

1. tomcat历史
2. Tomcat总体框架
3. Tomcat源码搭建
4. tomcat总体架构

1 Tomcat历史

Tomcat最初有sun公司的架构师James Duncan Davidson开发，名称“JavaWebServer”

1999与Apache软件基金会旗下的JServ项目合并，也就是Tomcat。

2001 tomcat4.0 里程碑式的版本。完全重新设计了其架构，并实现了Servlet2.3和JSP 1.2规范。

到目前，Tomcat已经成为成熟的Servlet容器产品，并作为 JBoss等应用的服务器的内嵌Servlet容器

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 规范JDK  版本 | 6.X | 7.X | 8.X | 8.5.X | 9.X |
| JDK | 》=5.0 | 》=6.0 | 》=7.0 | 》=7.0 | 》=8.0 |
| Servlet | 2.5 | 3.0 | 3.1 | 3.1 | 4.0 |
| JSP | 2.1 | 2.2 | 2.3 | 2.3 | 2.3 |
| EL | 2.1 | 2.2 | 3.0 | 3.0 | 3.0 |
| WebSocket | N/A | 1.1 | 1.1 | 1.1 | 1.1 |

Tomcat许可

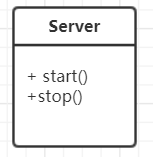
完全免费

修改后不必公开源代码

2、总体结构

Server：接受请求并解析，完成相关任务，返回处理结果

通常情况下使用Socket监听服务器指定端口来实现该功能，一个最简单的服务设计如下：

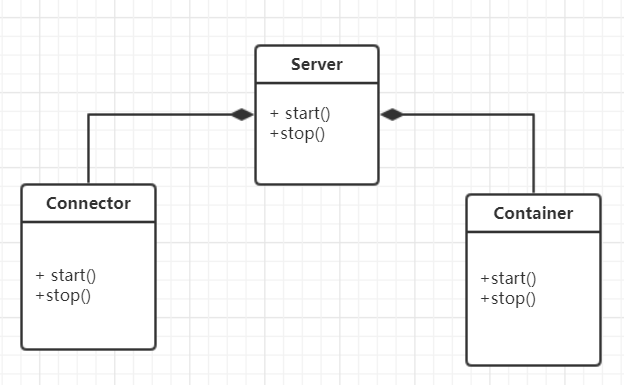


Start()：启动服务器，打开socket连接，监听服务端口，接受客户端请求、处理、返回响应

Stop()：关闭服务器，释放资源

缺点：请求监听和请求处理放一起扩展性很差（协议的切换 tomcat独立部署使用HTTP协议，与Apache集成时使用AJP协议）

改进：网络协议与请求处理分离



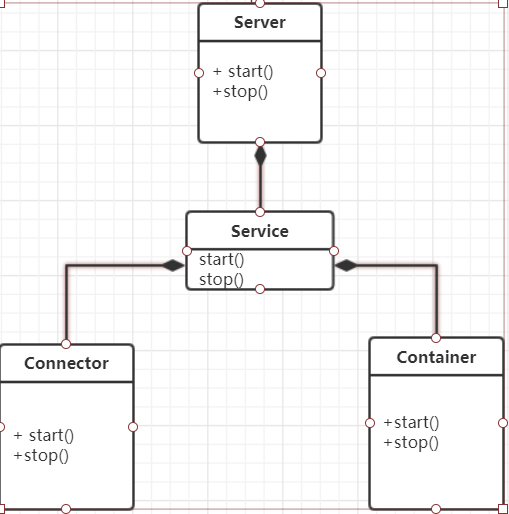
一个Server包含多个Connector（链接器）和Container（容器）

Connector：开启Socket并监听客户端请求，返回响应数据；

Container：负责具体的请求处理

缺点：Connector接受的请求由那个Container处理，需要建立映射规则

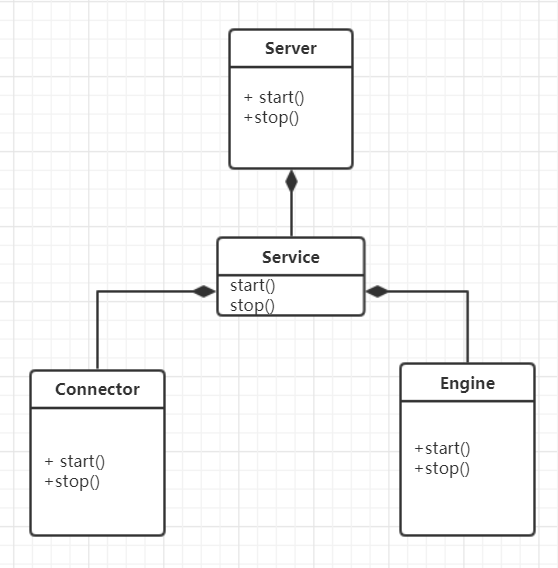
改进：



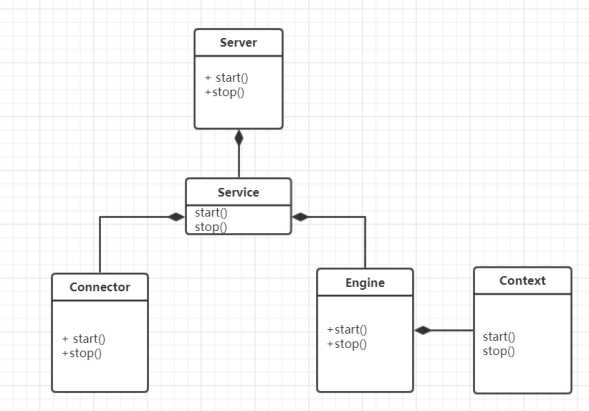
一个Server可以包含多个Service，每一个Service都是独立的，他们共享一个JVM以及系统类库。

一个Service负责维护多个Connector和一个Container，这样来自Connector的请求只能有它所属的Service维护的Container处理。

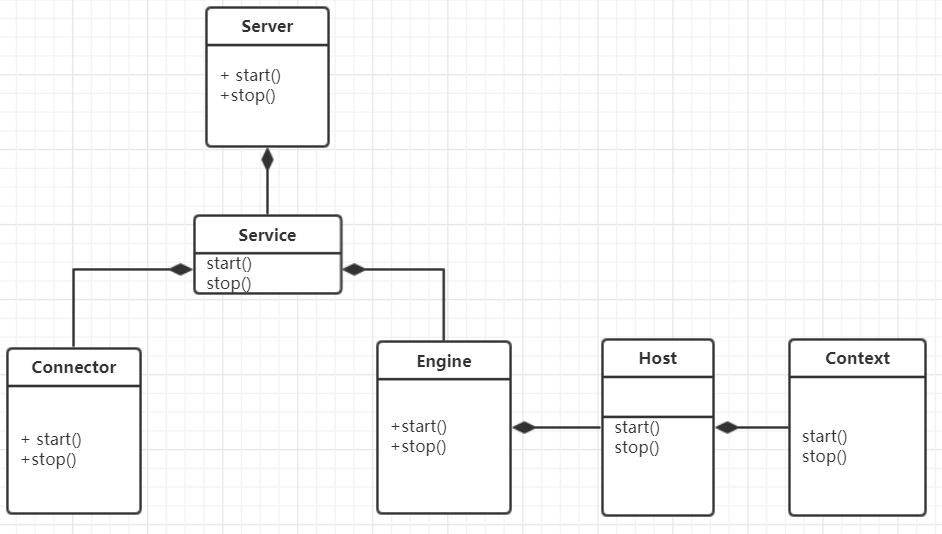
在这里Container是一个通用的概念，为了明确功能，并与Tomcat中的组件名称相同，可以将Container命名为Engineer



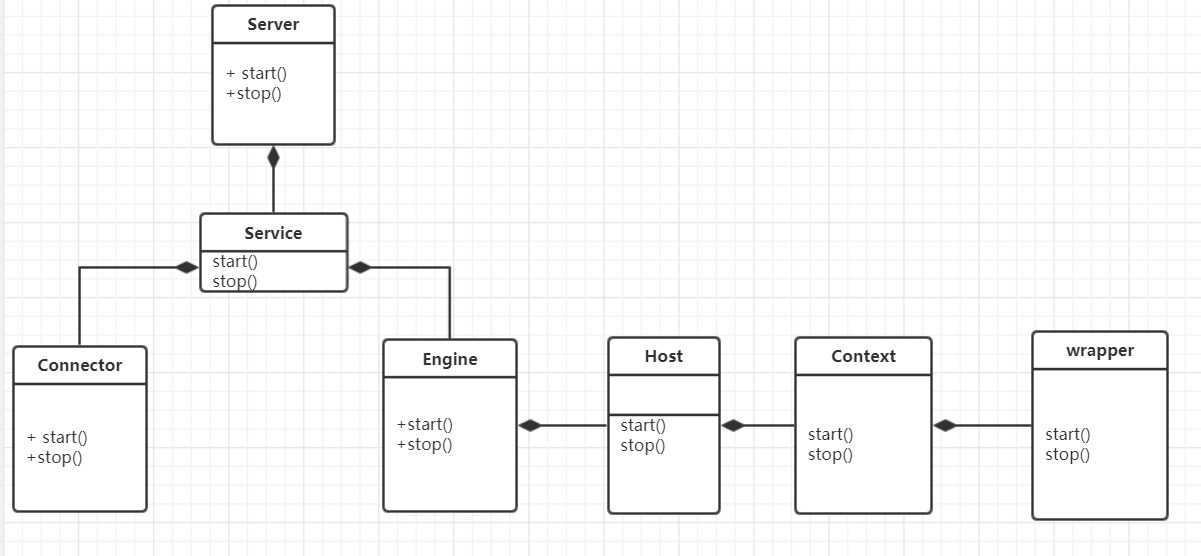
在Engine容器中需要支持管理WEB应用，当接收到Connector的处理请求时，Engine容器能够找到一个合适的Web应用来处理，因此在上面设计的基础上增加Context来表示一个WEB应用，并且一个Engine可以包含多个Context。



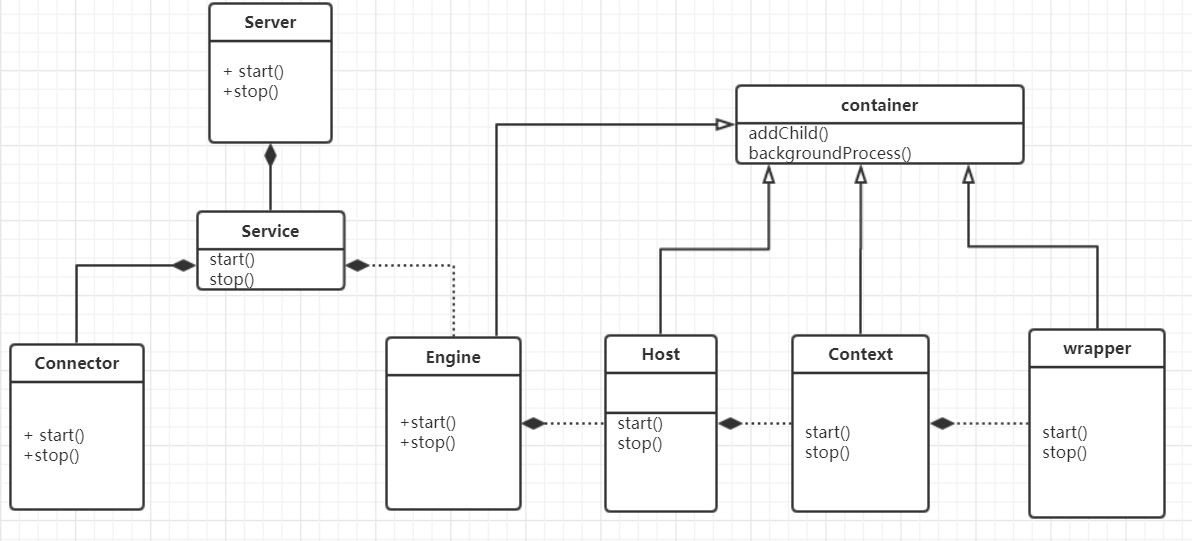
缺点：应用服务器需要将每个域名抽象为一个虚拟主机，



在一个web应用中，可以包含多个Servlet实例来处理来自不同的链接请求，因此我们还需要一个组件概念来表示Servlet定义，即Wrapper。

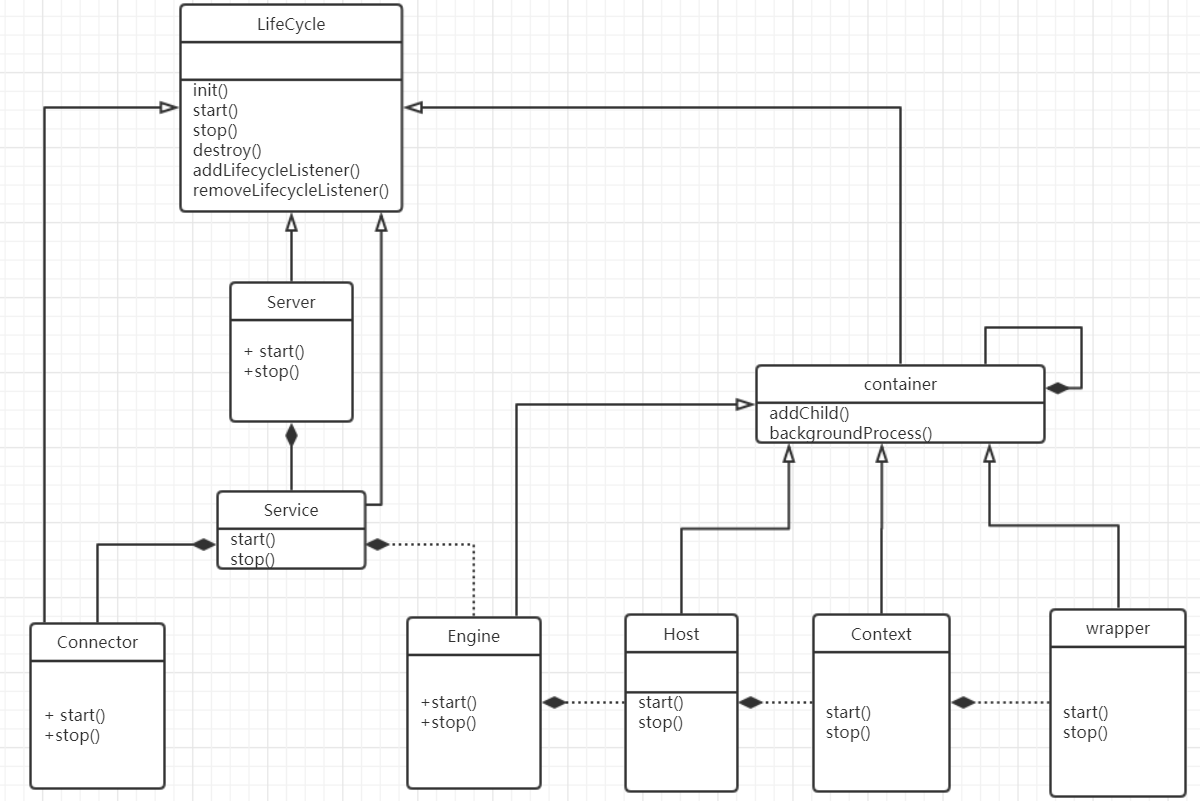


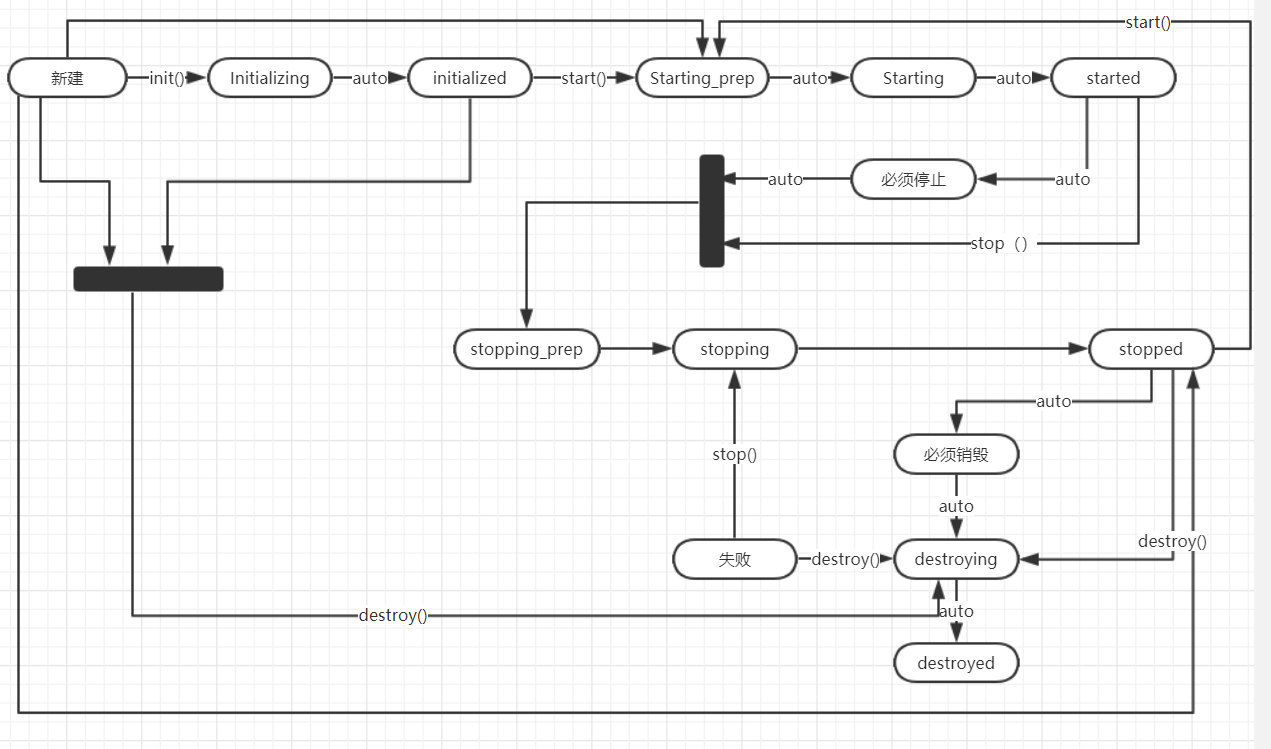
在前面的多次Container容器中，有Engine、Host、Context、Wrapper等，可以理解为Container的子类.



容器之间的组合关系是一种弱依赖，用虚线表示。

每一个组件都有启动、停止等生命周期方法，拥有生命周期的特征。所以定义一个通用的LifeCycle接口，





Pipeline和Value

Connector设计

功能：

* 监听服务器端口，读取来自客户端的请求
* 使用指定的协议解析请求数据
* 根据请求地址匹配正确的容器解析处理
* 讲响应返回给客户端

HTTP ajp I/O: BIO(tomcat 8 以后移除) NIO NIO2 APR