**详解 Tomcat 组成与工作原理！**

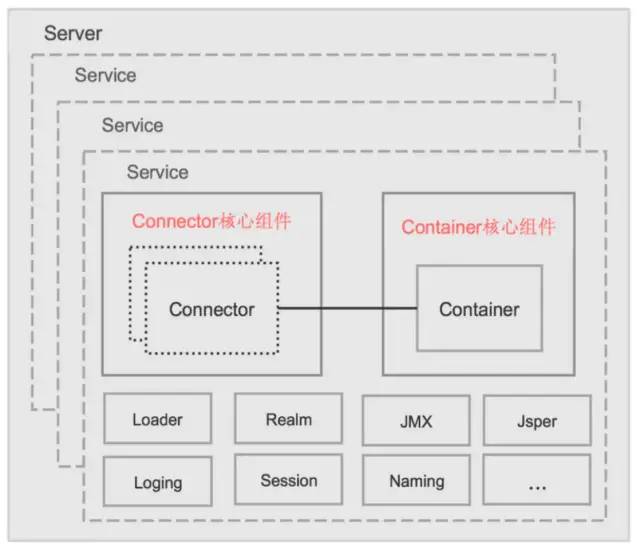
<https://mp.weixin.qq.com/s/n_Y4BG1Jsj-BlKYmjEOIoA>

Tomcat 是什么

开源的 Java Web 应用服务器，实现了 Java EE(Java Platform Enterprise Edition)的部 分技术规范，比如 Java Servlet、Java Server Page、JSTL、Java WebSocket。Java EE 是 Sun 公 司为企业级应用推出的标准平台，定义了一系列用于企业级开发的技术规范，除了上述的之外，还有 EJB、Java Mail、JPA、JTA、JMS 等，而这些都依赖具体容器的实现。

### Servlet 容器

Tomcat 组成如下图：主要有 Container 和 Connector 以及相关组件构成。



Server：指的就是整个 Tomcat 服 务器，包含多组服务，负责管理和 启动各个 Service，同时监听 8005 端口发过来的 shutdown 命令，用 于关闭整个容器 。

Service：Tomcat 封装的、对外提 供完整的、基于组件的 web 服务， 包含 Connectors、Container 两个 核心组件，以及多个功能组件，各 个 Service 之间是独立的，但是共享 同一 JVM 的资源 。

Connector：Tomcat 与外部世界的连接器，监听固定端口接收外部请求，传递给 Container，并 将 Container 处理的结果返回给外部。

Container：Catalina，Servlet 容器，内部有多层容器组成，用于管理 Servlet 生命周期，调用 servlet 相关方法。

Loader：封装了 Java ClassLoader，用于 Container 加载类文件；Realm：Tomcat 中为 web 应用程序提供访问认证和角色管理的机制。

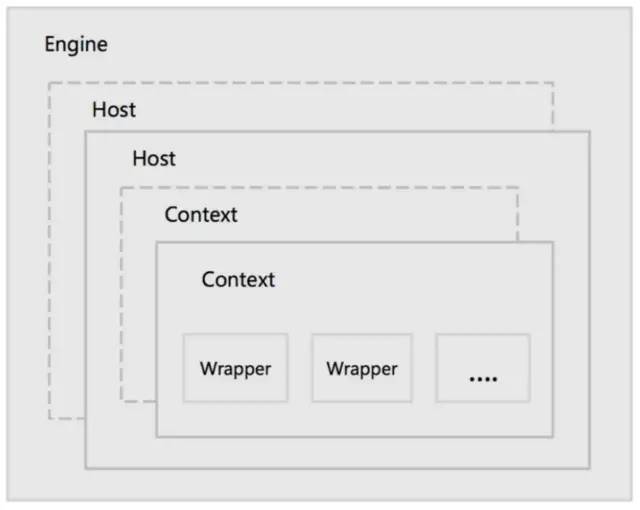
JMX：Java SE 中定义技术规范，是一个为应用程序、设备、系统等植入管理功能的框架，通过 JMX 可以远程监控 Tomcat 的运行状态。

Jasper：Tomcat 的 Jsp 解析引擎，用于将 Jsp 转换成 Java 文件，并编译成 class 文件。Session：负责管理和创建 session，以及 Session 的持久化(可自定义)，支持 session 的集 群。

Pipeline：在容器中充当管道的作用，管道中可以设置各种 valve(阀门)，请求和响应在经由管 道中各个阀门处理，提供了一种灵活可配置的处理请求和响应的机制。

Naming：命名服务，JNDI， Java 命名和目录接口，是一组在 Java 应用中访问命名和目录服务的 API。命名服务将名称和对象联系起来，使得我们可以用名称访问对象，目录服务也是一种命名 服务，对象不但有名称，还有属性。Tomcat 中可以使用 JNDI 定义数据源、配置信息，用于开发 与部署的分离。

### Container 组成

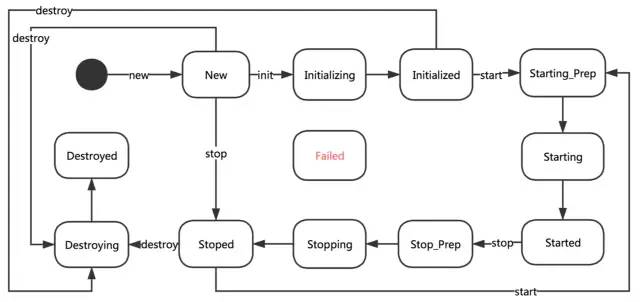


Engine：Servlet 的顶层容器，包含一 个或多个 Host 子容器；Host：虚拟主机，负责 web 应用的部 署和 Context 的创建；Context：Web 应用上下文，包含多个 Wrapper，负责 web 配置的解析、管 理所有的 Web 资源；Wrapper：最底层的容器，是对 Servlet 的封装，负责 Servlet 实例的创 建、执行和销毁。

生命周期管理

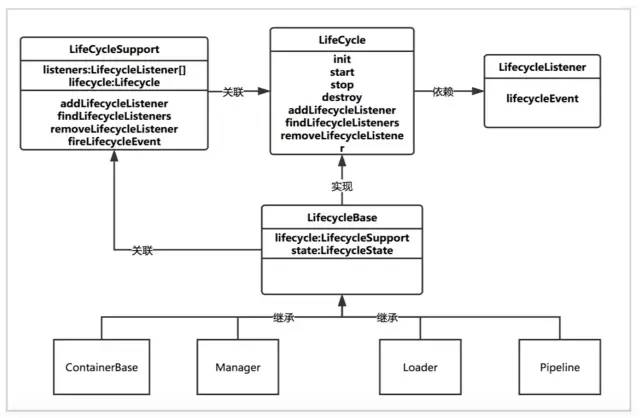
Tomcat 为了方便管理组件和容器的生命周期，定义了从创建、启动、到停止、销毁共 12 中状态，tomcat 生命周期管理了内部状态变化的规则控制，组件和容器只需实现相应的生命周期 方法即可完成各生命周期内的操作(initInternal、startInternal、stopInternal、 destroyInternal)；

比如执行初始化操作时，会判断当前状态是否 New，如果不是则抛出生命周期异常；是的 话则设置当前状态为 Initializing，并执行 initInternal 方法，由子类实现，方法执行成功则设置当 前状态为 Initialized，执行失败则设置为 Failed 状态；

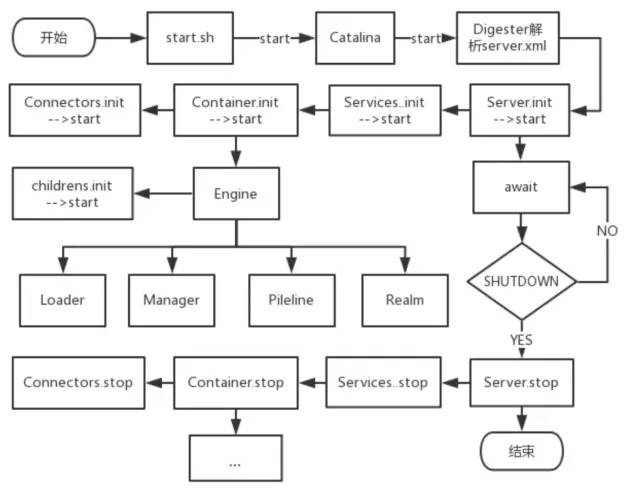


Tomcat 的生命周期管理引入了事件机制，在组件或容器的生命周期状态发生变化时会通 知事件监听器，监听器通过判断事件的类型来进行相应的操作。事件监听器的添加可以在 server.xml 文件中进行配置。

Tomcat 各类容器的配置过程就是通过添加 listener 的方式来进行的，从而达到配置逻辑与 容器的解耦。如 EngineConfig、HostConfig、ContextConfig。EngineConfig：主要打印启动和停止日志 HostConfig：主要处理部署应用，解析应用 META-INF/context.xml 并创建应用的 Context ContextConfig：主要解析并合并 web.xml，扫描应用的各类 web 资源 (filter、servlet、listener)。



### Tomcat 的启动过程



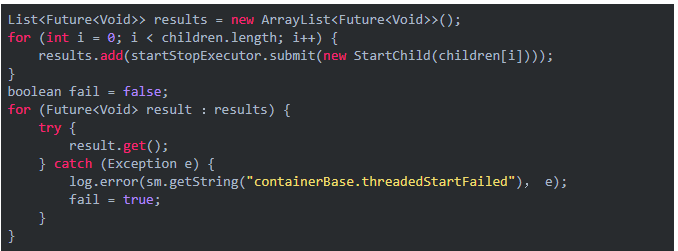
启动从 Tomcat 提供的 start.sh 脚本开始，shell 脚本会调用 Bootstrap 的 main 方法，实际 调用了 Catalina 相应的 load、start 方法。

load 方法会通过 Digester 进行 config/server.xml 的解析，在解析的过程中会根据 xml 中的关系 和配置信息来创建容器，并设置相关的属性。接着 Catalina 会调用 StandardServer 的 init 和 start 方法进行容器的初始化和启动。

按照 xml 的配置关系，server 的子元素是 service，service 的子元素是顶层容器 Engine，每层容器有持有自己的子容器，而这些元素都实现了生命周期管理 的各个方法，因此就很容易的完成整个容器的启动、关闭等生命周期的管理。

StandardServer 完成 init 和 start 方法调用后，会一直监听来自 8005 端口(可配置)，如果接收 到 shutdown 命令，则会退出循环监听，执行后续的 stop 和 destroy 方法，完成 Tomcat 容器的 关闭。同时也会调用 JVM 的 Runtime.getRuntime()﴿.addShutdownHook 方法，在虚拟机意外退 出的时候来关闭容器。

所有容器都是继承自 ContainerBase，基类中封装了容器中的重复工作，负责启动容器相关的组 件 Loader、Logger、Manager、Cluster、Pipeline，启动子容器(线程池并发启动子容器，通过 线程池 submit 多个线程，调用后返回 Future 对象，线程内部启动子容器，接着调用 Future 对象 的 get 方法来等待执行结果)。



Web 应用的部署方式注：catalina.home：安装目录;catalina.base：工作目录;默认值 user.dir：

* Server.xml 配置 Host 元素，指定 appBase 属性，默认$catalina.base/webapps/
* Server.xml 配置 Context 元素，指定 docBase，元素，指定 web 应用的路径
* 自定义配置：在$catalina.base/EngineName/HostName/XXX.xml 配置 Context 元素

HostConfig 监听了 StandardHost 容器的事件，在 start 方法中解析上述配置文件：

* 扫描 appbase 路径下的所有文件夹和 war 包，解析各个应用的 META-INF/context.xml，并 创建 StandardContext，并将 Context 加入到 Host 的子容器中。
* 解析$catalina.base/EngineName/HostName/下的所有 Context 配置，找到相应 web 应 用的位置，解析各个应用的 META-INF/context.xml，并创建 StandardContext，并将 Context 加入到 Host 的子容器中。

注：

* HostConfig 并没有实际解析 Context.xml，而是在 ContextConfig 中进行的。
* HostConfig 中会定期检查 watched 资源文件(context.xml 配置文件)

ContextConfig 解析 context.xml 顺序：

* 先解析全局的配置 config/context.xml
* 然后解析 Host 的默认配置 EngineName/HostName/context.xml.default
* 最后解析应用的 META-INF/context.xml

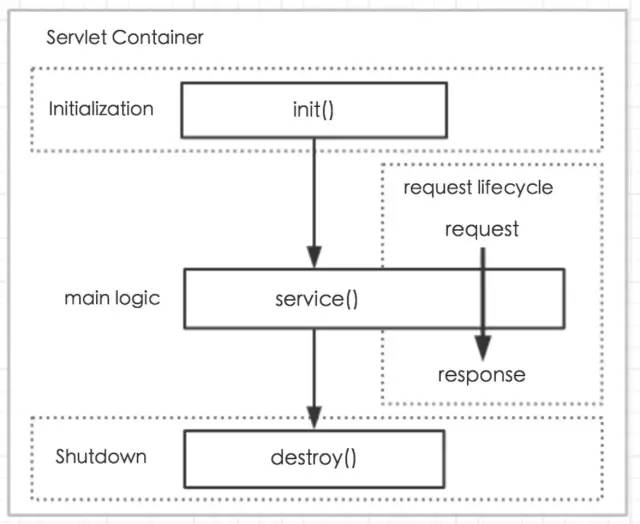
ContextConfig 解析 web.xml 顺序：

* 先解析全局的配置 config/web.xml
* 然后解析 Host 的默认配置 EngineName/HostName/web.xml.default 接着解析应用的 MEB-INF/web.xml
* 扫描应用 WEB-INF/lib/下的 jar 文件，解析其中的 META-INF/web-fragment.xml 最后合并 xml 封装成 WebXml，并设置 Context

注：

* 扫描 web 应用和 jar 中的注解(Filter、Listener、Servlet)就是上述步骤中进行的。
* 容器的定期执行：backgroundProcess，由 ContainerBase 来实现的，并且只有在顶层容器 中才会开启线程。(backgroundProcessorDelay=10 标志位来控制)

### Servlet 生命周期



Servlet 是用 Java 编写的服务器端程序。其主要功能在于交互式地浏览和修改数据，生成动态 Web 内容。

* 请求到达 server 端，server 根据 url 映射到相应的 Servlet
* 判断 Servlet 实例是否存在，不存在则加载和实例化 Servlet 并调用 init 方法
* Server 分别创建 Request 和 Response 对象，调用 Servlet 实例的 service 方法(service 方法 内部会根据 http 请求方法类型调用相应的 doXXX 方法)
* doXXX 方法内为业务逻辑实现，从 Request 对象获取请求参数，处理完毕之后将结果通过 response 对象返回给调用方
* 当 Server 不再需要 Servlet 时(一般当 Server 关闭时)，Server 调用 Servlet 的 destroy() 方 法。

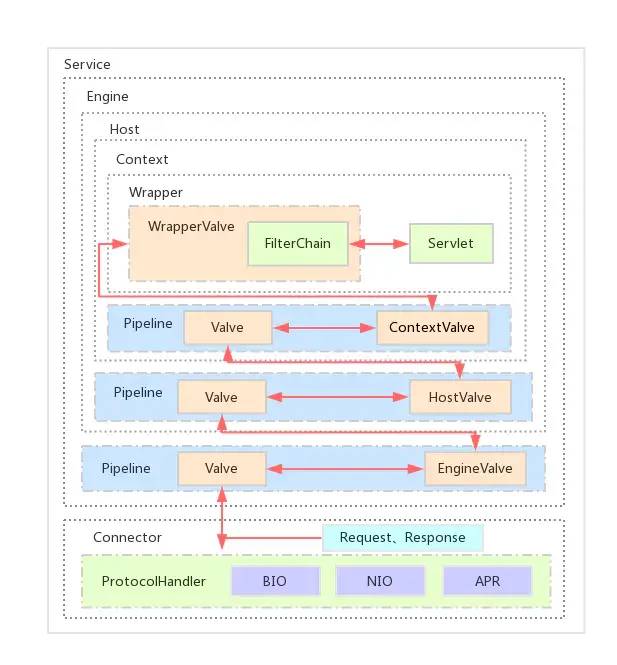
### load on startup

当值为 0 或者大于 0 时，表示容器在应用启动时就加载这个 servlet; 当是一个负数时或者没有指定时，则指示容器在该 servlet 被选择时才加载; 正数的值越小，启动该 servlet 的优先级越高。

#### single thread model

每次访问 servlet，新建 servlet 实体对象，但并不能保证线程安全，同时 tomcat 会限制 servlet 的实例数目 最佳实践：不要使用该模型，servlet 中不要有全局变量。

### 请求处理过程



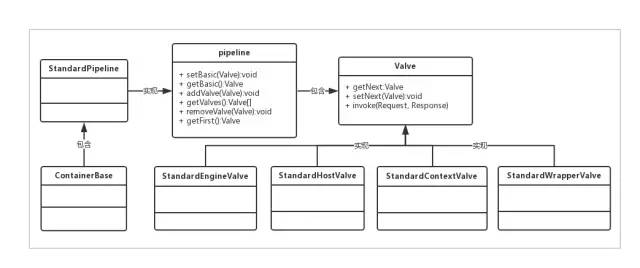
根据 server.xml 配置的指定的 connector 以及端口监听 http、或者 ajp 请求

请求到来时建立连接,解析请求参数,创建 Request 和 Response 对象,调用顶层容器 pipeline 的 invoke 方法

容器之间层层调用,最终调用业务 servlet 的 service 方法

Connector 将 response 流中的数据写到 socket 中

### Pipeline 与 Valve



Pipeline 可以理解为现实中的管道,Valve 为管道中的阀门,Request 和 Response 对象在管道中 经过各个阀门的处理和控制。

每个容器的管道中都有一个必不可少的 basic valve,其他的都是可选的,basic valve 在管道中最 后调用,同时负责调用子容器的第一个 valve。

Valve 中主要的三个方法:setNext、getNext、invoke;valve 之间的关系是单向链式结构,本身 invoke 方法中会调用下一个 valve 的 invoke 方法。

各层容器对应的 basic valve 分别是 StandardEngineValve、StandardHostValve、 StandardContextValve、StandardWrapperValve。