlet instantiation** : After loading, it instantiates one or more object instances of the servlet class to service the client requests.
* **Initialization (call the init method)** : After instantiation, the container initializes a servlet before it is ready to handle client requests. The container initializes the servlet by invoking its init() method, passing an object implementing the ServletConfig interface. In the init() method, the servlet can read configuration parameters from the deployment descriptor or perform any other one-time activities, so the init() method is invoked once and only once by the servlet container.
* **Request handling (call the service method)** : After the servlet is initialized, the container may keep it ready for handling client requests. When client requests arrive, they are delegated to the servlet through the service() method, passing the request and response objects as parameters. In the case of HTTP requests, the request and response objects are implementations of HttpServletRequest and HttpServletResponse respectively. In the HttpServlet class, the service() method invokes a different handler method for each type of HTTP request, doGet() method for GET requests, doPost() method for POST requests, and so on.
* **Removal from service (call the destroy method)** : A servlet container may decide to remove a servlet from service for various reasons, such as to conserve memory resources. To do this, the servlet container calls the destroy() method on the servlet. Once the destroy() method has been called, the servlet may not service any more client requests. Now the servlet instance is eligible for garbage collection

The life cycle of a servlet is controlled by the container in which the servlet has been deployed.



**6. Why do we need a constructor in a servlet if we use the init method?**

Even though there is an init method in a servlet which gets called to initialize it, a constructor is still required to instantiate the servlet. Even though you as the developer would never need to explicitly call the servlet's constructor, it is still being used by the container (the container still uses the constructor to create an instance of the servlet). Just like a normal POJO (plain old java object) that might have an init method, it is no use calling the init method if you haven't constructed an object to call it on yet.

**7. How the servlet is loaded?**

A servlet can be loaded when:

* First request is made.
* Server starts up (auto-load).
* There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
* Administrator manually loads.

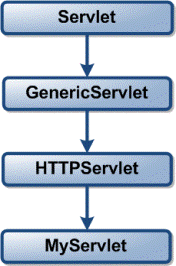
**8. How a Servlet is unloaded?**

A servlet is unloaded when:

* Server shuts down.
* Administrator manually unloads.

**9. What is Servlet interface?**

The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or , more commonly by extending a class that implements it.



Note: Most Servlets, however, extend one of the standard implementations of that interface, namely javax.servlet.GenericServlet andjavax.servlet.http.HttpServlet.

|  |
| --- |
|  |

**10. What is the GenericServlet class?**

GenericServlet is an abstract class that implements the Servlet interface and the ServletConfig interface. In addition to the methods declared in these two interfaces, this class also provides simple versions of the lifecycle methods init and destroy, and implements the log method declared in the ServletContext interface.   
Note: This class is known as generic servlet, since it is not specific to any protocol.

**11.What's the difference between GenericServlet and HttpServlet?**

|  |  |
| --- | --- |
| **GenericServlet** | **HttpServlet** |
| The GenericServlet is an abstract class that is extended by HttpServlet to provide HTTP protocol-specific methods. | An abstract class that simplifies writing HTTP servlets.  It extends the GenericServlet base class and provides an  framework for handling the HTTP protocol. |
| The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers. | The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method. |
| GenericServlet is not specific to any protocol. | HttpServlet only supports HTTP and HTTPS protocol. |

|  |
| --- |
|  |

**12. Why is HttpServlet declared abstract?**

The HttpServlet class is declared abstract because the default implementations of the main service methods do nothing and must be overridden. This is a convenience implementation of the Servlet interface, which means that developers do not need to implement all service methods. If your servlet is required to handle doGet() requests for example, there is no need to write a doPost() method too.

**13. Can servlet have a constructor ?**

One can definitely have constructor in servlet.Even you can use the constrctor in servlet for initialization purpose,but this type of approch is not so common. You can perform common operations with the constructor as you normally do.The only thing is that you cannot call that constructor explicitly by the new keyword as we normally do.In the case of servlet, servlet container is responsible for instantiating the servlet, so the constructor is also called by servlet container only.

**14. What are the types of protocols supported by HttpServlet ?**

It extends the GenericServlet base class and provides a framework for handling the HTTP protocol. So, HttpServlet only supports HTTP and HTTPS protocol.

**15. What is the difference between doGet() and doPost()?**

|  |  |  |
| --- | --- | --- |
| **#** | **doGet()** | **doPost()** |
| 1 | In doGet() the parameters are appended to the URL and sent along with header information. | In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar. |
| 2 | The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters. | You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects! |
| 3 | doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent) | doPost() provides information (such as placing an order for merchandise) that the server is expected to remember |
| 4 | Parameters are not encrypted | Parameters are encrypted |
| 5 | doGet() is faster if we set the response content length since the same connection is used. Thus increasing the performance | doPost() is generally used to update or post some information to the server.doPost is slower compared to doGet since doPost does not write the content length |
| 6 | doGet() should be idempotent. i.e. doget should be able to be repeated safely many times | This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable. |
| 7 | doGet() should be safe without any side effects for which user is held responsible | This method does not need to be either safe |
| 8 | It allows bookmarks. | It disallows bookmarks. |

**16. When to use doGet() and when doPost()?**

Always prefer to use GET (As because GET is faster than POST), except mentioned in the following reason:

* If data is sensitive
* Data is greater than 1024 characters
* If your application don't need bookmarks.

**17. How do I support both GET and POST from the same Servlet?**

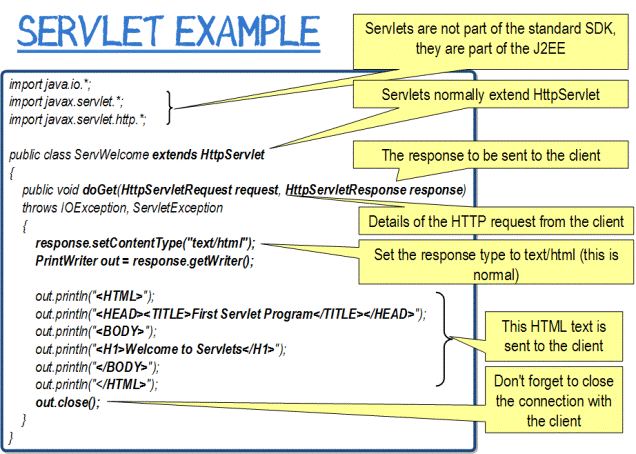
The easy way is, just support POST, then have your doGet method call your doPost method:

**public void doGet(HttpServletRequest request, HttpServletResponse response)  
                        throws ServletException, IOException  
{  
    doPost(request, response);     
}**

**18. Should I override the service() method?**

We never override the service method, since the HTTP Servlets have already taken care of it . The default service function invokes the doXXX() method corresponding to the method of the HTTP request.For example, if the HTTP request method is GET, doGet() method is called by default. A servlet should override the doXXX() method for the HTTP methods that servlet supports. Because HTTP service method check the request method and calls the appropriate handler method, it is not necessary to override the service method itself. Only override the appropriate doXXX() method.

**19. How the typical servlet code look like ?**



**20.What is a servlet context object?**

A servlet context object contains the information about the Web application of which the servlet is a part. It also provides access to the resources common to all the servlets in the application. Each Web application in a container has a single servlet context associated with it.

**21. What are the differences between the ServletConfig interface and the ServletContext interface?**

|  |  |
| --- | --- |
| **ServletConfig** | **ServletContext** |
| The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method. | A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container. |
| There is one ServletConfig parameter per servlet. | There is one ServletContext for the entire webapp and all the servlets in a webapp share it. |
| The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file | The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file. |

**22. What's the difference between forward() and sendRedirect() methods?**

|  |  |  |
| --- | --- | --- |
| **forward()** | | **sendRedirect()** |
| A forward is performed internally by the servlet. | A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original. | |
| The browser is completely unaware that it has taken place, so its original URL remains intact. | The browser, in this case, is doing the work and knows that it's making a new request. | |
| Any browser reload of the resulting page will simple repeat the original request, with the original URL | A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL. | |
| Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable) | This method can be used to redirect users to resources that are not part of the current context, or even in the same domain. | |
| Since both resources are part of same context, the original request context is retained | Because this involves a new request, the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables will need to be passed by via the session object). | |
| Forward is  marginally faster than redirect. | redirect is marginally slower than a forward, since it requires two browser requests, not one. | |

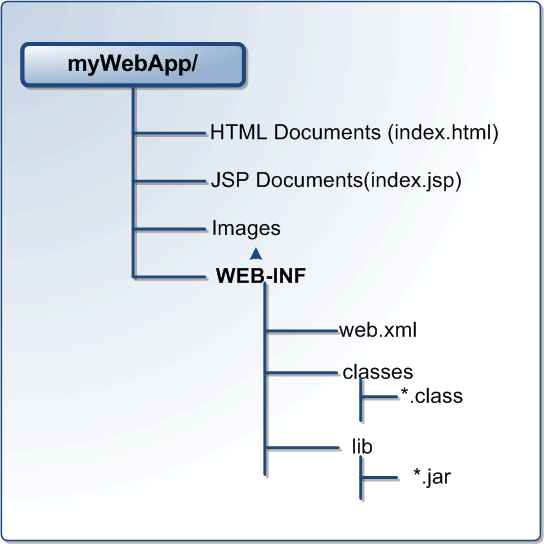
**23. What is the difference between the include() and forward() methods?**

|  |  |
| --- | --- |
| **include()** | **forward()** |
| The RequestDispatcher include() method inserts the the contents of the specified resource directly in the flow of the servlet response, as if it were part of the calling servlet. | The RequestDispatcher forward() method is used to show a different resource in place of the servlet that was originally called. |
| If you include a servlet or JSP document, the included resource must not attempt to change the response status code or HTTP headers, any such request will be ignored. | The forwarded resource may be another servlet, JSP or static HTML document, but the response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection. |
| The include() method is often used to include common "boilerplate" text or template markup that may be included by many servlets. | The forward() method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a particular response page. |

**24.What's the use of the servlet wrapper classes??**

The HttpServletRequestWrapper and HttpServletResponseWrapper classes are designed to make it easy for developers to create custom implementations of the servlet request and response types. The classes are constructed with the standard HttpServletRequest and HttpServletResponse instances respectively and their default behaviour is to pass all method calls directly to the underlying objects.

**25.What is the directory structure of a WAR file?**



**26. What is a deployment descriptor?**

A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. It declares transaction attributes and security authorization for an enterprise bean. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean.  
The JavaEE server reads the deployment descriptor at run time and acts upon the component accordingly.

**27. What is the difference between the getRequestDispatcher(String path) method of javax.servlet.ServletRequest interface and javax.servlet.ServletContext interface?**

|  |  |
| --- | --- |
| **ServletRequest.getRequestDispatcher(String path)** | **ServletContext.getRequestDispatcher(String path)** |
| The getRequestDispatcher(String path) method ofjavax.servlet.ServletRequest interface accepts parameter the path to the resource to be included or forwarded to, which can be relative to the request of the calling servlet. If the path begins with a “/” it is interpreted as relative to the current context root. | The getRequestDispatcher(String path) method ofjavax.servlet.ServletContext interface cannot accept relative paths. All path must start with a “/” and are   interpreted as relative to current context root. |

|  |
| --- |
|  |

**28. What is preinitialization of a servlet?**

A container does not initialize the servlets as soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading. The servlet specification defines the element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up. The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

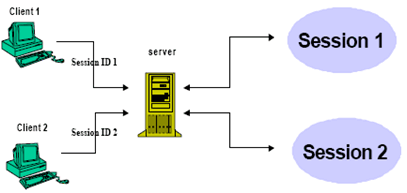
**29. What is the <load-on-startup> element?**

The <load-on-startup> element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

<servlet>  
   <servlet-name>ServletName</servlet-name>  
   <servlet-class>ClassName</servlet-class>  
   <load-on-startup>1</load-on-startup>  
</servlet>

**30. What is session?**

A session refers to all the requests that a single client might make to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.



**31. What is Session Tracking?**

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

**32. What is the need of Session Tracking in web application?abh**

HTTP is a stateless protocol i.e., every request is treated as new request. For web applications to be more realistic they have to retain information across multiple requests. Such information which is part of the application is reffered as "state". To keep track of this state we need session tracking.   
  
Typical example: Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests.

**33. What are the types of Session Tracking ?**

Sessions need to work with all web browsers and take into account the users security preferences. Therefore there are a variety of ways to send and receive the identifier:

* **URL rewriting :**URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies.
* **Hidden Form Fields :** Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client.
* **Cookies :**Cookie is a small amount of information sent by a servlet to a Web browser. Saved by the browser, and later sent back to the server in subsequent requests. A cookie has a name, a single value, and optional attributes. A cookie's value can uniquely identify a client.
* **Secure Socket Layer (SSL) Sessions :** Web browsers that support Secure Socket Layer communication can use SSL's support via HTTPS for generating a unique session key as part of the encrypted conversation.

**34. How do I use cookies to store session state on the client?**

In a servlet, the HttpServletResponse and HttpServletRequest objects passed to method HttpServlet.service() can be used to create cookies on the client and use cookie information transmitted during client requests. JSPs can also use cookies, in scriptlet code or, preferably, from within custom tag code.

* To set a cookie on the client, use the addCookie() method in class HttpServletResponse. Multiple cookies may be set for the same request, and a single cookie name may have multiple values.
* To get all of the cookies associated with a single HTTP request, use the getCookies() method of class HttpServletRequest

**35. What are some advantages of storing session state in cookies?**

* Cookies are usually persistent, so for low-security sites, user data that needs to be stored long-term (such as a user ID, historical information, etc.) can be maintained easily with no server interaction.
* For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

**36. What are some disadvantages of storing session state in cookies?**

* Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.
* All data for a session are kept on the client. Corruption, expiration or purging of cookie files can all result in incomplete, inconsistent, or missing information.
* Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, the browser may be behind a firewall that filters cookies, and so on. Servlets and JSP pages that rely exclusively on cookies for client-side session state will not operate properly for all clients. Using cookies, and then switching to an alternate client-side session state strategy in cases where cookies aren't available, complicates development and maintenance.
* Browser instances share cookies, so users cannot have multiple simultaneous sessions.
* Cookie-based solutions work only for HTTP clients. This is because cookies are a feature of the HTTP protocol. Notice that the while package javax.servlet.http supports session management (via class HttpSession), packagejavax.servlet has no such support.

**37.What is URL rewriting?**

URL rewriting is a method of session tracking in which some extra data is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.

Every URL on the page must be encoded using method HttpServletResponse.encodeURL(). Each time a URL is output, the servlet passes the URL to encodeURL(), which encodes session ID in the URL if the browser isn't accepting cookies, or if the session tracking is turned off.  
E.g., http://abc/path/index.jsp;jsessionid=123465hfhs

**Advantages**

* URL rewriting works just about everywhere, especially when cookies are turned off.
* Multiple simultaneous sessions are possible for a single user. Session information is local to each browser instance, since it's stored in URLs in each page being displayed. This scheme isn't foolproof, though, since users can start a new browser instance using a URL for an active session, and confuse the server by interacting with the same session through two instances.
* Entirely static pages cannot be used with URL rewriting, since every link must be dynamically written with the session state. It is possible to combine static and dynamic content, using (for example) templating or server-side includes. This limitation is also a barrier to integrating legacy web pages with newer, servlet-based pages.

**DisAdvantages**

* Every URL on a page which needs the session information must be rewritten each time a page is served. Not only is this expensive computationally, but it can greatly increase communication overhead.
* URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.
* URL rewriting does not work well with JSP technology.
* If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

**38 .How can an existing session be invalidated?**

An existing session can be invalidated in the following two ways:

* Setting timeout in the deployment descriptor: This can be done by specifying timeout between the <session-timeout>tags as follows:

<session-config>   
       <**session-timeout>10</session-timeout>**  
</session-config>

This will set the time for session timeout to be ten minutes.

* Setting timeout programmatically: This will set the timeout for a specific session. The syntax for setting the timeout programmatically is as follows:

public void setMaxInactiveInterval(int interval)

The setMaxInactiveInterval() method sets the maximum time in seconds before a session becomes invalid.   
Note :Setting the inactive period as negative(-1), makes the container stop tracking session, i.e, session never expires.

**39. How can the session in Servlet can be destroyed?**

An existing session can be destroyed in the following two ways:

* Programatically : Using session.invalidate() method, which makes the container abonden the session on which the method is called.
* When the server itself is shutdown.

**40. A client sends requests to two different web components. Both of the components access the session. Will they end up using the same session object or different session ?**

Creates only one session i.e., they end up with using same session .

Sessions is specific to the client but not the web components. And there is a 1-1 mapping between client and a session.

**41. What is servlet lazy loading?**

* A container doesnot initialize the servlets ass soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading.
* The servlet specification defines the <load-on-startup> element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up.
* The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

**42. What is Servlet Chaining?**

Servlet Chaining is a method where the output of one servlet is piped into a second servlet. The output of the second servlet could be piped into a third servlet, and so on. The last servlet in the chain returns the output to the Web browser.

**43. How are filters?**

Filters are Java components that are used to intercept an incoming request to a Web resource and a response sent back from the resource. It is used to abstract any useful information contained in the request or response. Some of the important functions performed by filters are as follows:

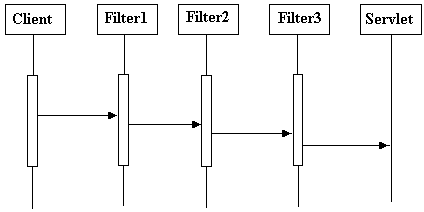
* Security checks
* Modifying the request or response
* Data compression
* Logging and auditing
* Response compression

Filters are configured in the deployment descriptor of a Web application. Hence, a user is not required to recompile anything to change the input or output of the Web application.

**44. What are the functions of an intercepting filter?**

The functions of an intercepting filter are as follows:

* It intercepts the request from a client before it reaches the servlet and modifies the request if required.
* It intercepts the response from the servlet back to the client and modifies the request if required.
* There can be many filters forming a chain, in which case the output of one filter becomes an input to the next filter. Hence, various modifications can be performed on a single request and response.



**45. What are the functions of the Servlet container?**

The functions of the Servlet container are as follows:

* **Lifecycle management**: It manages the life and death of a servlet, such as class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.
* **Communication support**: It handles the communication between the servlet and the Web server.
* **Multithreading support**: It automatically creates a new thread for every servlet request received. When the Servlet service() method completes, the thread dies.
* **Declarative security** : It manages the security inside the XML deployment descriptor file.
* **JSP support** : The container is responsible for converting JSPs to servlets and for maintaining them.

=====================================================================

### What is different between web server and application server?

A web server responsibility is to handler HTTP requests from client browsers and respond with HTML response. A web server understands HTTP language and runs on HTTP protocol.  
Apache Web Server is kind of a web server and then we have specific containers that can execute servlets and JSPs known as servlet container, for example Tomcat.  
Application Servers provide additional features such as Enterprise JavaBeans support, JMS Messaging support, Transaction Management etc. So we can say that Application server is a web server with additional functionalities to help developers with enterprise applications.

1. Which HTTP method is non-idempotent?

A HTTP method is said to be idempotent if it returns the same result every time. HTTP methods GET, PUT, DELETE, HEAD, and OPTIONS are idempotent method and we should implement our application to make sure these methods always return same result. HTTP method POST is non-idempotent method and we should use post method when implementing something that changes with every request.

For example, to access an HTML page or image, we should use GET because it will always return the same object but if we have to save customer information to database, we should use POST method. Idempotent methods are also known as safe methods and we don’t care about the repetitive request from the client for safe methods.

1. What is the difference between GET and POST method?
   * GET is a safe method (idempotent) where POST is non-idempotent method.
   * We can send limited data with GET method and it’s sent in the header request URL whereas we can send large amount of data with POST because it’s part of the body.
   * GET method is not secure because data is exposed in the URL and we can easily bookmark it and send similar request again, POST is secure because data is sent in request body and we can’t bookmark it.
   * GET is the default HTTP method whereas we need to specify method as POST to send request with POST method.
   * Hyperlinks in a page uses GET method.
2. What is MIME Type?

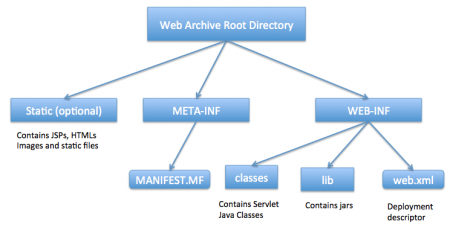
The “Content-Type” response header is known as MIME Type. Server sends MIME type to client to let them know the kind of data it’s sending. It helps client in rendering the data for user. Some of the mostly used mime types are text/html, text/xml, application/xml etc.

We can use ServletContext getMimeType() method to get the correct MIME type of the file and use it to set the response content type. It’s very useful in downloading file through servlet from server.

1. What is a web application and what is it’s directory structure?

Web Applications are modules that run on server to provide both static and dynamic content to the client browser. Apache web server supports PHP and we can create web application using PHP. Java provides web application support through Servlets and JSPs that can run in a servlet container and provide dynamic content to client browser.

Java Web Applications are packaged as Web Archive (WAR) and it has a defined structure like below image.

[](http://1988780851.rsc.cdn77.org/wp-content/uploads/2013/08/WAR-directory-structure.png)

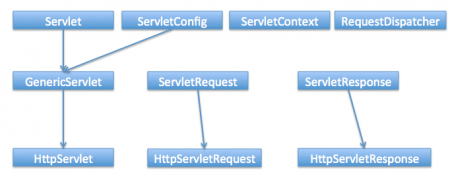
Read more about web applications at [**Java Web Application**](http://www.journaldev.com/1854/java-web-application-tutorial-for-beginners).

1. What is a servlet?

Java Servlet is server side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.

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

Most of the times, web applications are accessed using HTTP protocol and thats why we mostly extend HttpServlet class. Servlet API hierarchy is shown in below image.

[](http://1988780851.rsc.cdn77.org/wp-content/uploads/2013/08/Servlet-Hierarchy.png)

Read more at [**Servlet Tutorial**](http://www.journaldev.com/1877/java-servlet-tutorial-with-examples-for-beginners).

1. What are the advantages of Servlet over 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 and platform and system independent, the web application developed with Servlet can be run on any standard web container such as Tomcat, JBoss, Glassfish servers and on operating systems such as Windows, Linux, Unix, Solaris, Mac etc.
  + Servlets are robust because container takes care of life cycle of servlet and we don’t need to worry about memory leaks, security, garbage collection etc.
  + Servlets are maintainable and learning curve is small because all we need to take care is business logic for our application.

1. What are common tasks performed by Servlet Container?

Servlet containers are also known as web container, for example Tomcat. Some of the important tasks of servlet container are:

* + **Communication Support**: Servlet Container provides easy way of communication between web client (Browsers) and the servlets and JSPs. Because of container, we don’t need to build a server socket to listen for any request from web client, parse the request and generate response. All these important and complex tasks are done by container and all we need to focus is on business logic for the applications.
  + **Lifecycle and Resource Management**: Servlet Container takes care of managing the life cycle of servlet. From the loading of servlets into memory, initializing servlets, invoking servlet methods and to destroy 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 provide them request and response objects to process. So servlets are not initialized for each request and saves time and memory.
  + **JSP Support**: JSPs doesn’t look like normal java classes but every JSP in the application is compiled by container and converted to Servlet and then container manages them like other servlets.
  + **Miscellaneous Task**: Servlet container manages the resource pool, perform memory optimizations, execute garbage collector, provides security configurations, support for multiple applications, hot deployment and several other tasks behind the scene that makes a developer life easier.

1. What is ServletConfig object?

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.

1. What is ServletContext object?

javax.servlet.ServletContext interface provides access to web application parameters 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 containers 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.

ServletContext is enhanced in Servlet Specs 3 to introduce methods through which we can programmatically add Listeners and Filters and Servlet to the application. It also provides some utility methods such as *getMimeType()*, *getResourceAsStream()* etc.

1. What is difference between ServletConfig and ServletContext?

Some of the differences between ServletConfig and ServletContext are:

* + ServletConfig is a unique object per servlet whereas ServletContext is a unique object for complete application.
  + ServletConfig is used to provide init parameters to the servlet whereas ServletContext is used to provide application level init parameters that all other servlets can use.
  + We can’t set attributes in ServletConfig object whereas we can set attributes in ServletContext that other servlets can use in their implementation.

1. What is Request Dispatcher?

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

1. What is difference between PrintWriter and ServletOutputStream?

PrintWriter is a character-stream class whereas ServletOutputStream is a byte-stream class. We can use PrintWriter to write character based information such as character array and String to the response whereas we can use ServletOutputStream to write byte array data to the response.

We can use ServletResponse getWriter() to get the PrintWriter instance whereas we can use ServletResponse getOutputStream() method to get the ServletOutputStream object reference.

You can read more about IO in java at [**Java IO Tutorial**](http://www.journaldev.com/942/java-io-tutorial).

1. Can we get PrintWriter and ServletOutputStream both in a servlet?

We can’t get instances of both PrintWriter and ServletOutputStream in a single servlet method, if we invoke both the methods; getWriter() and getOutputStream() on response; we will getjava.lang.IllegalStateException at runtime with message as other method has already been called for this response.

1. How can we create deadlock situation in servlet?

We can create deadlock in servlet by making a loop of method invocation, just call doPost() method from doGet() method and doGet() method to doPost() method to create deadlock situation in servlet.

Read more about deadlock in multithreading at [**Java Deadlock Example**](http://www.journaldev.com/1058/java-deadlock-example-and-how-to-analyze-deadlock-situation).

1. What is the use of servlet wrapper classes?

Servlet HTTP API provides two wrapper classes – HttpServletRequestWrapper andHttpServletResponseWrapper. These wrapper classes are provided to help developers with custom implementation of servlet request and response types. We can extend these classes and override only specific methods we need to implement for custom request and response objects. These classes are not used in normal servlet programming.

1. What is SingleThreadModel interface?

SingleThreadModel interface was provided for thread safety and it guarantees that no two threads will execute concurrently in the servlet’s service method. However SingleThreadModel does not solve all thread safety issues. For example, session attributes and static variables can still be accessed by multiple requests on multiple threads at the same time, even when SingleThreadModel servlets are used. Also it takes out all the benefits of multithreading support of servlets, thats why this interface is Deprecated in Servlet 2.4.

1. Do we need to override service() method?

When servlet container receives client request, it invokes the service() method which in turn invokes the doGet(), doPost() methods based on the HTTP method of request. I don’t see any use case where we would like to override service() method. The whole purpose of service() method is to forward to request to corresponding HTTP method implementations. If we have to do some pre-processing of request, we can always use servlet filters and listeners.

1. Is it good idea to create servlet constructor?

We can define a constructor for servlet but I don’t think its of any use because we won’t be having access to the ServletConfig object until unless servlet is initialized by container. Ideally if we have to initialize any resource for servlet, we should override init() method where we can access servlet init parameters using ServletConfig object.

1. What is difference between GenericServlet and HttpServlet?

GenericServlet is protocol independent implementation of Servlet interface whereas HttpServlet is HTTP protocol specific implementation. Most of the times we use servlet for creating web application and that’s why we extend HttpServlet class. HttpServlet class extends GenericServlet and also provide some other methods specific to HTTP protocol.

[sociallocker id=”2713″]

1. What is the inter-servlet communication?

When we want to invoke another servlet from a servlet service methods, we use inter-servlet communication mechanisms. We can invoke another servlet using RequestDispatcher forward() and include() methods and provide additional attributes in request for other servlet use.

1. Are Servlets Thread Safe? How to achieve thread safety in servlets?

HttpServlet init() method and destroy() method are called only once in servlet life cycle, so we don’t need to worry about their synchronization. But service methods such as doGet() or doPost() are getting called in every client request and since servlet uses multithreading, we should provide thread safety in these methods.

If there are any local variables in service methods, we don’t need to worry about their thread safety because they are specific to each thread but if we have a shared resource then we can use synchronization to achieve thread safety in servlets when working with shared resources.

The thread safety mechanisms are similar to thread safety in standalone java application, read more about them at [**Thread Safety in Java**](http://www.journaldev.com/1061/java-synchronization-and-thread-safety-tutorial-with-examples).

### What is servlet attributes and their scope?

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.

### How do we call one servlet from another servlet?

We can use RequestDispatcher forward() method to forward the processing of a request to another servlet. If we want to include the another servlet output to the response, we can use RequestDispatcher include() method.

### How can we invoke another servlet in a different application?

We can’t use RequestDispatcher to invoke servlet from another application because it’s specific for the application. If we have to forward the request to a resource in another application, we can use ServletResponse sendRedirect() method and provide complete URL of another servlet. This sends the response to client with response code as 302 to forward the request to another URL. If we have to send some data also, we can use cookies that will be part of the servlet response and sent in the request to another servlet.

### What is difference between ServletResponse sendRedirect() and RequestDispatcher forward() method?

* 1. RequestDispatcher forward() is used to forward the same request to another resource whereas ServletResponse sendRedirect() is a two step process. In sendRedirect(), web application returns the response to client with status code 302 (redirect) with URL to send the request. The request sent is a completely new request.
  2. forward() is handled internally by the container whereas sednRedirect() is handled by browser.
  3. We should use forward() when accessing resources in the same application because it’s faster than sendRedirect() method that required an extra network call.
  4. In forward() browser is unaware of the actual processing resource and the URL in address bar remains same whereas in sendRedirect() URL in address bar change to the forwarded resource.
  5. forward() can’t be used to invoke a servlet in another context, we can only use sendRedirect() in this case.

### Why HttpServlet class is declared abstract?

HttpServlet class provide HTTP protocol implementation of servlet but it’s left abstract because there is no implementation logic in service methods such as doGet() and doPost() and we should override at least one of the service methods. That’s why there is no point in having an instance of HttpServlet and is declared abstract class.

Read more about [**abstract class**](http://www.journaldev.com/1582/abstract-class-in-java-with-example).

### What are the phases of servlet life cycle?

We know that Servlet Container manages the life cycle of Servlet, there are four phases of servlet life cycle.

* 1. Servlet Class Loading – When container receives request for a servlet, it first loads the class into memory and calls it’s default no-args constructor.
  2. Servlet Class Initialization – Once the servlet class is loaded, container initializes the ServletContext object for the servlet and then invoke it’s init method by passing servlet config object. This is the place where a servlet class transforms from normal class to servlet.
  3. Request Handling – Once servlet is initialized, its ready to handle the client requests. For every client request, servlet container spawns a new thread and invokes the service() method by passing the request and response object reference.
  4. Removal from Service – When container stops or we stop the application, servlet container destroys the servlet class by invoking it’s destroy() method.

### What are life cycle methods of a servlet?

Servlet Life Cycle consists of three methods:

* 1. public void init(ServletConfig config) – This method is used by container to initialize the servlet, this method is invoked only once in the lifecycle of servlet.
  2. public void service(ServletRequest request, ServletResponse response) – This method is called once for every request, container can’t invoke service() method until unless init() method is executed.
  3. public void destroy() – This method is invoked once when servlet is unloaded from memory.

### why we should override only no-agrs init() method.

If we have to initialize some resource before we want our servlet to process client requests, we should override init() method. If we override init(ServletConfig config) method, then the first statement should be super(config) to make sure superclass init(ServletConfig config) method is invoked first. That’s why GenericServlet provides another helper init() method without argument that get’s called at the end of init(ServletConfig config) method. We should always utilize this method for overriding init() method to avoid any issues as we may forget to add super() call in overriding init method with ServletConfig argument.

### What is URL Encoding?

URL Encoding is the process of converting data into CGI form so that it can travel across the network without any issues. URL Encoding strip the white spaces and replace special characters with escape characters. We can use java.net.URLEncoder.encode(String str, String unicode) to encode a String. URL Decoding is the reverse process of encoding and we can use java.net.URLDecoder.decode(String str, String unicode) to decode the encoded string. For example “Pankaj’s Data” is encoded to “Pankaj%27s+Data”.

### What are different methods of session management in servlets?

Session is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, 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.

Some of the common ways of session management in servlets are:

* 1. User Authentication
  2. HTML Hidden Field
  3. Cookies
  4. URL Rewriting
  5. Session Management API

Read more about these session management approaches in detail at [**Servlet Session Management Tutorial**](http://www.journaldev.com/1907/java-servlet-session-management-tutorial-with-examples-of-cookies-httpsession-and-url-rewriting).

### What is URL Rewriting?

We can use HttpSession for session management in servlets but it works with Cookies and we can disable the cookie in client browser. Servlet API provides support for URL rewriting that we can use to manage session in this case.

The best part is that from coding point of view, it’s very easy to use and involves one step – encoding the URL. Another good thing with Servlet URL Encoding is that it’s a fallback approach and it kicks in only if browser cookies are disabled.

We can encode URL with HttpServletResponse encodeURL() method and if we have to redirect the request to another resource and we want to provide session information, we can use encodeRedirectURL() method.

Read More at [**Servlet URL Rewriting**](http://www.journaldev.com/1907/java-servlet-session-management-tutorial-with-examples-of-cookies-httpsession-and-url-rewriting#servlet-url-rewriting).

### How does Cookies work in Servlets?

Cookies are used a lot in web client-server communication, it’s not something specific to java. Cookies are text data sent by server to the client and it gets saved at the client local machine.

Servlet API provides cookies support through javax.servlet.http.Cookie class that implements Serializable and Cloneable interfaces.

HttpServletRequest getCookies() method is provided to get the array of Cookies from request, since there is no point of adding Cookie to request, there are no methods to set or add cookie to request.

Similarly HttpServletResponse addCookie(Cookie c) method is provided to attach cookie in response header, there are no getter methods for cookie.

Read more at [**Cookies in Servlets**](http://www.journaldev.com/1956/servlet-cookie-example-tutorial).

### How to notify an object in session when session is invalidated or timed-out?

If we have to make sure an object gets notified when session is destroyed, the object should implement javax.servlet.http.HttpSessionBindingListener interface. This interface defines two callback methods – valueBound() and valueUnbound() that we can define to implement processing logic when the object is added as attribute to the session and when session is destroyed.

Recommended reading [**Servlet Listener**](http://www.journaldev.com/1945/servlet-listener-example-servletcontextlistener-httpsessionlistener-and-servletrequestlistener).

### What is the difference between encodeRedirectUrl and encodeURL?

HttpServletResponse provide method to encode URL in HTML hyperlinks so that the special characters and white spaces are escaped and append session id to the URL. It behaves similar to URLEncoder encode method with additional process to append jsessionid parameter at the end of the URL.

However HttpServletResponse encodeRedirectUrl() method is used specially for encode the redirect URL in response.

So when we are providing URL rewriting support, for hyperlinks in HTML response, we should use encodeURL() method whereas for redirect URL we should use encodeRedirectUrl() method.

### Why do we have servlet filters?

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.

Some common tasks that we can do with filters are:

* 1. Logging request parameters to log files.
  2. Authentication and autherization of request for resources.
  3. Formatting of request body or header before sending it to servlet.
  4. Compressing the response data sent to the client.
  5. Alter response by adding some cookies, header information etc.

Read more about filters at [**Servlet Filter**](http://www.journaldev.com/1933/java-servlet-filter-example-tutorial).

### What is the effective way to make sure all the servlets are accessible only when user has a valid session?

We know that servlet filters can be used to intercept request between servlet container and servlet, we can utilize it to create authentication filter and check if request contains a valid session or not.

Check out Authentication Filter example at [**Servlet Filter Example**](http://www.journaldev.com/1933/java-servlet-filter-example-tutorial#servlet-filter-example).

### Why do we have servlet listeners?

We know that using ServletContext, we can create an attribute with application scope that all other servlets can access but we can initialize ServletContext init parameters as String only in deployment descriptor (web.xml). What if our application is database oriented and we want to set an attribute in ServletContext for Database Connection.

If you application has a single entry point (user login), then you can do it in the first servlet request but if we have multiple entry points then doing it everywhere will result in a lot of code redundancy. Also if database is down or not configured properly, we won’t know until first client request comes to server. To handle these scenario, servlet API provides Listener interfaces that we can implement and configure to listen to an event and do certain operations.

Read more about different types of listeners and example at [**Servlet Listener**](http://www.journaldev.com/1945/servlet-listener-example-servletcontextlistener-httpsessionlistener-and-servletrequestlistener).

### How to handle exceptions thrown by application with another servlet?

If you notice, doGet() and doPost() methods throw ServletException and IOException. Since browser understand only HTML, when our application throw exception, servlet container processes the exception and generate a HTML response. Same goes with other error codes like 404, 403 etc.

Servlet API provides support for custom Exception and Error Handler servlets that we can configure in deployment descriptor, the whole purpose of these servlets are to handle the Exception or Error raised by application and send HTML response that is useful for the user. We can provide link to application home page or some details to let user know what went wrong.

We can configure them in web.xml like below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <error-page>      <error-code>404</error-code>      <location>/AppExceptionHandler</location>  </error-page>    <error-page>      <exception-type>javax.servlet.ServletException</exception-type>      <location>/AppExceptionHandler</location>  </error-page> |

Read more at [**Servlet Exception Handling**](http://www.journaldev.com/1973/servlet-exception-and-error-handling-example-tutorial).

### What is a deployment descriptor?

Deployment descriptor is a configuration file for the web application and it’s name is web.xml and it resides in WEB-INF directory. Servlet container use this file to configure web application servlets, servlet config params, context init params, filters, listeners, welcome pages and error handlers.

With servlet 3.0 annotations, we can remove a lot of clutter from web.xml by configuring servlets, filters and listeners using annotations.

### How to make sure a servlet is loaded at the application startup?

Usually servlet container loads a servlet on the first client request but sometimes when the servlet is heavy and takes time to loads, we might want to load it on application startup. We can use load-on-startup element with servlet configuration in web.xml file or use WebServlet annotation loadOnStartup variable to tell container to load the servlet on system startup.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <servlet>      <servlet-name>foo</servlet-name>      <servlet-class>com.foo.servlets.Foo</servlet-class>      <load-on-startup>5</load-on-startup>  </servlet> |

The load-on-startup value should be int, if it’s negative integer then servlet container will load the servlet based on client requests and requirement but if it’s 0 or positive, then container will load it on application startup.

If there are multiple servlets with load-on-startup value such as 0,1,2,3 then lower integer value servlet will be loaded first.

### How to get the actual path of servlet in server?

We can use following code snippet to get the actual path of the servlet in file system.

getServletContext().getRealPath(request.getServletPath())

### How to get the server information in a servlet?

We can use below code snippet to get the servlet information in a servlet through servlet context object.

getServletContext().getServerInfo()

### Write a servlet to upload file on server.

File Upload and Download and common tasks in a java web application. Unfortunately Servlet API doesn’t provide easy methods to upload file on server, so we can use Apache FileUpload jar to make our life easier.

Please read [**File Upload Servlet**](http://www.journaldev.com/1964/servlet-upload-file-and-download-file-example) post that provide all the necessary details with example program to upload and download file using servlets.

### How do we go with database connection and log4j integration in servlet?

If you work with database connection a lot in your web application, its best to initialize it in a servlet context listener and set it as a context attribute for other servlets to use.

Integrating Log4j is also very easy in web applications, all we need is a log4j configuration XML or property file and then configure it in a servlet context listener.

For complete example, please look into [**Servlet Database and Log4j Example**](http://www.journaldev.com/1997/servlet-example-in-java-with-database-connection-and-log4j-integration).

### How to get the IP address of client in servlet?

We can use request.getRemoteAddr() to get the client IP address in servlet.

### What are important features of Servlet 3?

Servlet Specs 3.0 was a major release and some of the important features are:

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

Servlet 3 introduced use of java annotations to define a servlet, filter and listener servlets and init parameters. Some of the important Servlet API annotations are WebServlet, WebInitParam, WebFilter and WebListener. Read more about them at [Servlet 3 annotations](http://www.journaldev.com/1877/java-servlet-tutorial-with-examples-for-beginners#servlet-3-annotations).

* 1. **Web Fragments**: Prior to servlet specs 3.0, all the web application configurations are required to be present in the web.xml that makes it cluttered with lot of elements and chances of error increases. So servlet 3 specs introduced web fragments where we can have multiple modules in a single web application, all these modules should have web-fragment.xml file in META-INF directory. We can include all the elements of web.xml inside the web-fragment.xml too. This helps us in dividing our web application into separate modules that are included as JAR file in the web application lib directory.
  2. **Adding Web Components dynamically**: We can use ServletContext object to add servlets, filters and listeners programmatically. This helps us in building dynamic system where we are loading a component only if we need it. These methods are addServlet(), addFilter() and addListener() defined in the servlet context object.
  3. **Asynchronous Processing**: Asynchronous support was added to delegate the request processing to another thread rather than keeping the servlet thread busy. It can increase the throughput performance of the application. This is an advance topic and I recommend to read [**Async Servlet**](http://www.journaldev.com/2008/async-servlet-feature-of-servlet-3)tutorial.

### What are different ways for servlet authentication?

Servlet Container provides different ways of login based servlet authentication:

* 1. **HTTP Basic Authentication**
  2. **HTTP Digest Authentication**
  3. **HTTPS Authentication**
  4. **Form Based Login**: A standard HTML form for authentication, advantage is that we can change the login page layout as our application requirements rather than using HTTP built-in login mechanisms.

### How can we achieve transport layer security for our web application?

We can configure our servlet container to use SSL for message communication over the network. To configure SSL on Tomcat, we need a digital certificate that can be created using Java keytool for development environment. For production environment, you should get the digital certificate from SSL certificate providers, for example, Verisign or Entrust.

Read more at [**Configure SSL on Tomcat**](http://www.journaldev.com/160/steps-to-configure-ssl-on-tomcat-and-setup-auto-redirect-from-http-to-https) article.

[/sociallocker]

That’s all for the servlet interview questions and answers. I will keep on adding more servlet based interview questions to the list in future, so don’t forget to bookmark it for future reference.

s