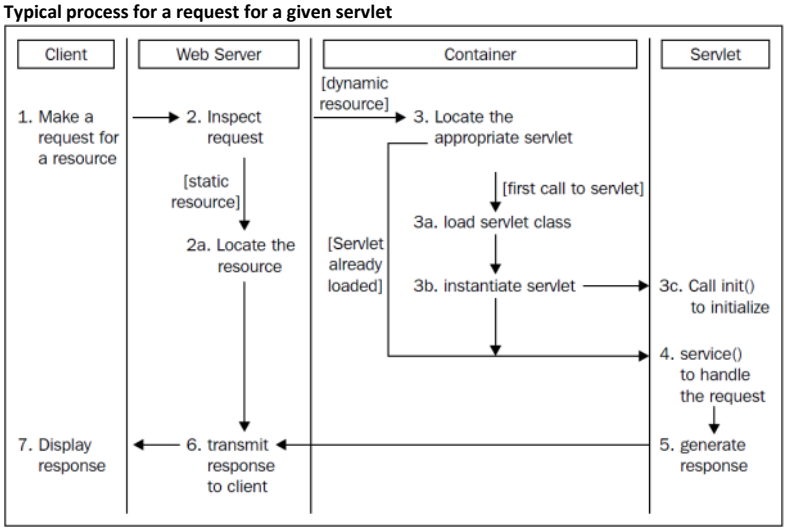
Tomcat Component Structure

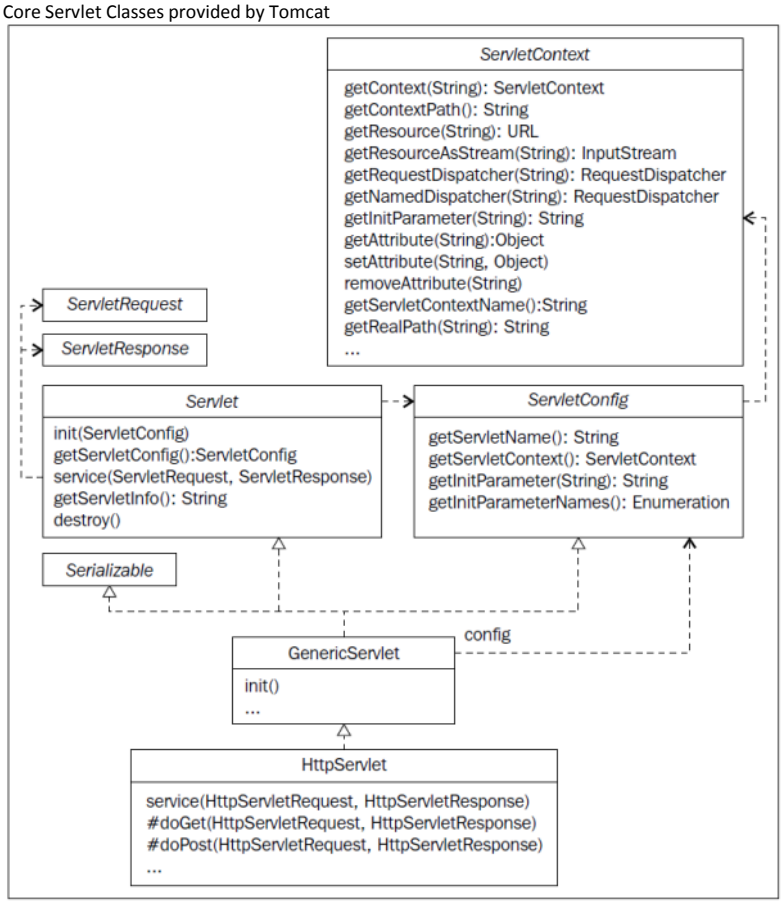
Monday, March 31, 2014 09:14

1. **What is Tomcat**
   Tomcat is classified as a servlet container, that is, an environment within which servlets can live and prosper. As a container, it provides a lot of administrative support to servlets, allowing programmers to focus on the core application logic that is to be implemented, without having to bother about low level specifics such as session management and class loading.

2. **Typical process for a request for a given servlet**

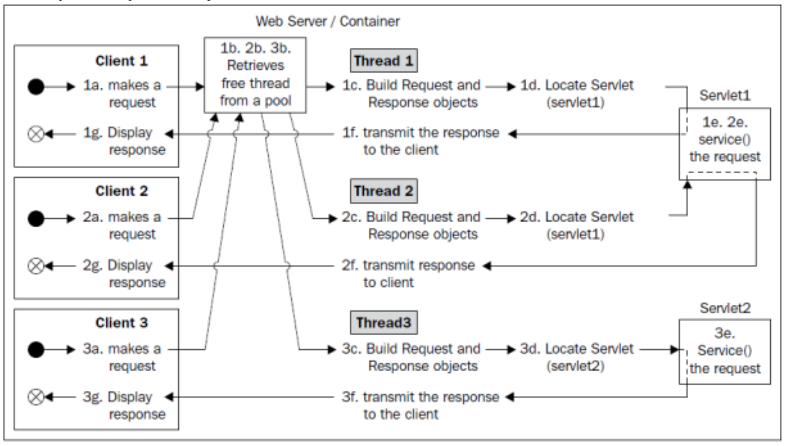


3. Core Servlet Classes provided by Tomcat



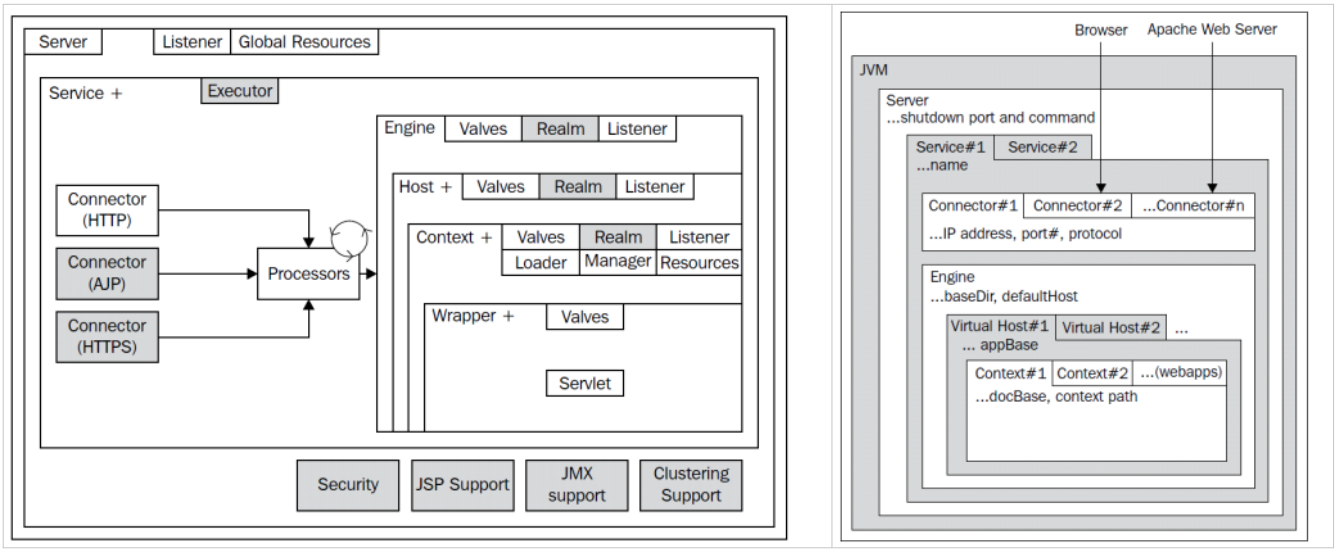
   a. **Servlets loads process**
      i. After a servlet is first loaded and instantiated, its init(ServletConfig) method is called, so as to provide it with its initialization parameters, as well as to allow it to access its runtime context through this ServletConfig parameter. Only after the init() method completes, will the servlet be asked to service any incoming requests. A servlet will typically use this opportunity to initialize any resources that it needs for its functioning.
      ii. **The service() method is called once per incoming request for this servlet.**
      The service() method is implemented by javax.servlet.http.HttpServlet which inspects the incoming HTTP method and invokes the appropriate method for that request type. (Most application developers write servlets that override only the doGet() and doPost() methods.**)**
      iii. The destroy() method is called by the container just before the servlet is taken out of service and gives the servlet a chance to release any resources that it has acquired.
   b. A javax.servlet.ServletContext instance represents the web application to which this servlet belongs. A servlet context repre sents the collection of web components (servlets, filters, listeners, and JSP files), utility classes, library JARs, static content (HTML, CSS, JavaScript, and so o n), and other resources that are made available to clients under a specific context path within the servlet container. **There is one context per web application that is deployed into the container.**
   c. **Multithreading in Servlets**
      Servlets are inherently multithreaded. In other words, **the servlet container will create only one instance per servlet declaration within the deployment descriptor** .

(Note that this is per declaration and not per servlet class.)
As a result, **at any given time, multiple processor threads within the container may be executing within a servlet instance's service() met hod. It is the servlet developer's responsibility to ensure that this invocation is thread-safe.**



This container has two servlet instances, servlet1 of the class Servlet1 and servlet2 of the class Servlet2. Let us take a snapshot of this system while all three threads are executing within the appropriate service() methods. At this particular instant in time, both processor thread Thread1 and processor thread Thread2 are in the process of executing servlet1.service(), whereas processor thread Thread3 is in the process of executing servlet2.service(). Any instance variables of the servlet1 instance are now being accessed concurrently by both Thread1 and Thread2. Therefore, they must be protected against corruptio n. Likewise, any variables in context or session scope (which we'll see later) may be accessed simultaneously by all three threads, and so must also be protected a gainst concurrent access.

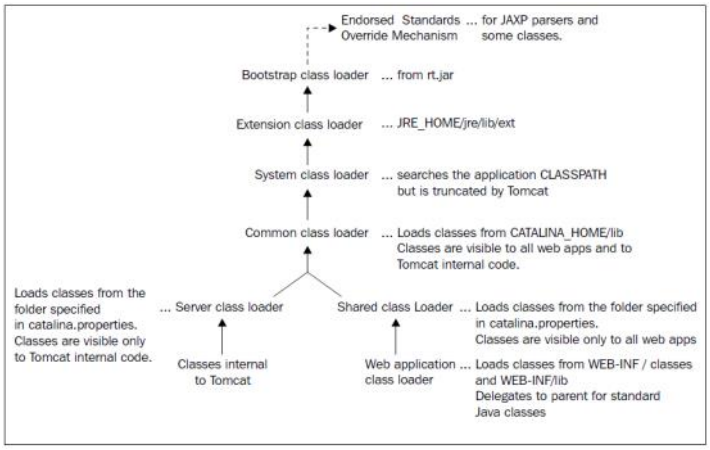
4. **Bird's eye view of the responsibilities of a servlet container**
The primary responsibility of a container is to process an incoming request and to generate a corresponding response that is then returned to the client.
Note that the '+' sign in the above image indicates that there can be more than one instance of that component.



a. **the Server and Service components are special containers, designated as Top Level Elements** as they represent aspects of the running Tomcat instance. When Tomcat is started, the Java Virtual Machine (JVM) instance in which it runs will contain a singleton Server top level element, which represents the entire Tomcat server. A Server will usually contain just one Service object, which is a structural element that combines one or more Connectors
b. **The Engine, Host, and Context components are officially termed Containers**, and refer to components that process incoming requests and generate an appropriate outgoing response. The Engine represents the core request processing code within Tomcat and supports the definition of multiple Virtual Hosts wi thin it. A virtual host allows a single running Tomcat engine to make it seem to the outside world that there are multiple separate domains being hosted on a single machine. Each virtual host can, in turn, support multiple web applications known as Contexts that are deployed to it. A context is represented using the web application format specified by the servlet specification, either as a single compressed WAR (Web Application Archive) file or as an uncompresse d directory. In addition, a context is configured using a web.xml file, as defined by the servlet specification. A context can, in turn, contain multiple servlets that are deployed into it, each of which is wrapped in a Wrapper component.
c. **Nested Components can be thought of as sub-elements that can be nested inside either Top Level Elements or other Containers to configure how they function.** Examples of nested components include the Valve, which represents a reusable unit of work; the Pipeline, which represents a c hain of Valves strung together; and a Realm which helps set up container-managed security for a particular container.
d. **A final major component, which falls into its own category, is the Connector.** It represents the connection end point that an external client (such as a web browser) can use to connect to the Tomcat container.
e. All the configuration is in the "server.xml" files, which is read at the tomcat startup. **Note, this file is read only once, and edits to it will not be picked up until Tomcat is restarted.**

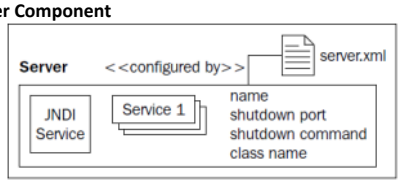
5. **Deployment descriptors**
a. **the <servlet> element** maps a fully qualified servlet class to a logical name that will be used to refer to that servlet. That logical name is then mapped using a servlet-mapping element to a given URL pattern.
b. **Its <filter> element** associates the fully qualified class name of a servlet filter with a logical name. This logical name can then be mapped to a given URL pattern or to a servlet's logical name using the filter-mapping element.
c. A **<context-param> element** lets you specify global servlet context initialization parameters, which are available to all servlets that run within this application context.
d. **The <listener> element** defines the fully qualified name of a class that is to be registered as a web application event listener.
e. **Another key element is <session-config>**, which lets you set the default session timeout interval for all sessions that are created in this web application.
f. **A final major component, which falls into its own category, is the Connector.** It represents the connection end point that an external client (such as a web browser) can use to connect to the Tomcat container.

6. **Tomcat Class Loader**



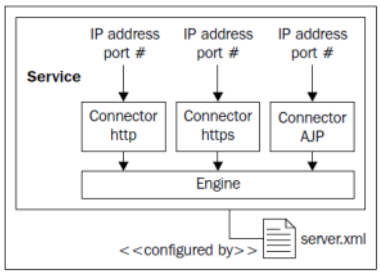
a. During startup, Tomcat first neutralizes the System class loader by clearing out the CLASSPATH and resetting it to point to CATALINA_HOME/bin/bootstrap.jar (for the classes required for Tomcat to start up), tomcat-juli.jar (for logging), and tools.jar (for the JSP compiler). This leaves the System class loader useful only for loading a minimal set of Tomcat-specific classes.
b. Tomcat also changes the endorsed directory to point to CATALINA_HOME/endorsed.
c. Below it, Tomcat establishes its own hierarchy of class loaders by appending the Server class loader, the Shared class loader , the Common class loader, and one web application class loader per deployed application.
d. Loading process:
   i. This class loader first delegates to the System class loader to allow the delegation hierarchy to locate any core Java classes.
   ii. If the requested class cannot be found, then the web application class loader attempts to locate the class within its own repositories.
   iii. If the class is still not found, it will delegate to the Common class loader, or to the Shared class loader if it is installed.
e. The Shared class loader and the Server class loader are not instantiated by default. You can enable them by editing the CATALINA_HOME/conf/catalina.properties file and adding the shared.loader and server.loader entries
f. The Common class loader monitors the contents of the CATALINA_HOME/lib folder, which contains commonly used JARs such as servlet-api.jar, jasper.jar, coyote.jar, and jsp-api.jar.

7. Server Component



a. Definition: **A Server represents the entire Tomcat instance and is a singleton within a Java Virtual Machine**, and is responsible for managing the life cycle of its contained services.
b. Base Class: *org.apache.catalina.core.StandardServer*
c. Default opens a server socket on port 8005 to listen a shutdown command, by default it is "SHUTDOWN".
d. It also provides an implementation of JNDI (Java Naming and Directory Interface), allowing you to register arbitrary objects (such as data sources) or environment variables, by name.

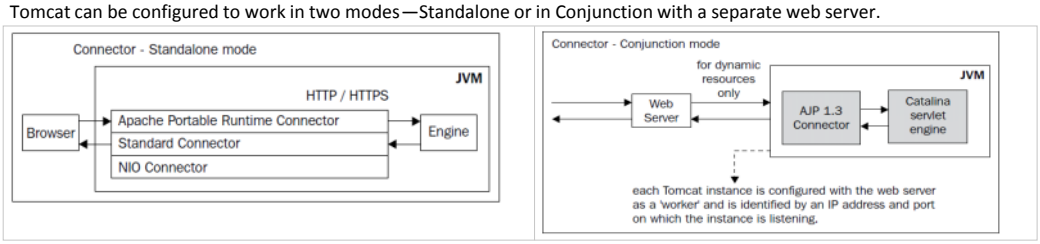
8. Service Component



a. Definition: **A Service represents the set of request processing components within Tomcat.**
b. It's rear reason to modify this element, and the default instance is usually sufficient.
c. An example use case for having multiple services, therefore, is when you want to partition your services (and their contained engines, hosts, and web applications) by IP address and/or port number.
d. The Service, therefore, is nothing more than a grouping construct.

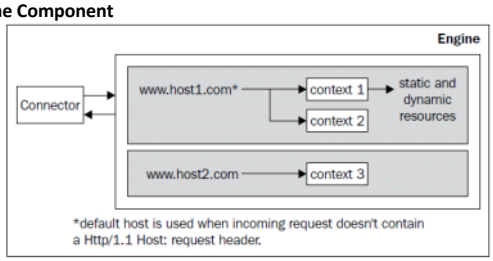
9. Connector Component

a. Definition: A Connector is a service endpoint on which a client connects to the Tomcat container. It serves to insulate the engine from the various communication protocols that are used by clients, such as HTTP, HTTPS, or the Apache JServ Protocol (AJP).
b. Tomcat can be configured to work in two modes—Standalone or in Conjunction with a separate web server.



c. The primary attribute of a Connector are the IP address and port
d. Another key attribute is the maximum number of request processing threads that can be created to concurrently handle incoming requests. Once all these threads are busy, any incoming request will be ignored until a thread becomes available.
e. By default, it listens on all the IP address (0.0.0.0).

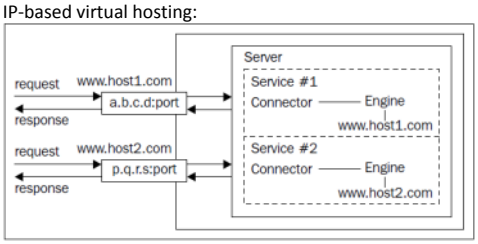
10. **Engine Component**



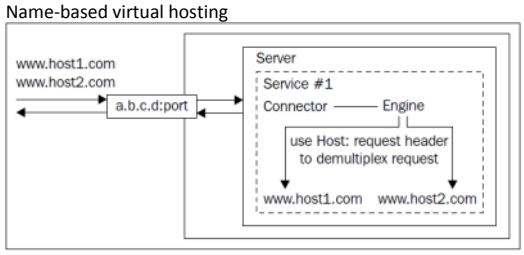
   a. Definition: An Engine represents a running instance of the Catalina servlet engine and comprises the heart of a servlet container's funct ion.
   b. **There can only be one engine within a given service**
   c. Its main function is to delegate the processing of the incoming request to the appropriate virtual host.
   d. If the engine has no virtual host with a name matching the one to which the request should be directed, it consults its defau ltHost attribute to determine the host that should be used.

11. **Virtual Host**
   a. Definition: A virtual host in Tomcat is represented by the Host component, which is a container for web applications, or, in Tomcat parla nce, contexts.
   b. Two key concepts come into play when working with virtual hosts—the host's domain name and its application base folder.
      i. Domain name: Each virtual host is identified by the domain name that you registered for use with this host. This is the value that you expect the client browser to send in the Host: request header. A host's name is required to be unique within its containing engine.
      ii. Application base folder: This folder is the location that contains the contexts that will be deployed to this host. This folder location can either be specified as an absolute path or as a path relative to CATALINA_BASE.
   c. Virtual host techniques

IP-based virtual hosting:



Name-based virtual hosting



12. **Context configuration**
   a. Definition: Tomcat lets you configure a Context by letting you extract the <Context> element from the server.xml file and move it into a separate file called a context fragment file. **Context fragments are monitored and reloaded by Tomcat at runtime.**
   b. **File path:**
      i. CATALINA_HOME/conf/context.xml
      ii. CATALINA_HOME/conf/<EngineName>/<HostName>/
      iii. <project_path>/META_INF/context.xml
      iv. Also can be configured using web.xml

13. **Executor**
   a. Definition: This is a new element, available only since 6.0.11. It allows you to configure a shared thread pool that is available to all your connectors. This places an upper limit on the number of concurrent threads that may be started by your connectors. Note that this limit applies even if a part icular connector has not used up all the threads configured for it.
   b. **The Executor represents a thread pool that can be shared between components in Tomcat. Historically there has been a thread p ool per connector created but this allows you to share a thread pool, between (primarly) connector but also other components when those get configured to suppor t executors.**
   c. **The Executor element has to appear prior to the Connector element in server.xml**

14. **Listener**
Event listeners are classes that implement one or more of the standard servlet event listener interfaces. For example, a class that implements the javax.servlet.ServletContextListener interface will be notified when the servlet context has been started, and when the servlet context is about to be shut down.

Listener classes are declared in the web application's deployment descriptor. When the web application is deployed, the web container will automatically instantiate each listener that it finds in the deployment descriptor, and will register them with their subjects, according to the interfaces that they implement, and the order in which they appear in the deployment descriptor.

15. **Connector**
   a. Definition: A Connector component provides the external interface that allows clients to connect to the container. This component not onl y accepts incoming connections, but is also responsible for delegating the processing of the request to an available request processor thread.

16. **Difference of CATALINA_HOME and CATALINA_BASE**
CATALINA_HOME is an environment variable that references the location of the Tomcat binaries. The CATALINA_BASE environment variable makes it possible to use a single binary installation of Tomcat to run multiple Tomcat instances with different configurations