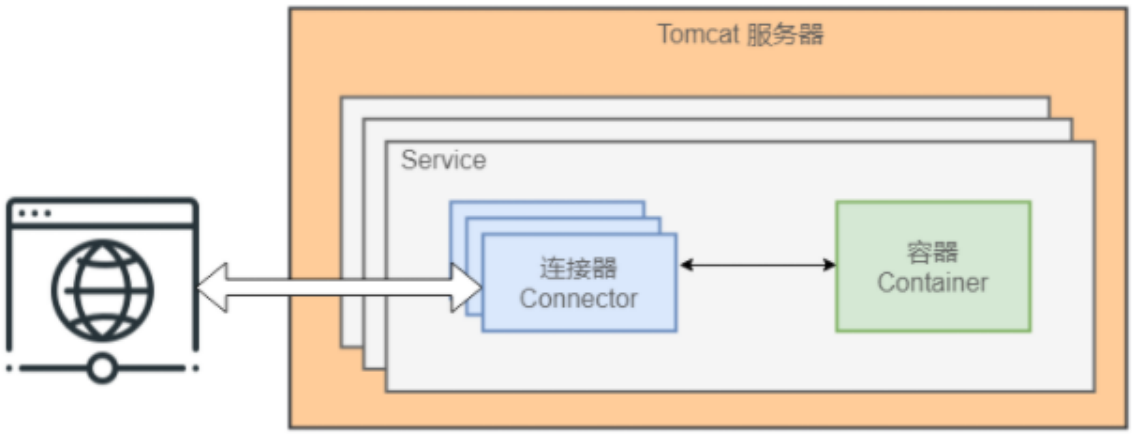
Tomcat

整体结构

目录	功能说明
bin	存放可执行的文件，如 startup 和 shutdown
conf	存放配置文件，如核心配置文件 server.xml 和应用默认的部署描述文件 web.xml
lib	存放 Tomcat 运行需要的jar包
logs	存放运行的日志文件
webapps	存放默认的 web 应用部署目录
work	存放 web 应用代码生成和编译文件的临时目录

功能组件结构

- 连接器（connector），负责接收和反馈外部请求。是Tomcat与外界的枢纽，监听端口接收外界请求，并将请求处理后传递给容器做业务处理，最后将容器处理后的结果反馈给外界。
- 容器（container）负责处理请求，内部由Engine，Host，Context和Wrapper四个容器组成，用于管理和调用servlet相关逻辑。
- Service：对外提供的Web服务，主要包含连接器和容器两个核心组件。Tomcat可以管理多个service，且各service之间相互独立



连接器

Tomcat连接器框架：Coyote

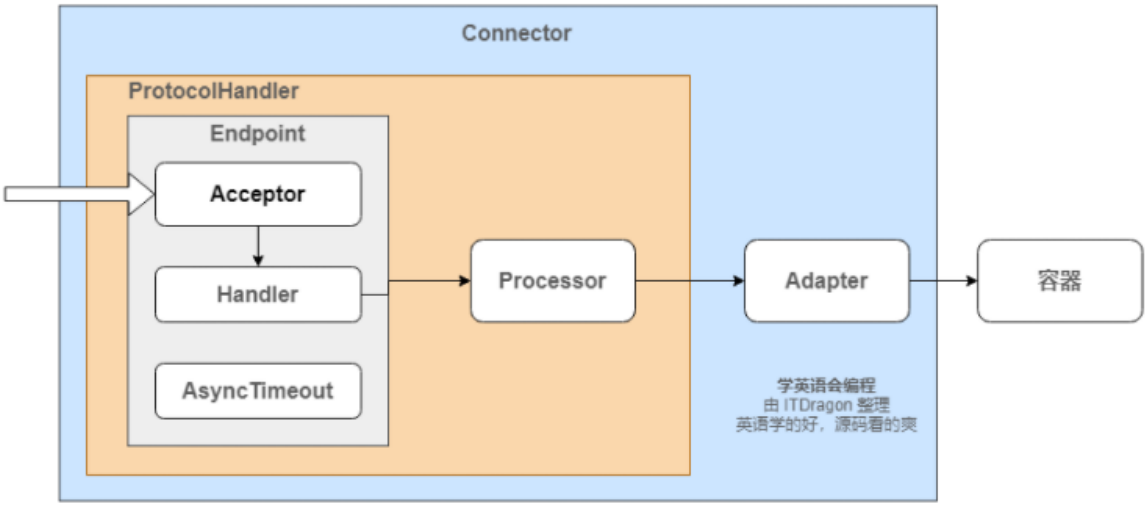
1. 监听网络端口，接收和响应网络请求。
2. 网络字节流处理。

   网络字节流 -> Tomcat Request -> 标准ServletRequest 给容器

从容器 ServletResponse -> Tomcat Response -> 网络字节流

连接器模块

- Endpoint：端点，用来处理socket发送和接收的逻辑，内部由Acceptor监听请求，Handler处理数据，AsyncTimeout检查请求超时
- Processor：处理器，负责构建Tomcat Request和Response对象
- Adapter：适配器，实现Tomcat Request和Response，与ServletRequest，ServletResponse之间的相互转换
- ProtocolHandler：协议处理器，将不同的协议和通讯方式封装成对应的协议处理器。



容器

Tomcat容器框架：Catalina

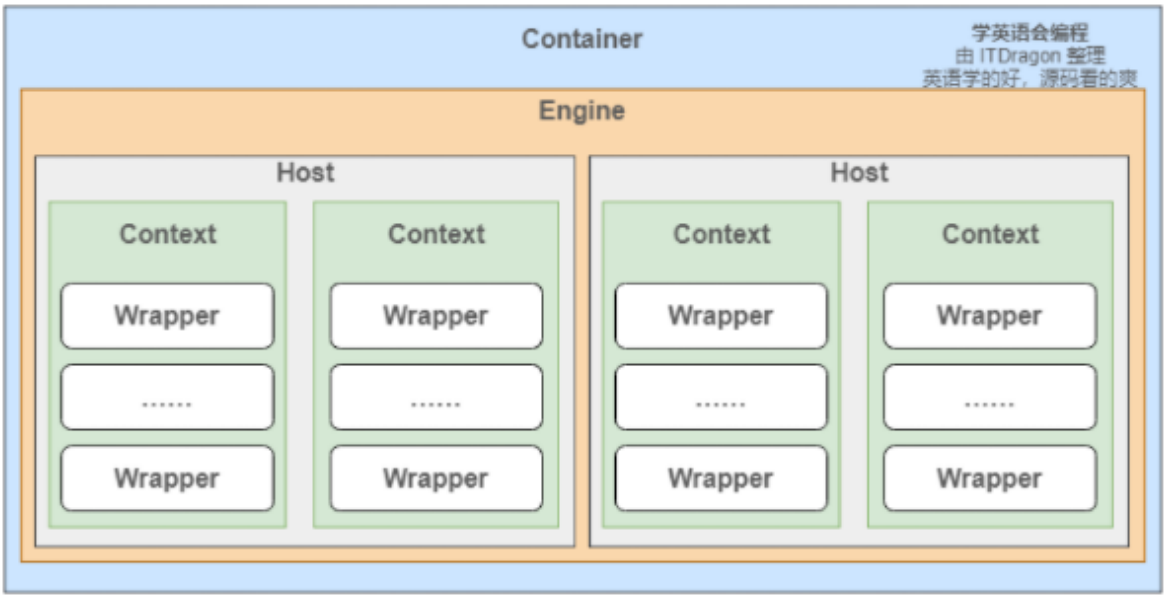
每个Service会包含一个容器。

容器由一个引擎可以管理多个虚拟主机。

每个虚拟主机可以管理多个Web应用。

每个Web应用会有多个Servlet包装器

- Engine：引擎，管理多个虚拟主机
- Host：虚拟主机，负责Web应用的部署
- Context：Web应用，包含多个Servlet封装器
- Wrapper：封装器，容器的最底层。对Servlet进行封装，负责实例的创建、执行和销毁功能



容器请求处理

在Engine，Host，Context和Wrapper之间层层调用，最后在Servlet中执行对应的业务逻辑。各容器会有一个通道Pipeline，每个通道上都会有一个 Basic Valve（如StandardEngineValve）， 类似一个闸门用来处理 Request 和 Response 。

Tomcat请求处理流程

Mapper

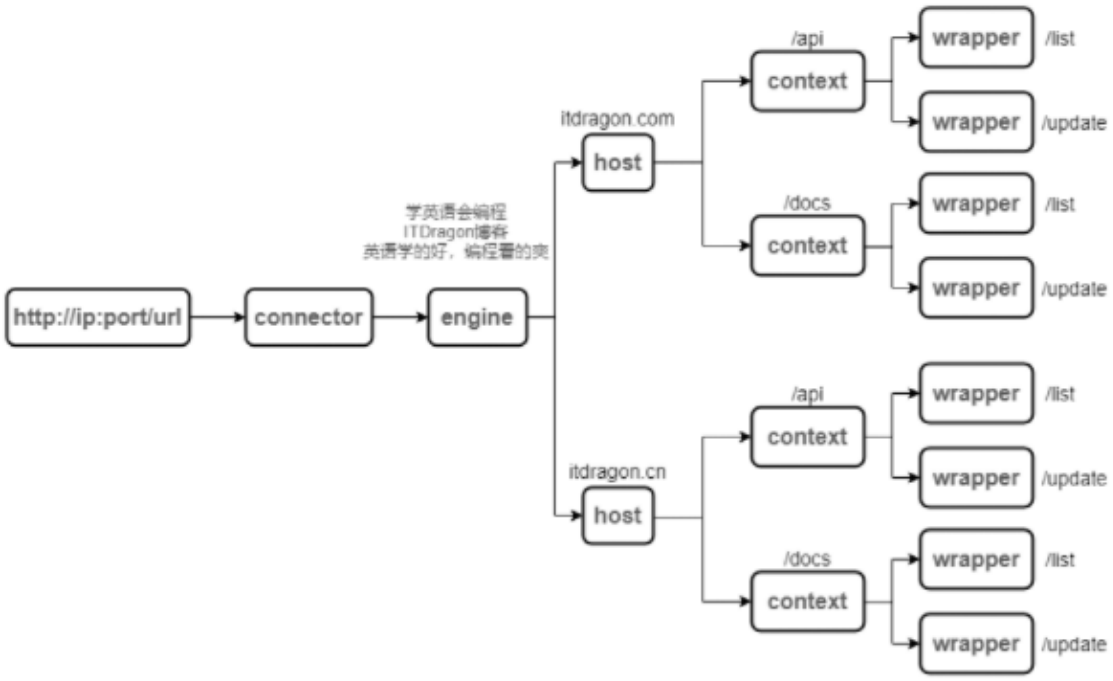
作用是提供请求路径的路由映射。根据请求URL地址匹配是由哪个容器来处理，其中每个容器都会有自己对应的Mapper。

第一步：连接器监听的端口是8080。由于请求的端口和监听的端口一致，连接器接受了该请求。

第二步：因为引擎的默认虚拟主机是 localhost，并且虚拟主机的目录是webapps。所以请求找到了 tomcat/webapps 目录。

第三步：解析的 docs 是 web 程序的应用名，也就是 context。此时请求继续从 webapps 目录下找 docs 目录。有的时候我们也会把应用名省略。

第四步：解析的 api 是具体的业务逻辑地址。此时需要从 docs/WEB-INF/web.xml 中找映射关系，最后调用具体的函数。



假设来自客户的请求为：`http://localhost:8080/test/index.jsp`

1. 请求被发送到本机端口8080，被在那里侦听的Coyote HTTP/1.1 Connector获得；

2. Connector把该请求交给它所在的Service的Engine来处理，并等待Engine的回应；

3. Engine获得请求localhost:8080/test/index.jsp，匹配它所有虚拟主机Host；

4. Engine匹配到名为localhost的Host（即使匹配不到也把请求交给该Host处理，因为该Host被定义为该Engine的默认主机）；

5. localhost Host获得请求/test/index.jsp，匹配它所拥有的所有Context；

6. Host匹配到路径为/test的Context（如果匹配不到就把该请求交给路径名为""的Context去处理）；

7. path="/test"的Context获得请求/index.jsp，在它的mapping table中寻找对应的servlet；

8. Context匹配到URL PATTERN为*.jsp的servlet，对应于JspServlet类；

9. 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法；

10. Context把执行完了之后的HttpServletResponse对象返回给Host；

11. Host把HttpServletResponse对象返回给Engine；

12. Engine把HttpServletResponse对象返回给Connector；

13. Connector把HttpServletResponse对象返回给客户browser；