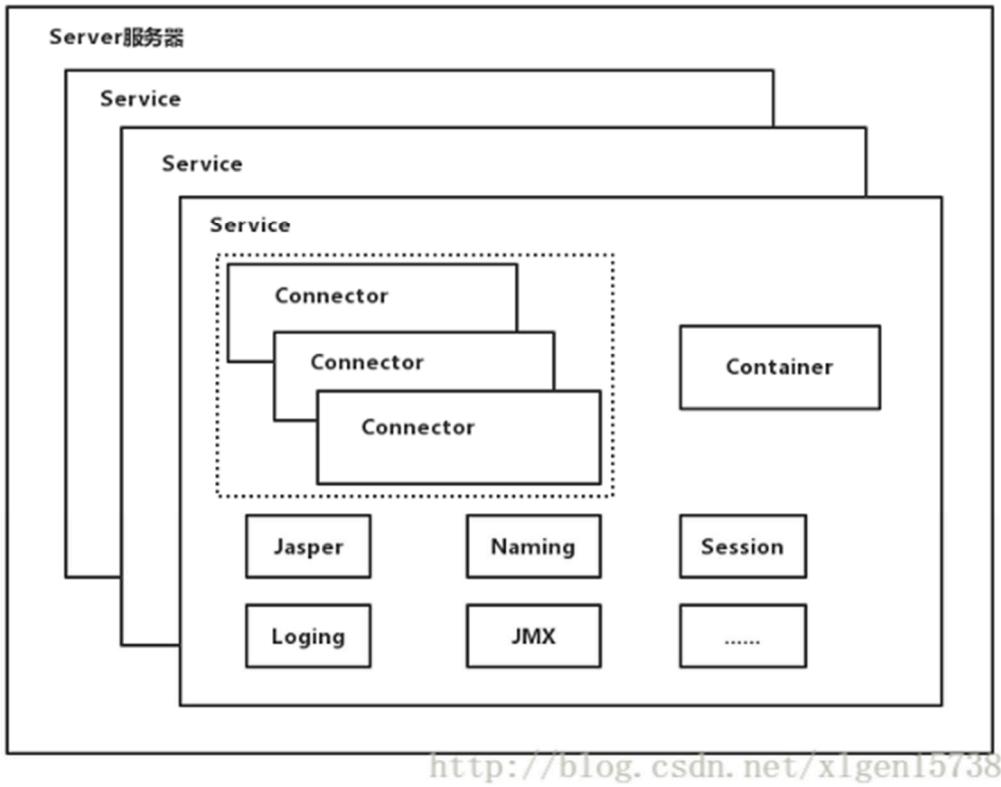
# 一、Tomcat 顶层架构

先上一张 Tomcat 的顶层结构图（图 A），如下：

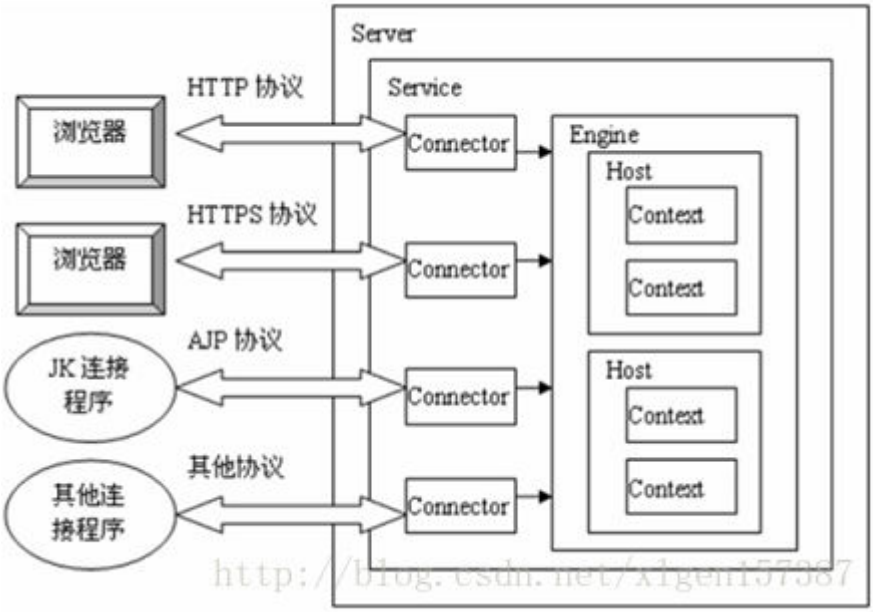



Tomcat 中最顶层的容器是 Server，代表着整个服务器，从上图中可以看出，一个 Server 可以包含至少一个 Service，用于具体提供服务。

Service 主要包含两个部分：Connector 和 Container。从上图中可以看出 Tomcat 的心脏就是这两个组件，他们的作用如下：

1、Connector 用于处理连接相关的事情，并提供 Socket 与 Request 和 Response 相关的转化;
2、Container 用于封装和管理 Servlet，以及具体处理 Request 请求；

一个 Tomcat 中只有一个 Server，一个 Server 可以包含多个 Service，一个 Service 只有一个 Container，但是可以有多个 Connectors，这是因为一个服务可以有多个连接，如同时提供 Http 和 Https 链接，也可以提供向相同协议不同端口的连接,示意图如下（Engine、Host、Context 下边会说到）：




多个 Connector 和一个 Container 就形成了一个 Service，有了 Service 就可以对外提供服务了，但是 Service 还要一个生存的环境，必须要有人能够给她生命、掌握其生死大权，那就非 Server 莫属了！所以整个 Tomcat 的生命周期由 Server 控制。

另外，上述的包含关系或者说是父子关系，都可以在 tomcat 的 conf 目录下的 server.xml 配置文件中看出，下图是删除了注释内容之后的一个完整的 server.xml 配置文件（Tomcat 版本为 8.0）

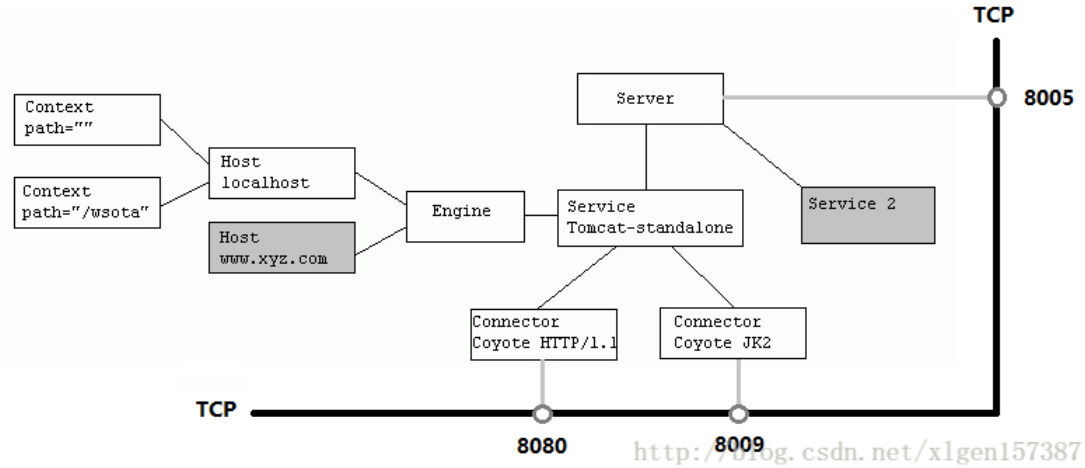
```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase"/>
      </Realm>
      <Host name="localhost"  appBase="webapps" unpackWARs="true" autoDeploy="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &quot;%r&quot; %s %b" />
      </Host>
    </Engine>
  </Service>
</Server>
```



上边的配置文件，还可以通过下边的一张结构图更清楚的理解：





Server 标签设置的端口号为 8005，shutdown=”SHUTDOWN” ，表示在 8005 端口监听“SHUTDOWN”命令，如果接收到了就会关闭 Tomcat。一个 Server 有一个 Service，当然还可以进行配置，一个 Service 有多个，Service 左边的内容都属于 Container 的，Service 下边是 Connector。

二、Tomcat 顶层架构小结：
（1）Tomcat 中只有一个 Server，一个 Server 可以有多个 Service，一个 Service 可以有多个 Connector 和一个 Container；
（2） Server 掌管着整个 Tomcat 的生死大权；
（4）Service 是对外提供服务的；
（5）Connector 用于接受请求并将请求封装成 Request 和 Response 来具体处理；
（6）Container 用于封装和管理 Servlet，以及具体处理 request 请求；

知道了整个 Tomcat 顶层的分层架构和各个组件之间的关系以及作用，对于绝大多数的开发人员来说 Server 和 Service 对我们来说确实很远，而我们开发中绝大部分进行配置的内容是属于 Connector 和 Container 的，所以接下来介绍一下 Connector 和 Container。

三、Connector 和 Container 的微妙关系
由上述内容我们大致可以知道一个请求发送到 Tomcat 之后，首先经过 Service 然后会交给我们的 Connector，Connector 用于接收请求并将接收的请求封装为 Request 和 Response 来具体处理，Request 和 Response 封装完之后再交由 Container 进行处理，Container 处理完请求之后再返回给 Connector，最后在由 Connector 通过 Socket 将处理的结果返回给客户端，这样整个请求的就处理完了！

Connector 最底层使用的是 Socket 来进行连接的，Request 和 Response 是按照 HTTP 协议来封装的，所以 Connector 同时需要实现 TCP/IP 协议和 HTTP 协议！

Tomcat 既然处理请求，那么肯定需要先接收到这个请求，接收请求这个东西我们首先就需要看一下 Connector！

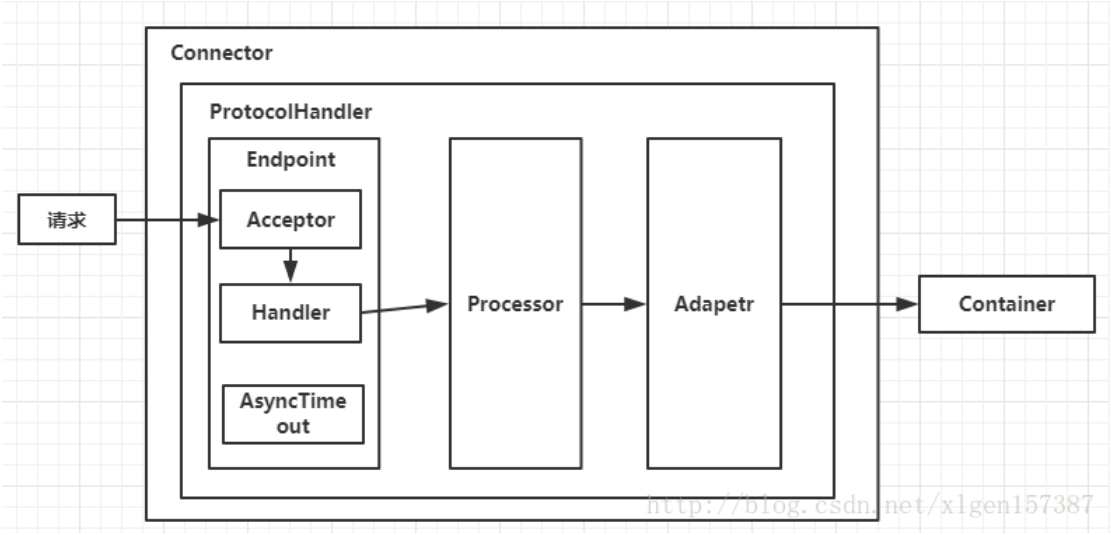
四、Connector 架构分析

Connector 用于接受请求并将请求封装成 Request 和 Response，然后交给 Container 进行处理，Container 处理完之后在交给 Connector 返回给客户端。

因此，我们可以把 Connector 分为四个方面进行理解：

（1）Connector 如何接受请求的？
（2）如何将请求封装成 Request 和 Response 的？
（3）封装完之后的 Request 和 Response 如何交给 Container 进行处理的？
（4）Container 处理完之后如何交给 Connector 并返回给客户端的？

首先看一下 Connector 的结构图（图 B），如下所示：




Connector 就是使用 ProtocolHandler 来处理请求的，不同的 ProtocolHandler 代表不同的连接类型，比如：Http11Protocol 使用的是普通 Socket 来连接的，Http11NioProtocol 使用的是 NioSocket 来连接的。

其中 ProtocolHandler 由包含了三个部件：Endpoint、Processor、Adapter。

（1）Endpoint 用来处理底层 Socket 的网络连接，Processor 用于将 Endpoint 接收到的 Socket 封装成 Request，Adapter 用于将 Request 交给 Container 进行具体的处理。

（2）Endpoint 由于是处理底层的 Socket 网络连接，因此 Endpoint 是用来实现 TCP/IP 协议的，而 Processor 用来实现 HTTP 协议的，Adapter 将请求适配到 Servlet 容器进行具体的处理。

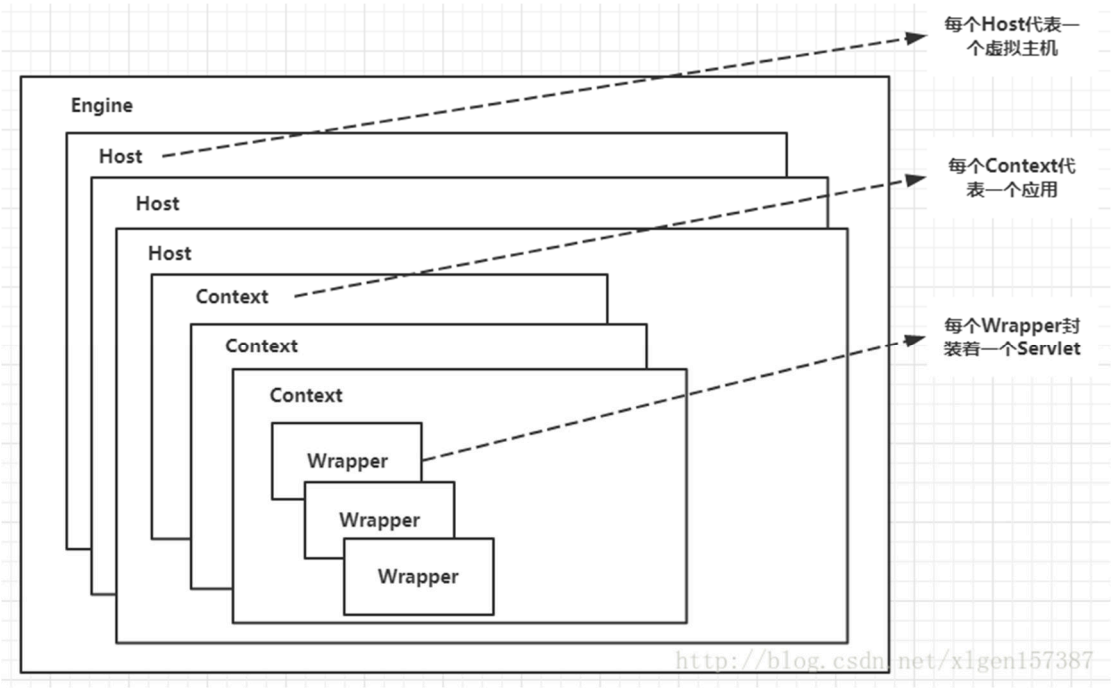
（3）Endpoint 的抽象实现 AbstractEndpoint 里面定义的 Acceptor 和 AsyncTimeout 两个内部类和一个 Handler 接口。Acceptor 用于监听请求，AsyncTimeout 用于检查异步 Request 的超时，Handler 用于处理接收到的 Socket，在内部调用 Processor 进行处理。

至此，我们应该很轻松的回答（1）（2）（3）的问题了，但是（4）还是不知道，那么我们就

来看一下 Container 是如何进行处理的以及处理完之后是如何将处理完的结果返回给 Connector 的？

五、Container 架构分析

Container 用于封装和管理 Servlet，以及具体处理 Request 请求，在 Connector 内部包含了 4 个子容器，结构图如下（图 C）：



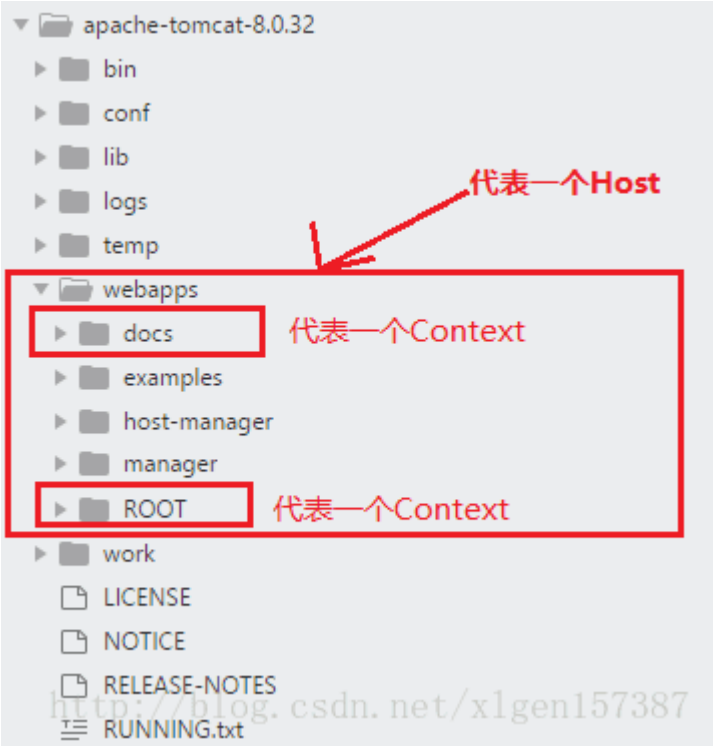

4 个子容器的作用分别是：

（1）Engine：引擎，用来管理多个站点，一个 Service 最多只能有一个 Engine；

（2）Host：代表一个站点，也可以叫虚拟主机，通过配置 Host 就可以添加站点；

（3）Context：代表一个应用程序，对应着平时开发的一套程序，或者一个 WEB-INF 目录以及下面的 web.xml 文件；

（4）Wrapper：每一 Wrapper 封装着一个 Servlet；





Context 和 Host 的区别是 Context 表示一个应用，我们的 Tomcat 中默认的配置下 webapps 下的每一个文件夹目录都是一个 Context，其中 ROOT 目录中存放着主应用，其他目录存放着子应用，而整个 webapps 就是一个 Host 站点。

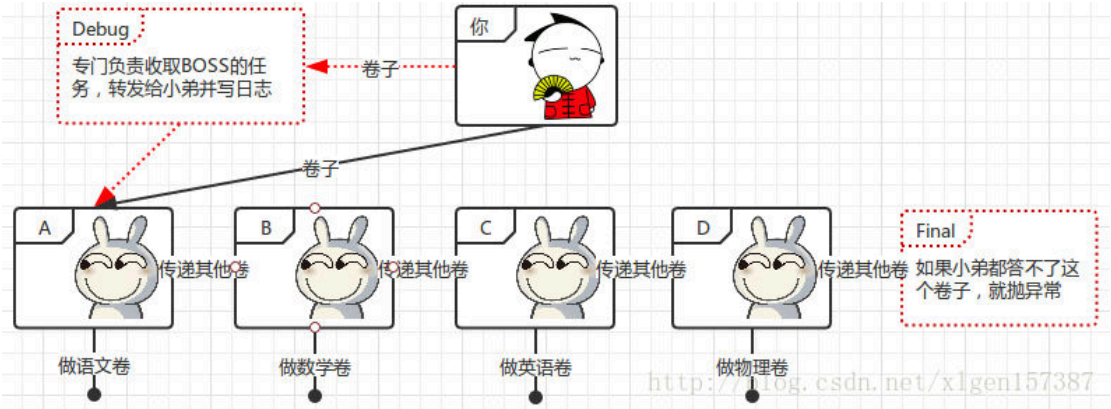
我们访问应用 Context 的时候，如果是 ROOT 下的则直接使用域名就可以访问，例如：www.ledouit.com,如果是 Host（webapps）下的其他应用，则可以使用 www.ledouit.com/docs 进行访问，当然默认指定的根应用（ROOT）是可以进行设定的，只不过 Host 站点下默认的主营用是 ROOT 目录下的。

看到这里我们知道 Container 是什么，但是还是不知道 Container 是如何进行处理的以及处理完之后是如何将处理完的结果返回给 Connector 的？别急！下边就开始探讨一下 Container 是如何进行处理的！

六、Container 如何处理请求的

Container 处理请求是使用 Pipeline-Valve 管道来处理的！（Valve 是阀门之意）

Pipeline-Valve 是责任链模式，责任链模式是指在一个请求处理的过程中有很多处理者依次对请求进行处理，每个处理者负责做自己相应的处理，处理完之后将处理后的请求返回，再让下一个处理着继续处理。



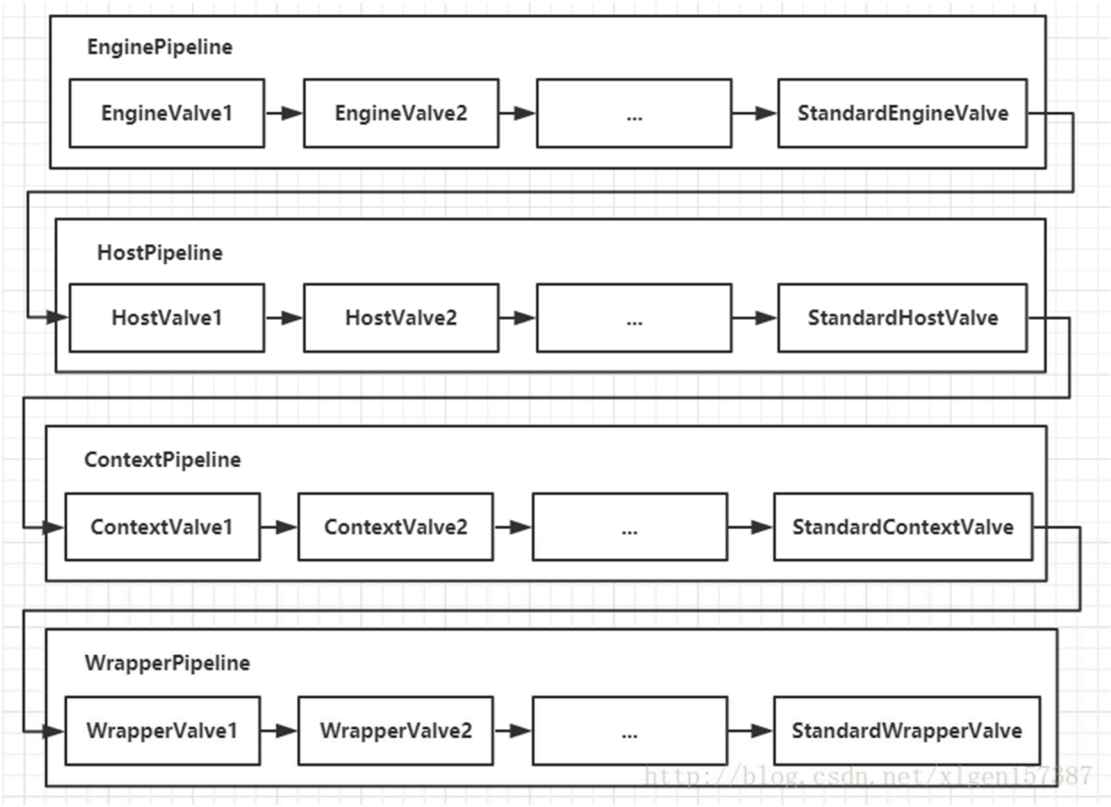

但是!Pipeline-Valve 使用的责任链模式和普通的责任链模式有些不同!区别主要有以下两点：

（1)每个 Pipeline 都有特定的 Valve，而且是在管道的最后一个执行，这个 Valve 叫做 BaseValve，BaseValve 是不可删除的；

（2）在上层容器的管道的 BaseValve 中会调用下层容器的管道。

我们知道 Container 包含四个子容器，而这四个子容器对应的 BaseValve 分别在：StandardEngineValve、StandardHostValve、StandardContextValve、StandardWrapperValve。

Pipeline 的处理流程图如下（图 D)：




（1）Connector 在接收到请求后会首先调用最顶层容器的 Pipeline 来处理，这里的最顶层容

器的 Pipeline 就是 EnginePipeline（Engine 的管道）；

（2）在 Engine 的管道中依次会执行 EngineValve1、EngineValve2 等等，最后会执行 StandardEngineValve，在 StandardEngineValve 中会调用 Host 管道，然后再依次执行 Host 的 HostValve1、HostValve2 等，最后在执行 StandardHostValve，然后再依次调用 Context 的管道和 Wrapper 的管道，最后执行到 StandardWrapperValve。

（3）当执行到 StandardWrapperValve 的时候，会在 StandardWrapperValve 中创建 FilterChain，并调用其 doFilter 方法来处理请求，这个 FilterChain 包含着我们配置的与请求相匹配的 Filter 和 Servlet，其 doFilter 方法会依次调用所有的 Filter 的 doFilter 方法和 Servlet 的 service 方法，这样请求就得到了处理！

（4）当所有的 Pipeline-Valve 都执行完之后，并且处理完了具体的请求，这个时候就可以将返回的结果交给 Connector 了，Connector 在通过 Socket 的方式将结果返回给客户端。