|  |
| --- |
| What is a Servlet Container and how is it different from a Web Server?    Broadly speaking, a servlet container restricts itself more or less to the implementation of the J2EE Servlet specification. Also, it's focus is on the runtime environment and not so much on providing additional tools.  In contrast, a full fledged application server implements the whole J2EE stack; plus it comes with all the enterprisey tools and integration possibilities. An application server usually has advanced administration interfaces, it supports clustering and other features used mostly in high-end systems development.  For a beginner, it's probably better to stay with a simple servlet container, since the learning curve there is much less steep. |
|  |  |
|  | What is Servlet interface. What is GenericServlet. What is HttpServlet?  Defines methods that all servlets must implement.  A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.  To implement this interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.  This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:   1. The servlet is constructed, then initialized with the init method. 2. Any calls from clients to the service method are handled. 3. The servlet is taken out of service, then destroyed with the destroy method, then garbage collected and finalized.   In addition to the life-cycle methods, this interface provides the getServletConfig method, which the servlet can use to get any startup information, and the getServletInfo method, which allows the servlet to return basic information about itself, such as author, version, and copyright.  Defines a generic, protocol-independent servlet. To write an HTTP servlet for use on the Web, extend [HttpServlet](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServlet.html" \o "class in javax.servlet.http) instead.  GenericServlet implements the Servlet and ServletConfig interfaces. GenericServlet may be directly extended by a servlet, although it's more common to extend a protocol-specific subclass such as HttpServlet.  GenericServlet makes writing servlets easier. It provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface. GenericServlet also implements the logmethod, declared in the ServletContext interface.  public abstract class **HttpServlet**  extends [GenericServlet](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/GenericServlet.html" \o "class in javax.servlet)  implements java.io.Serializable  Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:   * doGet, if the servlet supports HTTP GET requests * doPost, for HTTP POST requests * doPut, for HTTP PUT requests * doDelete, for HTTP DELETE requests * init and destroy, to manage resources that are held for the life of the servlet * getServletInfo, which the servlet uses to provide information about itself   There's almost no reason to override the service method. service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the do*XXX* methods listed above).  Likewise, there's almost no reason to override the doOptions and doTrace methods.  Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections. See the [Java Tutorial on Multithreaded Programming](http://java.sun.com/Series/Tutorial/java/threads/multithreaded.html) for more information on handling multiple threads in a Java program. |
|  |  |
|  | What is ServletContext ?  y the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.  **Easy to maintain** if   1. The object of ServletContext provides an interface between the container and servlet. 2. The ServletContext object can be used to get configuration information from the web.xml file. 3. The ServletContext object can be used to set, get or remove attribute from the web.xml file. 4. The ServletContext object can be used to provide inter-application communication. |
|  |  |
|  | What is the life cycle of a Servlet?  1) Servlet class is loaded  The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.  2) Servlet instance is created  The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.  3) init method is invoked   |  | | --- | | The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below: |  1. **public** **void** init(ServletConfig config) **throws** ServletException   4) service method is invoked  The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:   1. **public** **void** service(ServletRequest request, ServletResponse response) 2. **throws** ServletException, IOException   5) destroy method is invoked  The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:   1. **public** **void** destroy() |
|  |  |
|  | What is the role of web.xml file?  Generally speaking, this is the configuration file of web applications in java. It instructs the servlet container (tomcat for ex.) which classes to load, what parameters to set in the context, and how to intercept requests coming from browsers.  There you specify:   * what servlets (and filters) you want to use and what URLs you want to map them to * listeners - classes that are notified when some events happen (context starts, session created, etc) * configuration parameters (context-params) * error pages, welcome files * security constriants |
|  |  |
|  | What is load-on-startup in web.xml file?  The element load-on-startup indicates that this servlet should be loaded (instantiated and have its init() called) on the startup of the Web application. The element content of this element must be an integer indicating the order in which the servlet should be loaded. If the value is a negative integer, or the element is not present, the container is free to load the servlet whenever it chooses. If the value is a positive integer or 0, the container must load and initialize the servlet as the application is deployed. The container must guarantee that servlets marked with lower integers are loaded before servlets marked with higher integers. The container may choose the order of loading of servlets with the same load-on-startup value |
|  |  |
|  | What is the role of ServletException and IOException used in the signature of |
|  | service/doGet/doPost method?  doGet() and doPost() are HTTP requests handled by servlet classes.  In doGet(), the parameters are appended to the URL and sent along with header information. This does not happen in case of doPost(). In doPost(), the parameters are sent separately. Since most of the web servers support only a limited amount of information to be attached to the headers, the size of this header should not exceed 1024 bytes. doPost() does not have this constraint. Usually programmers find it difficult to choose between doGet() and doPost().  doGet() shall be used when small amount of data and insensitive data like a query has to be sent as a request. doPost() shall be used when comparatively large amount of sensitive data has to be sent. Examples are sending data after filling up a form or sending login id and password |
|  |  |
|  | Why is Servlet better than conventional CGI?  Servlets are run in one process (HTTP server with additional features, which called Servlet Container) and they exist as long as that process exists.  CGI means every time there's client request, HTTP server creates new instance of process to serve this request. This is performance killer. Additionally, since there's new process per each request, it means CGI can't aggregate data from several requests in memory, as Servlets can, and must resort to external persistent storage (file or DB). This is performance killer as well. |
|  |  |
|  | What is SingleThreadModel interface?  public abstract interface **SingleThreadModel**  Ensures that servlets handle only one request at a time. This interface has no methods.  If a servlet implements this interface, you are *guaranteed* that no two threads will execute concurrently in the servlet's service method. The servlet container can make this guarantee by synchronizing access to a single instance of the servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet.  This interface does not prevent synchronization problems that result from servlets accessing shared resources such as static class variables or classes outside the scope of the servlet. |