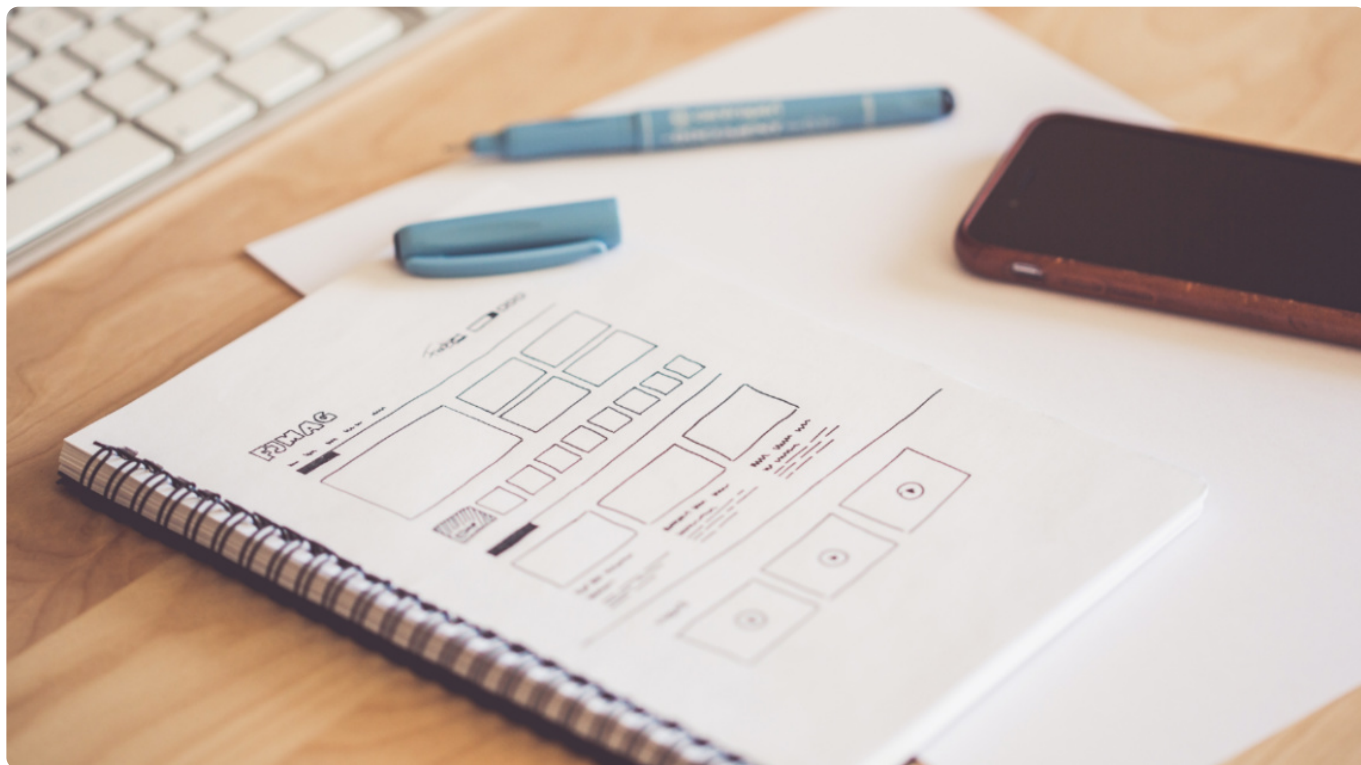


06 | Tomcat系统架构（下）：聊聊多层容器的设计

2019-05-23 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)

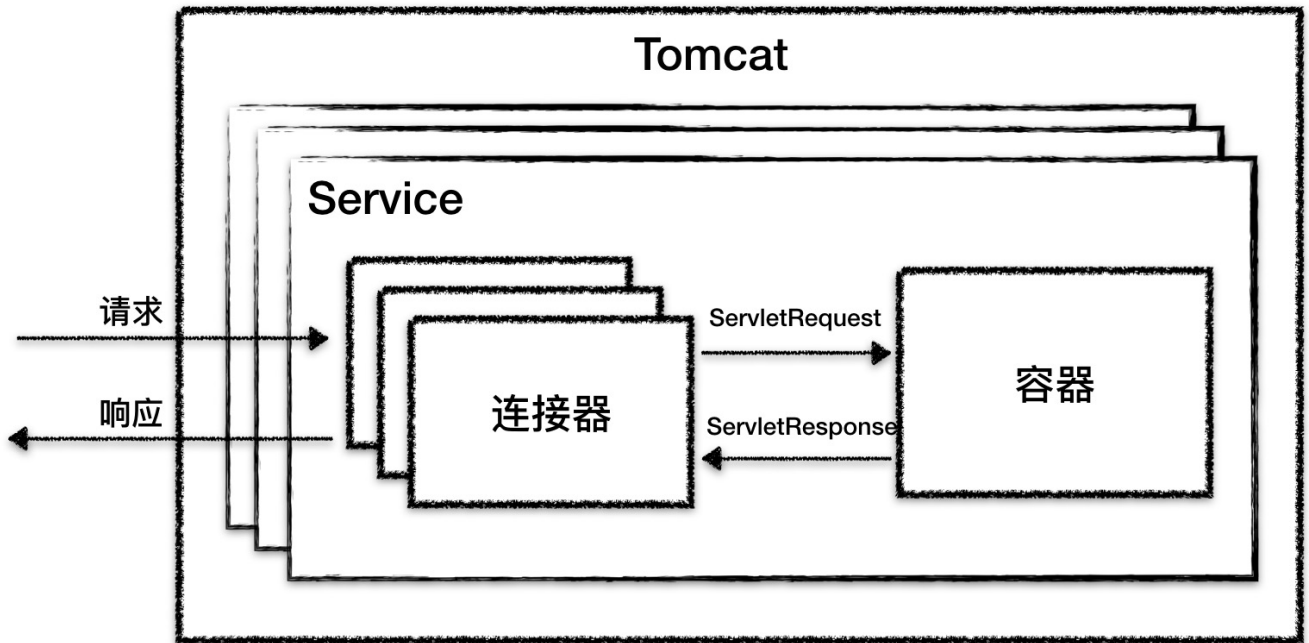


讲述：李号双

时长 11:00 大小 10.08M



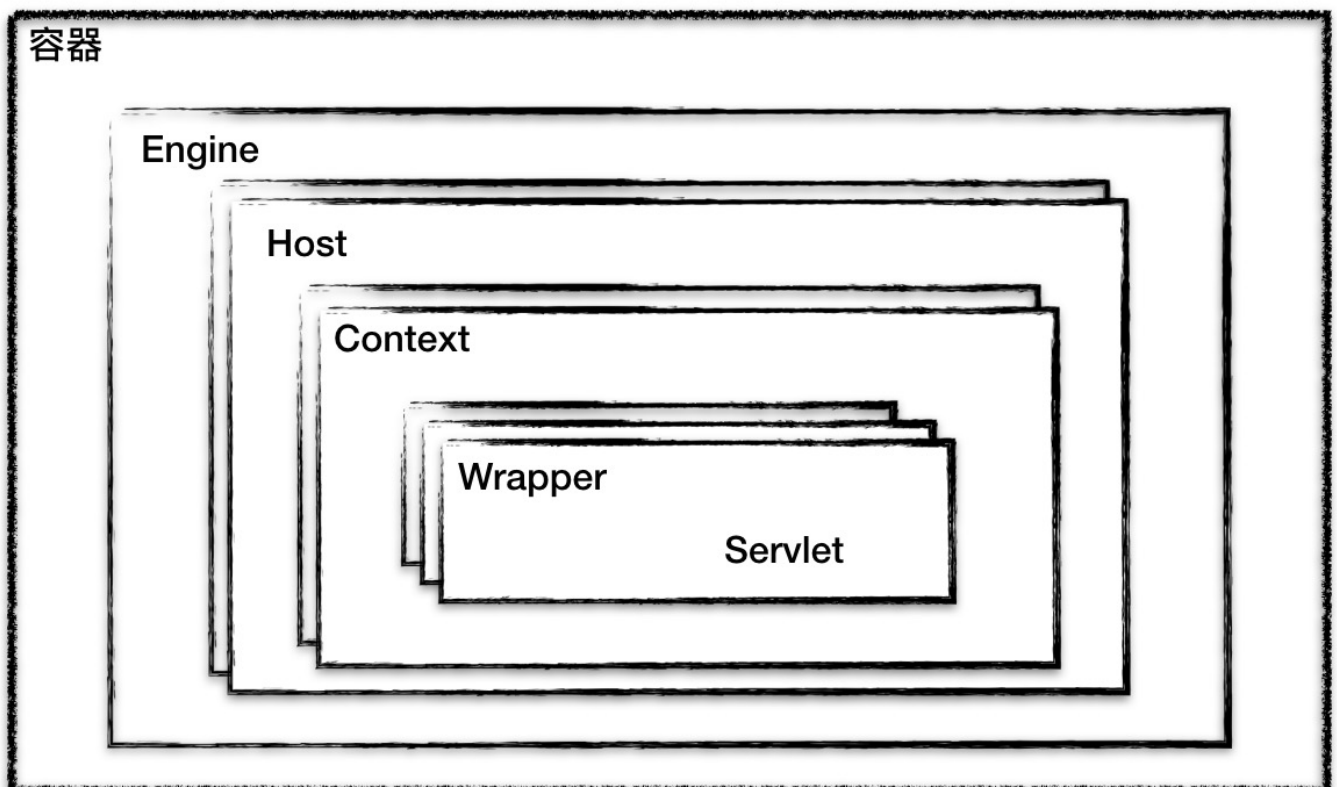
专栏上一期我们学完了连接器的设计，今天我们一起来看一下 Tomcat 的容器设计。先复习一下，上期我讲到了 Tomcat 有两个核心组件：连接器和容器，其中连接器负责外部交流，容器负责内部处理。具体来说就是，连接器处理 Socket 通信和应用层协议的解析，得到 Servlet 请求；而容器则负责处理 Servlet 请求。我们通过下面这张图来回忆一下。



容器，顾名思义就是用来装载东西的器具，在 Tomcat 里，容器就是用来装载 Servlet 的。那 Tomcat 的 Servlet 容器是如何设计的呢？

容器的层次结构

Tomcat 设计了 4 种容器，分别是 Engine、Host、Context 和 Wrapper。这 4 种容器不是平行关系，而是父子关系。下面我画了一张图帮你理解它们的关系。



你可能会问，为什么要设计成这么多层次的容器，这不是增加了复杂度吗？其实这背后的考虑是，**Tomcat 通过一种分层的架构，使得 Servlet 容器具有很好的灵活性。**

Context 表示一个 Web 应用程序；Wrapper 表示一个 Servlet，一个 Web 应用程序中可能会有多个 Servlet；Host 代表的是一个虚拟主机，或者说一个站点，可以给 Tomcat 配置多个虚拟主机地址，而一个虚拟主机下可以部署多个 Web 应用程序；Engine 表示引擎，用来管理多个虚拟站点，一个 Service 最多只能有一个 Engine。

你可以通过 Tomcat 的 server.xml 配置文件来加深对 Tomcat 容器的理解。Tomcat 采用了组件化的设计，它的构成组件都是可配置的，其中最外层的是 Server，其他组件按照一定的格式要求配置在这个顶层容器中。

```
<Server>                                //顶层组件，可以包括多个Service
  <Service>                              //顶层组件，可包含一个Engine，多个连接器
    <Connector>                          //连接器组件，代表通信接口
  </Connector>
  <Engine>                              //容器组件，一个Engine组件处理Service中的所有请求，包含多个Host
    <Host>                              //容器组件，处理特定的Host下客户请求，可包含多个Context
      <Context>                          //容器组件，为特定的Web应用处理所有的客户请求
    </Context>
    </Host>
  </Engine>
</Service>
</Server>
```

那么，Tomcat 是怎么管理这些容器的呢？你会发现这些容器具有父子关系，形成一个树形结构，你可能马上就想到了设计模式中的组合模式。没错，Tomcat 就是用组合模式来管理这些容器的。具体实现方法是，所有容器组件都实现了 Container 接口，因此组合模式可以使得用户对单容器对象和组合容器对象的使用具有一致性。这里单容器对象指的是最底层的 Wrapper，组合容器对象指的是上面的 Context、Host 或者 Engine。Container 接口定义如下：

复制代码

```
1 public interface Container extends Lifecycle {
2     public void setName(String name);
3     public Container getParent();
4     public void setParent(Container container);
5     public void addChild(Container child);
```

```
6     public void removeChild(Container child);
7     public Container findChild(String name);
8 }
```

正如我们期望的那样，我们在上面的接口看到了 `getParent`、`SetParent`、`addChild` 和 `removeChild` 等方法。你可能还注意到 `Container` 接口扩展了 `Lifecycle` 接口，`Lifecycle` 接口用来统一管理各组件的生命周期，后面我也用专门的篇幅去详细介绍。

请求定位 Servlet 的过程

你可能好奇，设计了这么多层次的容器，Tomcat 是怎么确定请求是由哪个 Wrapper 容器里的 Servlet 来处理的呢？答案是，Tomcat 是用 Mapper 组件来完成这个任务的。

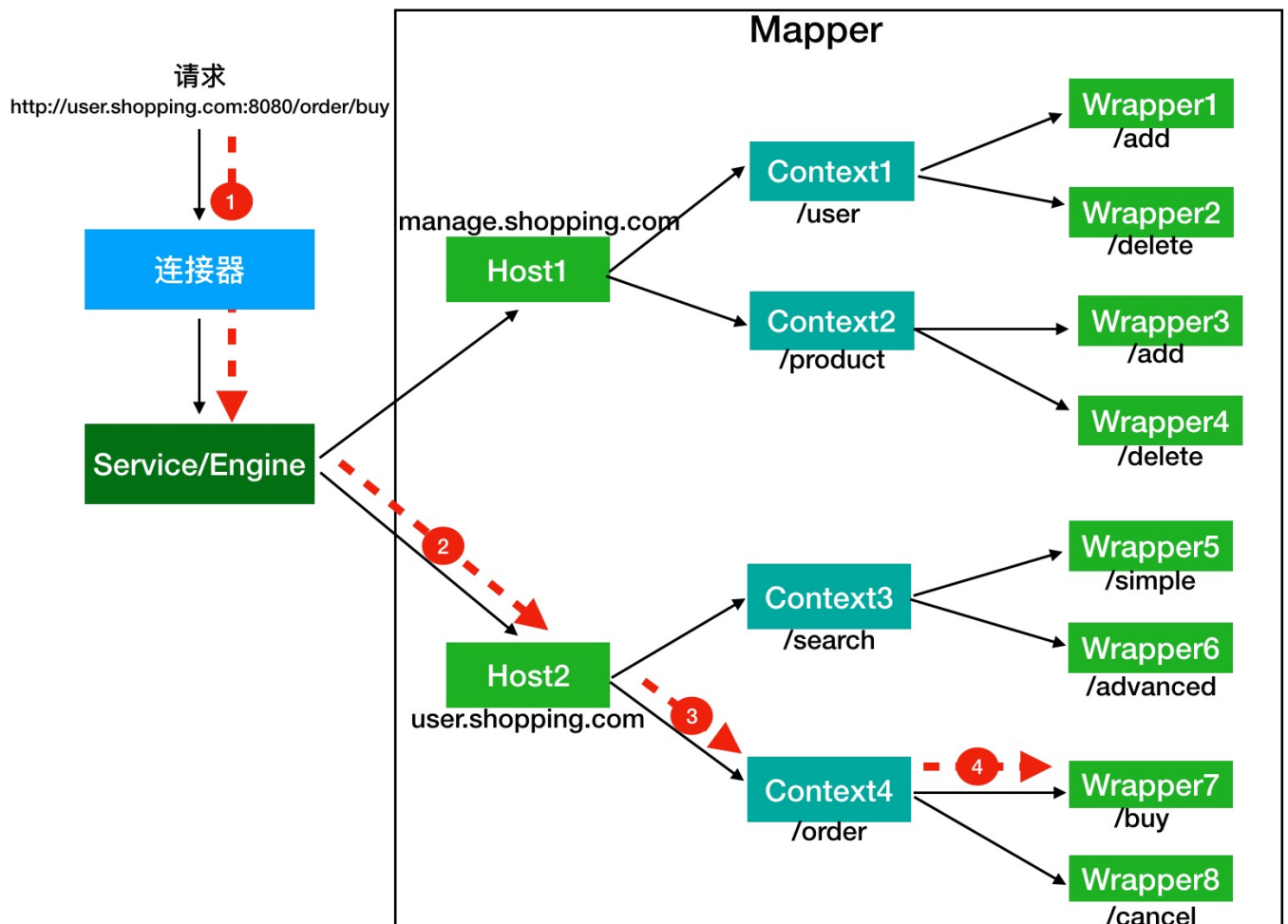
Mapper 组件的功能就是将用户请求的 URL 定位到一个 Servlet，它的工作原理是：Mapper 组件里保存了 Web 应用的配置信息，其实就是**容器组件与访问路径的映射关系**，比如 Host 容器里配置的域名、Context 容器里的 Web 应用路径，以及 Wrapper 容器里 Servlet 映射的路径，你可以想象这些配置信息就是一个多层次的 Map。

当一个请求到来时，Mapper 组件通过解析请求 URL 里的域名和路径，再到自己保存的 Map 里去查找，就能定位到一个 Servlet。请你注意，一个请求 URL 最后只会定位到一个 Wrapper 容器，也就是一个 Servlet。

读到这里你可能感到有些抽象，接下来我通过一个例子来解释这个定位的过程。

假如有一个网购系统，有面向网站管理人员的后台管理系统，还有面向终端客户的在线购物系统。这两个系统跑在同一个 Tomcat 上，为了隔离它们的访问域名，配置了两个虚拟域名：`manage.shopping.com`和`user.shopping.com`，网站管理人员通过`manage.shopping.com`域名访问 Tomcat 去管理用户和商品，而用户管理和商品管理是两个单独的 Web 应用。终端客户通过`user.shopping.com`域名去搜索商品和下订单，搜索功能和订单管理也是两个独立的 Web 应用。

针对这样的部署，Tomcat 会创建一个 Service 组件和一个 Engine 容器组件，在 Engine 容器下创建两个 Host 子容器，在每个 Host 容器下创建两个 Context 子容器。由于一个 Web 应用通常有多个 Servlet，Tomcat 还会在每个 Context 容器里创建多个 Wrapper 子容器。每个容器都有对应的访问路径，你可以通过下面这张图来帮助你理解。



假如有用户访问一个 URL，比如图中的

`http://user.shopping.com:8080/order/buy`，Tomcat 如何将这个 URL 定位到一个 Servlet 呢？

首先，根据协议和端口选定 Service 和 Engine。

我们知道 Tomcat 的每个连接器都监听不同的端口，比如 Tomcat 默认的 HTTP 连接器监听 8080 端口、默认的 AJP 连接器监听 8009 端口。上面例子中的 URL 访问的是 8080 端口，因此这个请求会被 HTTP 连接器接收，而一个连接器是属于一个 Service 组件的，这样 Service 组件就确定了。我们还知道一个 Service 组件里除了有多个连接器，还有一个容器组件，具体来说就是一个 Engine 容器，因此 Service 确定了也就意味着 Engine 也确定了。

然后，根据域名选定 Host。

Service 和 Engine 确定后，Mapper 组件通过 URL 中的域名去查找相应的 Host 容器，比如例子中的 URL 访问的域名是 `user.shopping.com`，因此 Mapper 会找到 Host2 这个

容器。

之后，根据 URL 路径找到 Context 组件。

Host 确定以后，Mapper 根据 URL 的路径来匹配相应的 Web 应用的路径，比如例子中访问的是 /order，因此找到了 Context4 这个 Context 容器。


最后，根据 URL 路径找到 Wrapper (Servlet) 。

Context 确定后，Mapper 再根据 web.xml 中配置的 Servlet 映射路径来找到具体的 Wrapper 和 Servlet。

看到这里，我想你应该已经了解了什么是容器，以及 Tomcat 如何通过一层一层的父子容器找到某个 Servlet 来处理请求。需要注意的是，并不是说只有 Servlet 才会去处理请求，实际上这个查找路径上的父子容器都会对请求做一些处理。我在上一期说过，连接器中的 Adapter 会调用容器的 Service 方法来执行 Servlet，最先拿到请求的是 Engine 容器，Engine 容器对请求做一些处理后，会把请求传给自己子容器 Host 继续处理，依次类推，最后这个请求会传给 Wrapper 容器，Wrapper 会调用最终的 Servlet 来处理。那么这个调用过程具体是怎么实现的呢？答案是使用 Pipeline-Valve 管道。


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，处理完之后将再调用下一个处理者继续处理。

Valve 表示一个处理点，比如权限认证和记录日志。如果你还不太理解的话，可以来看看 Valve 和 Pipeline 接口中的关键方法。

 复制代码

```
1 public interface Valve {
2     public Valve getNext();
3     public void setNext(Valve valve);
4     public void invoke(Request request, Response response)
5 }
```

由于 Valve 是一个处理点，因此 invoke 方法就是来处理请求的。注意到 Valve 中有 getNext 和 setNext 方法，因此我们大概可以猜到有一个链表将 Valve 链起来了。请你继续看 Pipeline 接口：

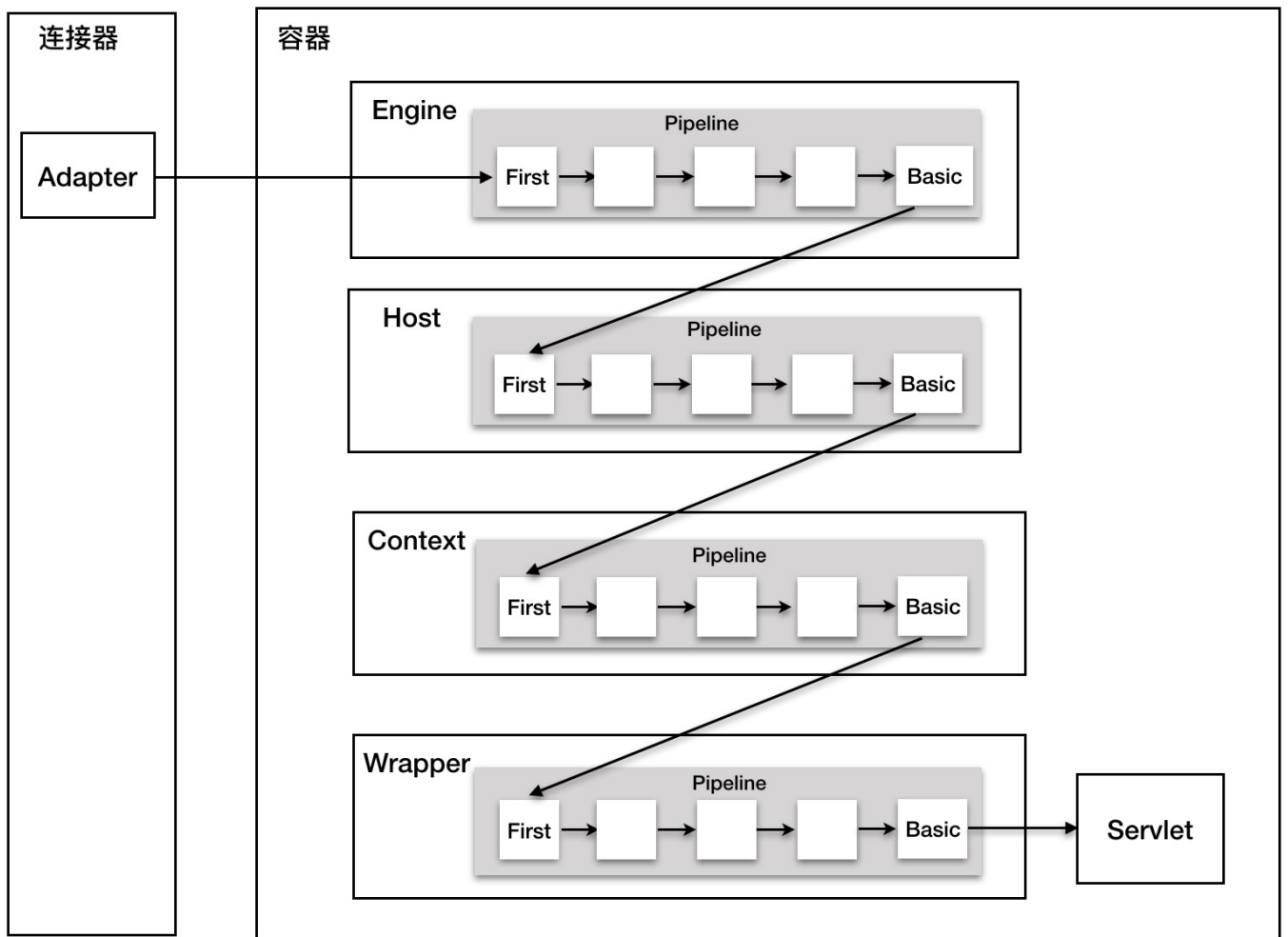
 复制代码

```
1 public interface Pipeline extends Contained {
2     public void addValve(Valve valve);
3     public Valve getBasic();
4     public void setBasic(Valve valve);
5     public Valve getFirst();
6 }
```

没错，Pipeline 中有 addValve 方法。Pipeline 中维护了 Valve 链表，Valve 可以插入到 Pipeline 中，对请求做某些处理。我们还发现 Pipeline 中没有 invoke 方法，因为整个调用链的触发是 Valve 来完成的，Valve 完成自己的处理后，调用 getNext.invoke() 来触发下一个 Valve 调用。

每一个容器都有一个 Pipeline 对象，只要触发这个 Pipeline 的第一个 Valve，这个容器里 Pipeline 中的 Valve 就都会被调用到。但是，不同容器的 Pipeline 是怎么链式触发的呢，比如 Engine 中 Pipeline 需要调用下层容器 Host 中的 Pipeline。

这是因为 Pipeline 中还有个 getBasic 方法。这个 BasicValve 处于 Valve 链表的末端，它是 Pipeline 中必不可少的一个 Valve，负责调用下层容器的 Pipeline 里的第一个 Valve。我还是通过一张图来解释。



整个调用过程由连接器中的 Adapter 触发的，它会调用 Engine 的第一个 Valve：

复制代码

```
1 // Calling the container
2 connector.getService().getContainer().getPipeline().getFirst().invoke(request, response
```

Wrapper 容器的最后一个 Valve 会创建一个 Filter 链，并调用 doFilter() 方法，最终会调到 Servlet 的 service 方法。

你可能会问，前面我们不是讲到了 Filter，似乎也有相似的功能，那 Valve 和 Filter 有什么区别吗？它们的区别是：

Valve 是 Tomcat 的私有机制，与 Tomcat 的基础架构 /API 是紧耦合的。Servlet API 是公有的标准，所有的 Web 容器包括 Jetty 都支持 Filter 机制。

另一个重要的区别是 Valve 工作在 Web 容器级别，拦截所有应用的请求；而 Servlet Filter 工作在应用级别，只能拦截某个 Web 应用的所有请求。如果想做整个 Web 容器的拦截器，必须通过 Valve 来实现。

本期精华

今天我们学习了 Tomcat 容器的层次结构、根据请求定位 Servlet 的过程，以及请求在容器中的调用过程。Tomcat 设计了多层容器是为了灵活性的考虑，灵活性具体体现在一个 Tomcat 实例（Server）可以有多个 Service，每个 Service 通过多个连接器监听不同的端口，而一个 Service 又可以支持多个虚拟主机。一个 URL 网址可以用不同的主机名、不同的端口和不同的路径来访问特定的 Servlet 实例。

请求的链式调用是基于 Pipeline-Valve 责任链来完成的，这样的设计使得系统具有良好的可扩展性，如果需要扩展容器本身的功能，只需要增加相应的 Valve 即可。

课后思考

Tomcat 内的 Context 组件跟 Servlet 规范中的 ServletContext 接口有什么区别？跟 Spring 中的 ApplicationContext 又有什么关系？

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，也欢迎你把你的课后思考和心得记录下来，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。

深入拆解 Tomcat & Jetty

从源码角度深度探索 Java 中间件

李号双

eBay 技术主管



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | Tomcat系统架构（上）：连接器是如何设计的？

下一篇 07 | Tomcat如何实现一键式启停？

精选留言 (52)

写留言



一路远行

2019-05-23

45

1) Servlet规范中ServletContext表示web应用的上下文环境，而web应用对应tomcat的概念是Context，所以从设计上，ServletContext自然会成为tomcat的Context具体实现的一个成员变量。

2) tomcat内部实现也是这样完成的，ServletContext对应tomcat实现是...

展开 ▾

作者回复:





阿旺

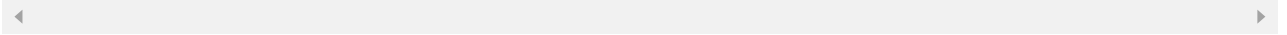
2019-05-23

👍 5

你好 请问到业务的controller是从哪部分进去的呢 谢谢

展开 ▾

作者回复: Wrapper -> Filter -> DispatcherServlet -> Controller



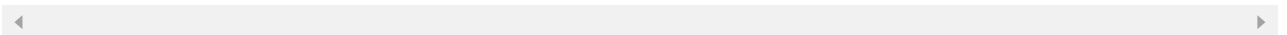
yhh

2019-05-24

👍 3

Basic value 有些疑惑。比如engine容器下有多个host容器，那engine容器的basic value 是怎么知道要指向哪个host容器的value呢？

作者回复: 好问题，Mapper组件在映射请求的时候，会在Request对象中存储相应的Host、Context等对象，这些选定的容器用来处理这个特定的请求，因此Engine中的Valve是从Request对象拿到Host容器的。



why

2019-05-28

👍 2

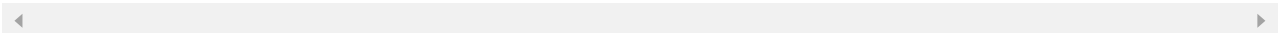
- 容器的层次结构

- Engine, Host, Context, Wrapper, 是嵌套关系
- 一个 Tomcat 实例包含一个 Engine, 一个 Engine 包含多个 Host, 以此类推
- 容器采用组合模式, 所有容器组件都实现 Container 接口, 保证使用一致性

- 定位 Servlet 的过程...

展开 ▾

作者回复: 📖



李海明

2019-05-27

👍 2

老师，能提供一份tomcat多host的配置吗？

展开 ▾

作者回复: <server port= "8005" shutdown= "SHUTDOWN" >
<service name= "Catalina" >
<engine defaultHost= "localhost" name= "Catalina" >
<host appbase= "webapps" autoDeploy= "true" name= "localhost"
unpackWARs= "true" ></host>
<host appbase= "webapps1" autoDeploy= "true" name= "www.domain1.com"
unpackWARs= "true" ></host>
<host appbase= "webapps2" autoDeploy= "true" name= "www.domain2.com"
unpackWARs= "true" ></host>
<host appbase= "webapps3" autoDeploy= "true" name= "www.domain3.com"
unpackWARs= "true" ></host>
</engine>
</service>
</server>



一路远行

2019-05-23

👍 2

这段话的描述有些不准确:

“首先, 根据协议和端口号选定 Service 和 Engine。”

我的理解是, connector配置中只要有端口号就可以确定service和engine, 协议的配置只是为了解析, 对请求的路由决策没有起到作用。

展开 ∨

作者回复: 你理解对了, 这段话的下面就有解释



Joker

2019-05-23

👍 2

从具体的数据上来说, Tomcat的Context里面是整个Web应用的参数环境。
而Servlet规范中的ServletContext接口则既有 Servlet 容器中运行时的参数环境。
从层级上来说, 一个是对应Web应用的, 一个则指对应单个Servlet的; 一个web应用中可以有多个Servlet。

ServletContext负责的是servlet运行环境上下信息, 不管session管理, 不管servlet的加...

展开 ∨



永光

2019-06-04

1

老师您好，

你看这样理解对吗？，一个Engine可以对应多个Host，一个Host下面可以放多个应用。

问题：

1、如果我有两个应用需要部署，现在就可以有很多种方案了。

部署在同一个service下（同一个Engine）的同一个Host目录下...

展开

作者回复: 如果你需要隔离访问域名，用第二种；如果要进一步隔离访问端口，用第三种。



发条橙子 ...

2019-05-26

1

老师， 我之前一直误以为一个 server表示我们一个应用(实际上只是代表一个tomcat实例)，所以一直不理解为什么 server 下通过 host 可以配置多个应用，学了这节，发现是一个context对应一个应用，自己百度了一下，原来可以通过 host 或者 service 来让 tomcat 访问不同的目录来访问多个应用。

...

展开

作者回复: 1，你说的对，在同一个Tomcat实例里部署多个Web应用是为了节省内存等资源，不过配置部署有点复杂，应用之间互相影响，加上现在硬件成本将低，多应用部署比较少见了。

2，Servlet接口中定义了service方法，没有doGet/doPost。HttpServlet是一个实现类，实现了service方法，同时留出了doGet/doPost方法让程序员来实现。

你可以通过web.xml配置一个或多个Filter，Servlet容器在调用Servlet的service之前，需要调用这些Filter，于是把这些Filter创建出来，形成链表，依次调用，这个Filter链中的最后一个Filter会负责调用Servlet的service方法。

通过一个调用栈来理解一下：

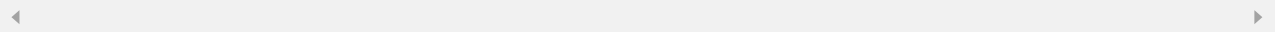
doGet:22, HelloServlet (servlet)

service:635, HttpServlet (javax.servlet.http)

service:742, HttpServlet (javax.servlet.http)

internalDoFilter:231, ApplicationFilterChain (org.apache.catalina.core)

doFilter:166, ApplicationFilterChain (org.apache.catalina.core)
invoke:199, StandardWrapperValve (org.apache.catalina.core)
invoke:96, StandardContextValve (org.apache.catalina.core)
invoke:493, AuthenticatorBase (org.apache.catalina.authenticator)
invoke:140, StandardHostValve (org.apache.catalina.core)
invoke:81, ErrorReportValve (org.apache.catalina.valves)
invoke:87, StandardEngineValve (org.apache.catalina.core)
service:342, CoyoteAdapter (org.apache.catalina.connector)
service:800, Http11Processor (org.apache.coyote.http11)
process:66, AbstractProcessorLight (org.apache.coyote)
process:806, AbstractProtocol\$ConnectionHandler (org.apache.coyote)
doRun:1498, NioEndpoint\$SocketProcessor (org.apache.tomcat.util.net)
run:49, SocketProcessorBase (org.apache.tomcat.util.net)
runWorker:1149, ThreadPoolExecutor (java.util.concurrent)
run:624, ThreadPoolExecutor\$Worker (java.util.concurrent)
run:61, TaskThread\$WrappingRunnable (org.apache.tomcat.util.threads)
run:748, Thread (java.lang)



z.l

2019-05-24

👍 1

既然tomcat设计上就支持多应用部署，为什么感觉生产环境下都是一个tomcat只部署一个应用，而且到了springboot还把tomcat内置到应用里了。



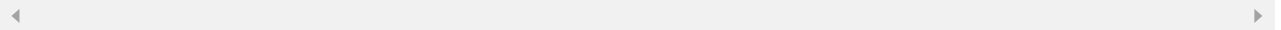
Monday

2019-05-24

👍 1

老师，tomcat容器的四层架构是一步一步演化过来的还是一步到位的？如果是演化过来的，是哪些故事推动他们如此演化的？谢谢

作者回复: Tomca的容器架构是一步到位的，设计之初就考虑到了使用的灵活性，如果说将来有变化，应该会朝着“瘦身”的方向发展。



allea

2019-05-24

👍 1

收获很多，谢谢老师。有几个问题：

1. Wrapper容器里有且只有一个Servlet，Context里可以有多个Servlet即可以有多个Wrapper，这样理解对吗？
2. Pipeline负责维护链式Vavle，具体Vavle负责处理具体请求？

作者回复: 1， 对的

2， Vavle可以理解对请求进行“拦截”和“修正”，类似Filter，但是Valve用来扩展容器本身的功能的，真正处理请求并产生响应的是Servlet。



allean

2019-05-23

👍 1

原文：“所有容器都实现了Container接口，因此组合模式可以使得用户对单容器和组合容器的对象使用具有一致性”

问题：这段话不太理解，这里说的一致性是指什么，麻烦老师指明🙏

展开 ∨

作者回复: 可以这样理解，我们可以往“组合容器”里添加其他容器，但是不能往“单容器”里去添加，是树枝节点和叶子节点的区别，但是“组合模式”的目的就是要对用户屏蔽这些差异，对用户看来它们都是一样的，于是定义了Container接口，接口中有add，remove等等方法，这些方法的参数也是Container对象，感觉不出树枝和树叶的区别。



kaiux

2019-05-23

👍 1

一直在用Tomcat，现在终于有机会能剥开它仔细看一看了，讲的很棒，醍醐灌顶👍



Cy190622

2019-05-23

👍 1

Tomcat中的Context组件是Web的应用程序，也就是Servlet(在具体代码中是Wrapper)运行的容器，而ServletContext是定义了Servlet的规范，在spring中ApplicationContext实现了ServletContext。我认为是Context组件为ApplicationContext提供了上下文环境。我所理解的url路由过程：

1. 首先根据URL在Mapper组件中找到Servlet的映射关系，应为在初始化Servlet过程中...

展开 ∨

作者回复: Spring没有实现ServletContext接口, 是Tomcat实现了这个接口, Spring可以拿到通过Request对象拿到ServletContext。

你可以在server.xml中配置server和service组件。

后面会谈Tomcat是如何实现Listener和Filter的



-W.LI-

2019-05-23

👍 1

老师好!Tomcat, server.xml还是搞不太清楚。最外层是server, 一个server对应一个Tomcat实例, server下面可以由很多service组件, service组件是区分协议和端口号(问题1:域名访问的话请求到达这里是已经被解析成port了吗? 如果已经解析了后面为啥还能拿到二级域名)。每个service对应多个连接器和一个engine。(问题2:为啥要设置多个连接器对应一个engine, 是为了提高连接器接受请求的并发吗?)。engine下面对应host, hos...
展开 ∨

作者回复: 1, DNS解析是在客户端完成的, 你需要在客户端把两个域名指向同一个IP, 可以通过hosts配置, 因此只要你使用的端口相同, 两个域名其实访问的是同一个Service组件。而Tomcat设计通过请求URL中Host来转发到不同Host组件的。

2. 不同的连接器代表一种通信的路径, 比如同时支持HTTP和HTTPS, 不是为了并发。

3. 对

4.域名相同指向不同的Web应用, 默认情况下你只需要把你的Web应用全部扔到webapps目录下, Tomcat自动检测, 创建相应的Context组件, 各个web应用的访问路径就是web应用的目录名。



夏天

2019-05-23

👍 1

一个tomcat实例下能部署多个工程上去, 每个工程下有多个servlet路径, 分别也就是context和warpper组件控制, 多个host组件目前还没遇到过, 能出现多个域名情况
展开 ∨



天天向上

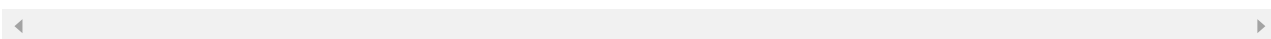
2019-05-23

👍 1

Connector和Engine是平级的，并且 Connector可以有多个 容器结构图 xml结构表示的有问题吧

作者回复: 是的，谢谢指出，已经在紧急修复了，正确的关系是这样的：

```
<Server>
  <Service>
    <Connector>
  </Connector>
  <Engine>
    <Host>
      <Context>
    </Context>
    </Host>
  </Engine>
</Service>
</Server>
```



Joker

2019-05-23

👍 1

而Spring中的ApplicationContext则负责管理是Spring容器中的bean，和Tomcat的Context的数据有着本质的区别。



永光

2019-06-05

👍

老师你好，你在回答中Pipeline-value这个机制的详细流程是怎样时提到：“在Tomcat启动的时候，这些Valve都已经设置好了。通过addValve方法向Pipeline中添加Valve，最新添加的总是成为‘first’。”

问题：当engine容器下有多个host容器，那engine容器的basic value，在Tomcat启动时是怎么设置Value的？ ...

展开 ∨

作者回复: 比如StandardEngine在构造函数中设置自己的BasicValve：

```
public StandardEngine() {
  ...
  pipeline.setBasic(new StandardEngineValve());
}
```

```
...  
  
}
```

StandardHost也在构造函数里设置它的BasicValve:

```
public StandardHost() {  
    ...  
    pipeline.setBasic(new StandardHostValve());  
  
}
```

在StandardHost的启动方法startInternal里, 通过addValve添加其他Valve:

```
Valve valve = (Valve) Class.forName(errorValve).getConstructor().newInstance();  
getPipeline().addValve(valve);
```

