Tomcat是什么？

Tomcat 服务器Apache软件基金会项目中的一个核心项目，是一个免费的开放源代码的Web 应用服务器，属于轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试JSP 程序的首选。

Tomcat的缺省端口是多少，怎么修改

1. 找到Tomcat目录下的conf文件夹
2. 进入conf文件夹里面找到server.xml文件
3. 打开server.xml文件
4. 在server.xml文件里面找到下列信息
5. 把Connector标签的8080端口改成你想要的端口

```
<Service name="Catalina">
<Connector port="8080" protocol="HTTP/1.1"
 connectionTimeout="20000"
 redirectPort="8443" />
```

tomcat 有哪几种Connector 运行模式(优化)？

下面，我们先大致了解Tomcat Connector的三种运行模式。

- **BIO：同步并阻塞** 一个线程处理一个请求。缺点：并发量高时，线程数较多，浪费资源。Tomcat7或以下，在Linux系统中默认使用这种方式。

配制项：protocol=”HTTP/1.1”

- NIO：同步非阻塞IO

利用Java的异步IO处理，可以通过少量的线程处理大量的请求，可以复用同一个线程处理多个connection(多路复用)。

Tomcat8在Linux系统中默认使用这种方式。

Tomcat7必须修改Connector配置来启动。

配制项：protocol=”org.apache.coyote.http11.Http11NioProtocol”

备注：我们常用的Jetty，Mina，ZooKeeper等都是基于java nio实现.

- APR：即Apache Portable Runtime，从操作系统层面解决io阻塞问题。**AIO方式，****异步非阻塞IO**(Java NIO2又叫AIO) 主要与NIO的区别主要是操作系统的底层区别.可以做个比喻:比作快递，NIO就是网购后要自己到官网查下快递是否已经到了(可能是多次)，然后自己去取快递；AIO就是快递员送货上门了(不用关注快递进度)。

配制项：protocol=”org.apache.coyote.http11.Http11AprProtocol”

备注：需在本地服务器安装APR库。Tomcat7或Tomcat8在Win7或以上的系统中启动默认使用这种方式。Linux如果安装了apr和native，Tomcat直接启动就支持apr。

Tomcat有几种部署方式？

在Tomcat中部署Web应用的方式主要有如下几种：

1. 利用Tomcat的自动部署。

把web应用拷贝到webapps目录。Tomcat在启动时会加载目录下的应用，并将编译后的结果放入work目录下。

2. 使用Manager App控制台部署。

在tomcat主页点击“Manager App” 进入应用管理控制台，可以指定一个web应用的路径或war文件。

3. 修改conf/server.xml文件部署。

修改conf/server.xml文件，增加Context节点可以部署应用。

4. 增加自定义的Web部署文件。

在conf/Catalina/localhost/ 路径下增加 xyz.xml文件，内容是Context节点，可以部署应用。

tomcat容器是如何创建servlet类实例？用到了什么原理？

1. 当容器启动时，会读取在webapps目录下所有的web应用中的web.xml文件，然后对 **xml文件进行解析，并读取servlet注册信息。**然后，将每个应用中注册的servlet类都进行加载，并通过 **反射的方式实例化。**（有时候也是在第一次请求时实例化）
2. 在servlet注册时加上1如果为正数，则在一开始就实例化，如果不写或为负数，则第一次请求实例化。

Tomcat工作模式

Tomcat作为servlet容器，有三种工作模式：

- 1、独立的servlet容器，servlet容器是web服务器的一部分；
- 2、进程内的servlet容器，servlet容器是作为web服务器的插件和java容器的实现，web服务器插件在内部地址空间打开一个jvm使得java容器在内部得以运行。反应速度快但伸缩性不足；
- 3、进程外的servlet容器，servlet容器运行于web服务器之外的地址空间，并作为web服务器的插件和java容器实现的结合。反应时间不如进程内但伸缩性和稳定性比进程内优；

进入Tomcat的请求可以根据Tomcat的工作模式分为如下两类：

- Tomcat作为应用程序服务器：请求来自于前端的web服务器，这可能是Apache, IIS, Nginx等；
- Tomcat作为独立服务器：请求来自于web浏览器；

面试时问到Tomcat相关问题的几率并不高，正式因为如此，很多人忽略了对Tomcat相关技能的掌握，下面这一篇文章整理了Tomcat相关的系统架构，介绍了Server、Service、Connector、Container之间的关系，各个模块的功能，可以说把这几个掌握住了，Tomcat相关的面试题你就不会有任何问题了！

另外，在面试的时候你还要有意识无意识的往Tomcat这个地方引，就比如说常见的Spring MVC的执行流程，一个URL的完整调用链路，这些相关的题目你是可以往Tomcat处理请求的这个过程去说的！掌握了Tomcat这些技能，面试官一定会佩服你的！

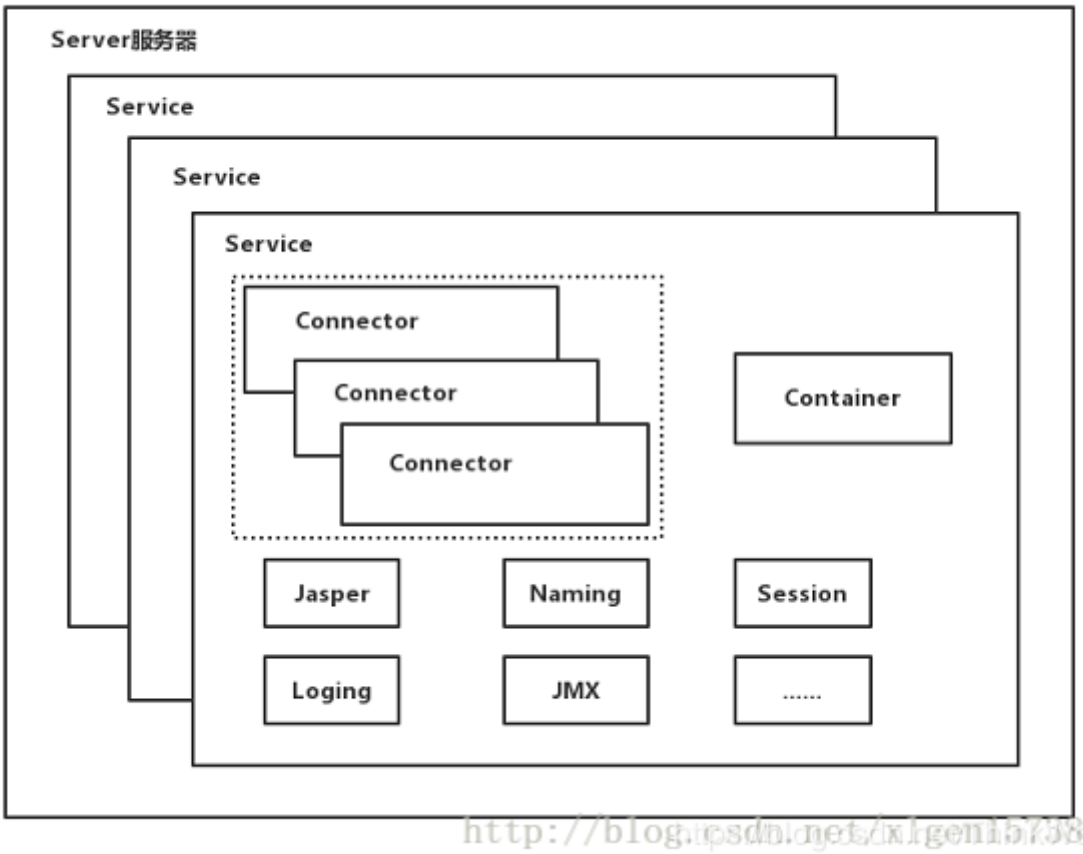
学了本章之后你应该明白的是：

- Server、Service、Connector、Container四大组件之间的关系和联系，以及他们的主要功能点；
- Tomcat执行的整体架构，请求是如何被一步步处理的；
- Engine、Host、Context、Wrapper相关的概念关系；
- Container是如何处理请求的；
- Tomcat用到的相关设计模式；

Tomcat顶层架构

俗话说，站在巨人的肩膀上看世界，一般学习的时候也是先总览一下整体，然后逐个部分个个击破，最后形成思路，了解具体细节，Tomcat的结构很复杂，但是 Tomcat 非常的模块化，找到了 Tomcat 最核心的模块，问题才可以游刃而解，了解了 Tomcat 的整体架构对以后深入了解 Tomcat 来说至关重要！

先上一张Tomcat的顶层结构图（图A），如下：



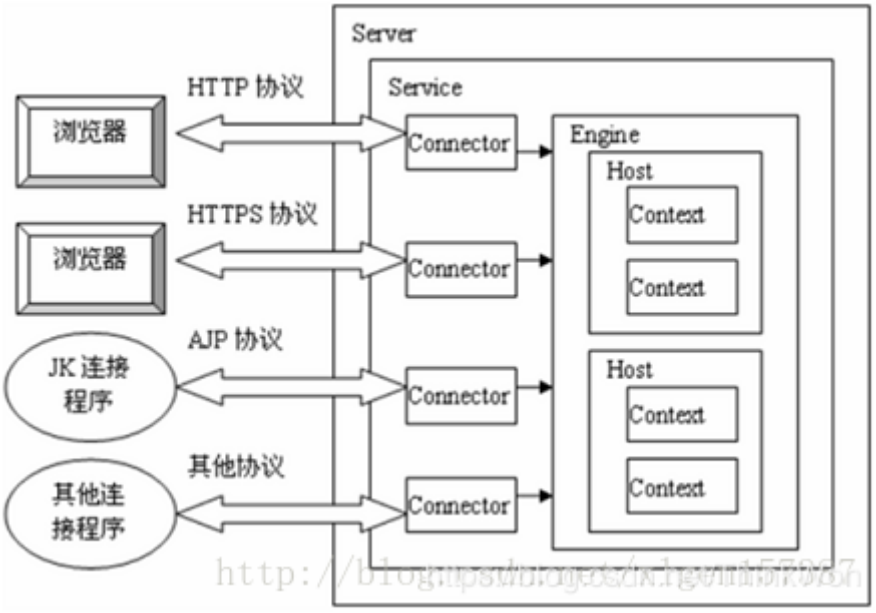
在这里插入图片描述

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，即可以包含多个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：

- Connector用于处理连接相关的事情，并提供Socket与Request请求和Response响应相关的转化;
- 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下面会说到）：



在这里插入图片描述

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。

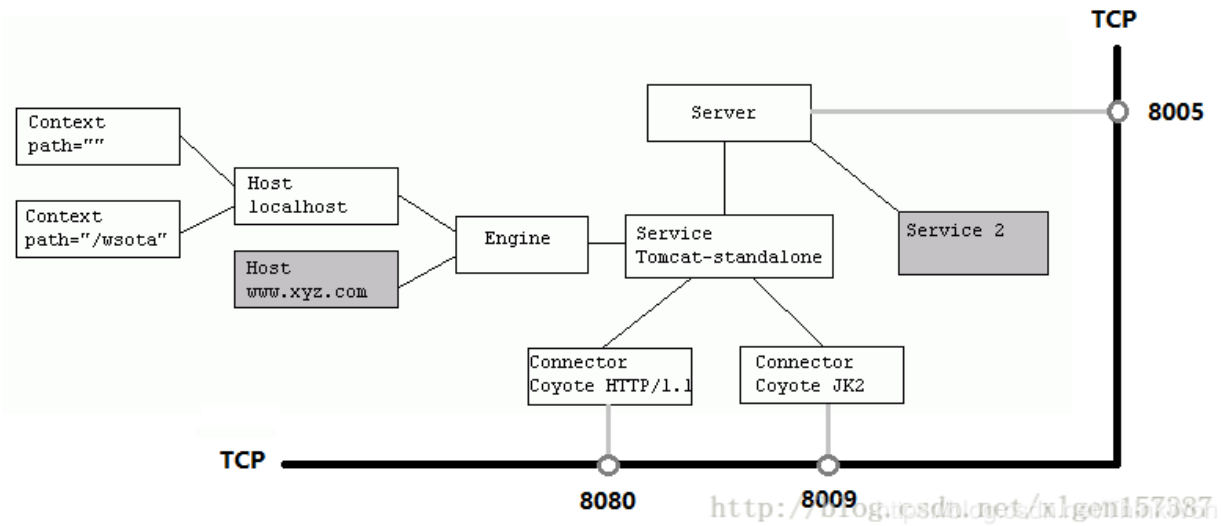
另外，上述的包含关系或者说是父子关系，都可以在tomcat的conf目录下的server.xml配置文件中看出，下图是删除了注释内容之后的一个完整的server.xml配置文件（Tomcat版本为8.0）

```xml
<?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>
```

在这里插入图片描述

详细的配置文件内容可以到Tomcat官网查看：[Tomcat配置文件]

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：



在这里插入图片描述

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Connector，Service左边的内容都属于Container的，Service下边是Connector。

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；
3. Service 是对外提供服务的；
4. Connector用于接受请求并将请求封装成Request和Response来具体处理；
5. 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。

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既然需要处理请求，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！

Connector架构分析

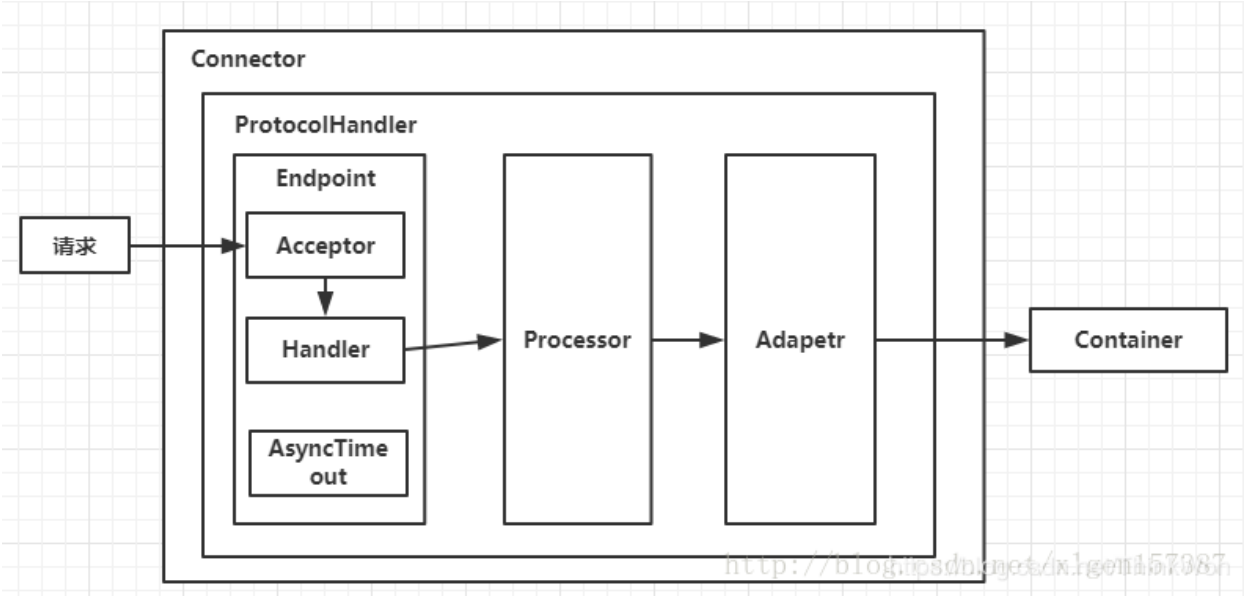
Connector用于接受请求并将请求封装成Request和Response，然后交给Container进行处理，Container处理完之后在交给Connector返回给客户端。

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：



在这里插入图片描述

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。

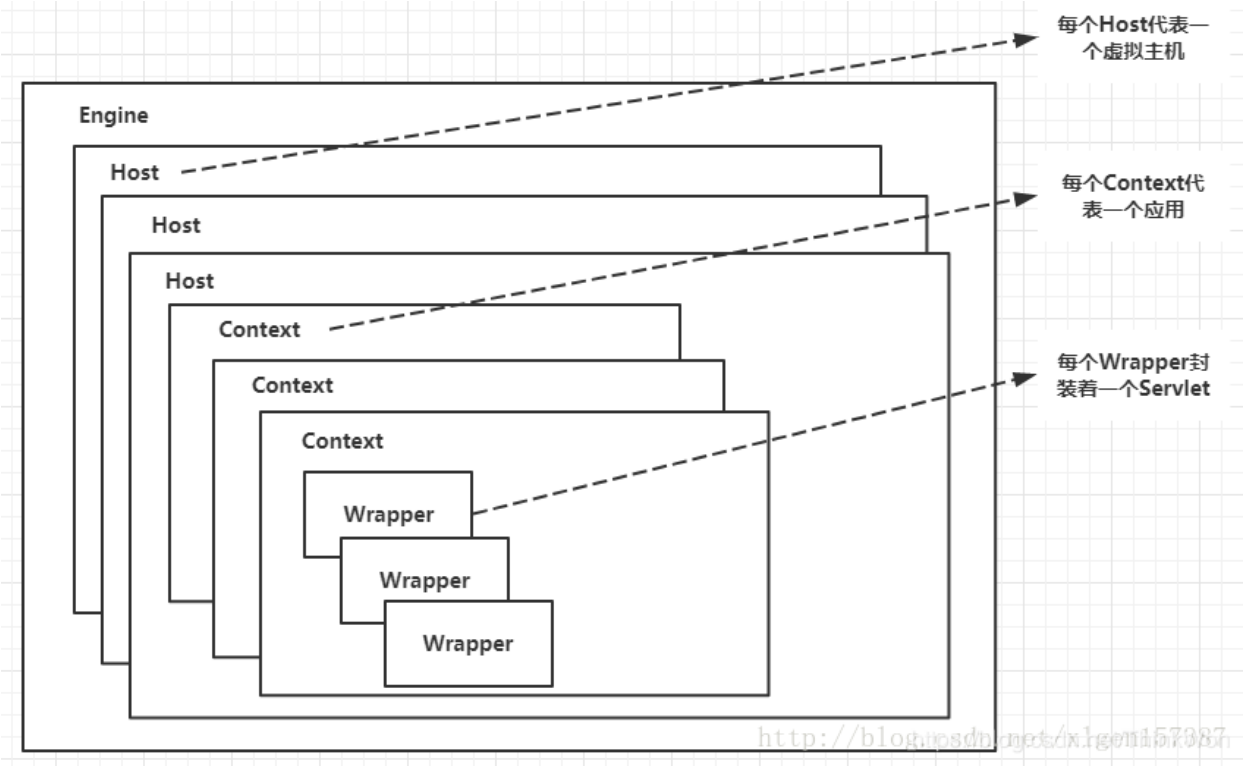
其中ProtocolHandler由包含了三个部件：Endpoint、Processor、Adapter。

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？

Container架构分析

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

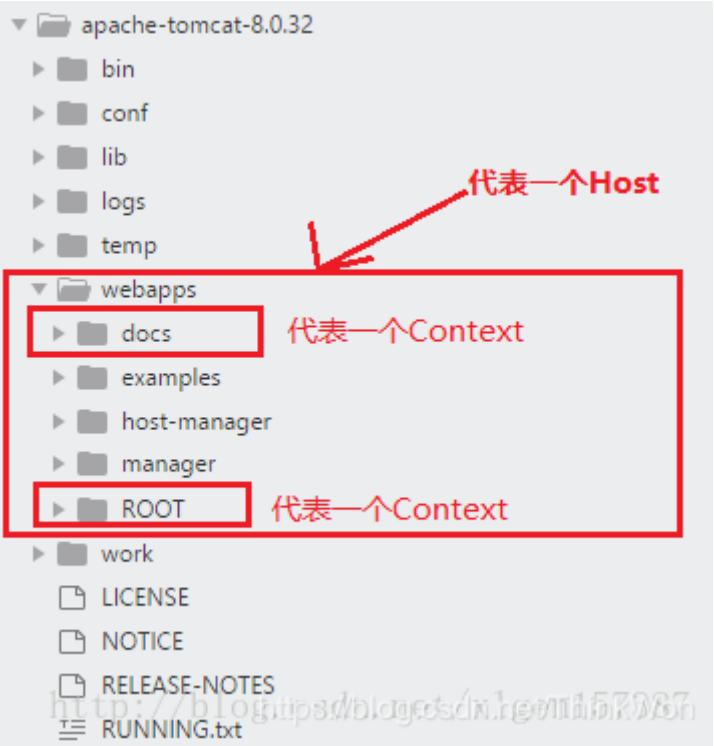


在这里插入图片描述

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；

下面找一个Tomcat的文件目录对照一下，如下图所示：



在这里插入图片描述

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。

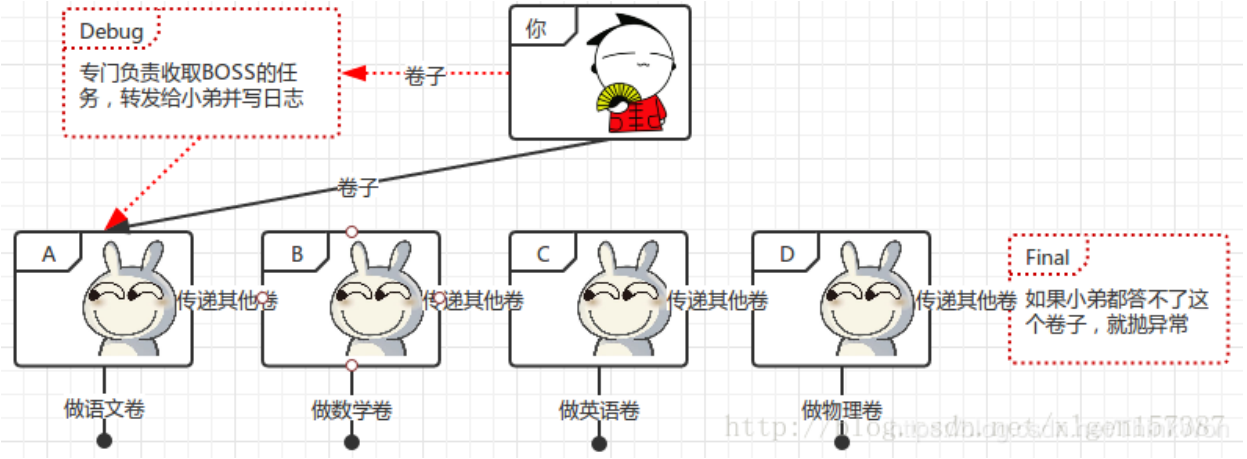
我们访问应用Context的时候，如果是ROOT下的则直接使用域名就可以访问，例如：www.baidu.com，如果是Host（webapps）下的其他应用，则可以使用www.baidu.com/docs进行访问，当然默认指定的根应用（ROOT）是可以进行设定的，只不过Host站点下默认的主应用是ROOT目录下的。

看到这里我们知道Container是什么，但是还是不知道Container是如何进行请求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！

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结果返回，再让下一个处理者继续处理。



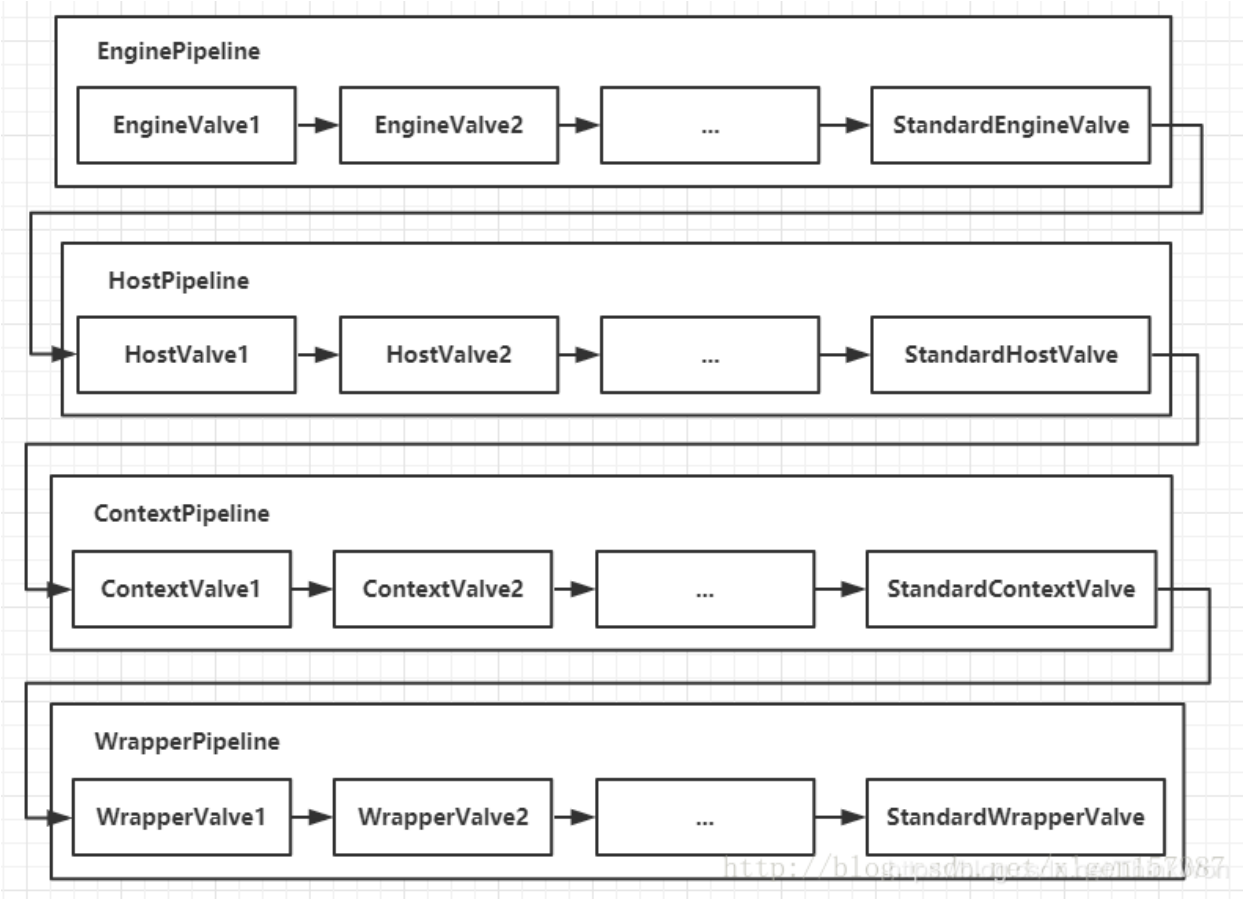
在这里插入图片描述

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：

- 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；
- 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）：



在这里插入图片描述

- 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）；
- 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。
- 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！
- 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。

总结

至此，我们已经对Tomcat的整体架构有了大致的了解，从图A、B、C、D可以看出来每一个组件的基本要素和作用。我们在脑海里应该有一个大概的轮廓了！如果你面试的时候，让你简单的聊一下Tomcat，上面的内容你能脱口而出吗？当你能够脱口而出的时候，面试官一定会对你刮目相看的！