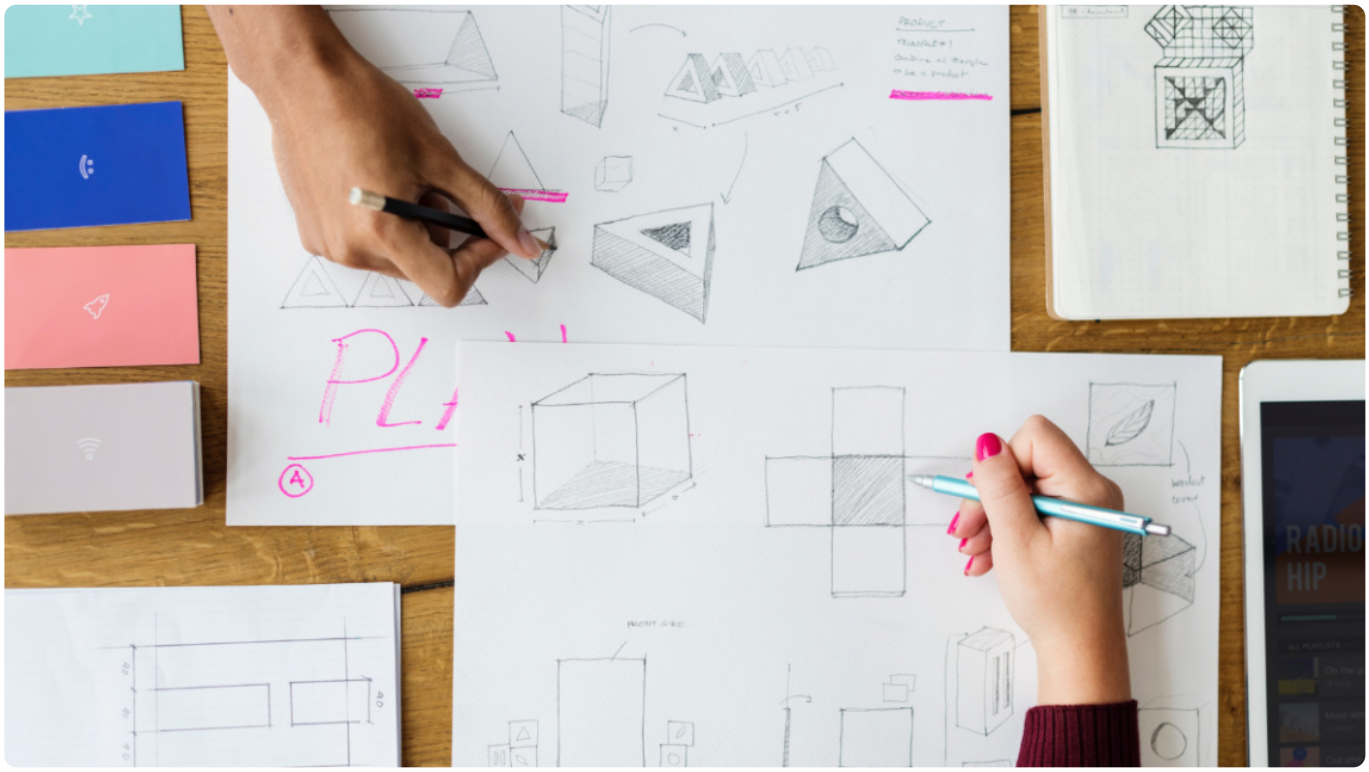
05 | Tomcat系统架构（上）： 连接器是如何设计的?

2019-05-21 李号双

深入拆解Tomcat & Jetty 进入课程 >



讲述：李号双
时长 11:43 大小 10.74M

在面试时我们可能经常被问到：你做的 XX 项目的架构是如何设计的，请讲一下实现的思路。对于面试官来说，可以通过你对复杂系统设计的理解，了解你的技术水平以及处理复杂问题的思路。

今天咱们就来一步一步分析 Tomcat 的设计思路，看看 Tomcat 的设计者们当时是怎么回答这个问题的。一方面我们可以学到 Tomcat 的总体架构，学会从宏观上怎么去设计一个复杂系统，怎么设计顶层模块，以及模块之间的关系；另一方面也为我们深入学习 Tomcat 的工作原理打下基础。

Tomcat 总体架构

我们知道如果要设计一个系统，首先是要了解需求。通过专栏前面的文章，我们已经了解了 Tomcat 要实现 2 个核心功能：

处理 Socket 连接，负责网络字节流与 Request 和 Response 对象的转化。

加载和管理 Servlet，以及具体处理 Request 请求。

因此 Tomcat 设计了两个核心组件连接器（Connector）和容器（Container）来分别做这两件事情。连接器负责对外交流，容器负责内部处理。

所以连接器和容器可以说是 Tomcat 架构里最重要的两部分，需要你花些精力理解清楚。这两部分内容我会分成两期，今天我来分析连接器是如何设计的，下一期我会介绍容器的设计。

在开始讲连接器前，我先铺垫一下 Tomcat 支持的多种 I/O 模型和应用层协议。

Tomcat 支持的 I/O 模型有：

NIO：非阻塞 I/O，采用 Java NIO 类库实现。

NIO2：异步 I/O，采用 JDK 7 最新的 NIO2 类库实现。

APR：采用 Apache 可移植运行库实现，是 C/C++ 编写的本地库。

Tomcat 支持的应用层协议有：

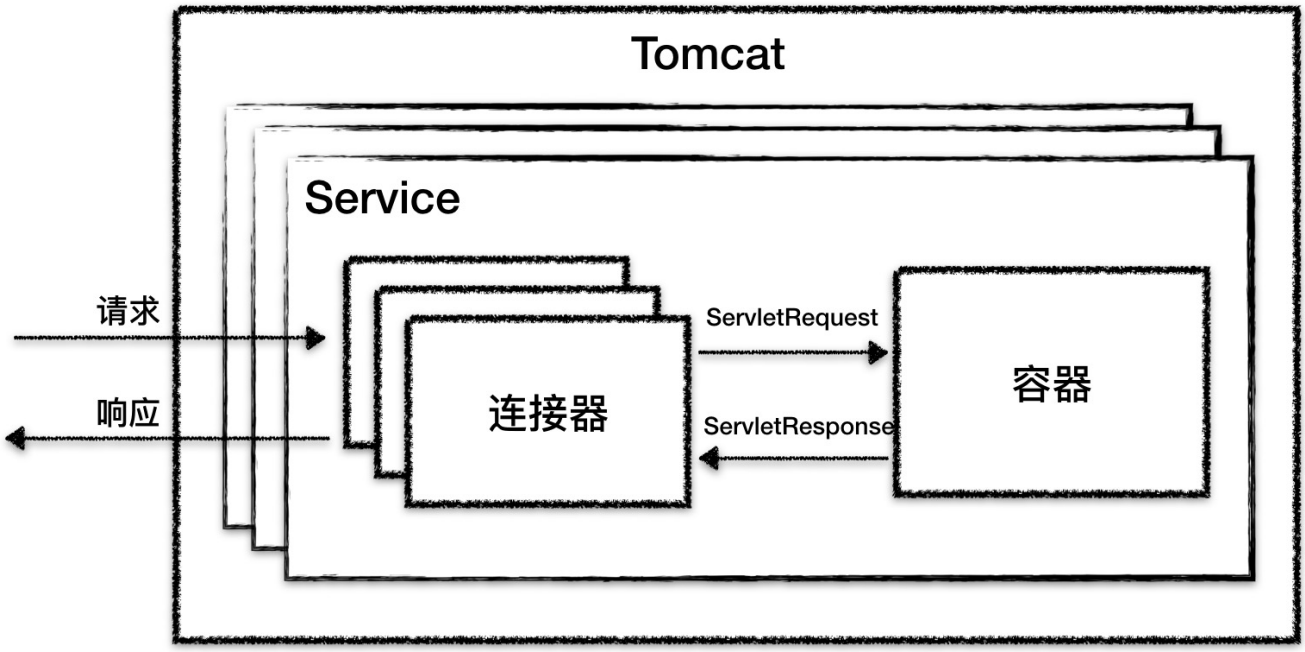
HTTP/1.1：这是大部分 Web 应用采用的访问协议。

AJP：用于和 Web 服务器集成（如 Apache）。

HTTP/2：HTTP 2.0 大幅度的提升了 Web 性能。

Tomcat 为了实现支持多种 I/O 模型和应用层协议，一个容器可能对接多个连接器，就好比一个房间有多个门。但是单独的连接器或者容器都不能对外提供服务，需要把它们组装起来才能工作，组装后这个整体叫作 Service 组件。这里请你注意，Service 本身没有做什么重要的事情，只是在连接器和容器外面多包了一层，把它们组装在一起。Tomcat 内可能有多个 Service，这样的设计也是出于灵活性的考虑。通过在 Tomcat 中配置多个 Service，可以实现通过不同的端口号来访问同一台机器上部署的不同应用。

到此我们得到这样一张关系图：



从图上你可以看到，最顶层是 Server，这里的 Server 指的就是一个 Tomcat 实例。一个 Server 中有一个或者多个 Service，一个 Service 中有多个连接器和一个容器。连接器与容器之间通过标准的 ServletRequest 和 ServletResponse 通信。

连接器

连接器对 Servlet 容器屏蔽了协议及 I/O 模型等的区别，无论是 HTTP 还是 AJP，在容器中获取到的都是一个标准的 ServletRequest 对象。

我们可以把连接器的功能需求进一步细化，比如：

监听网络端口。

接受网络连接请求。

读取请求网络字节流。

根据具体应用层协议（HTTP/AJP）解析字节流，生成统一的 Tomcat Request 对象。

将 Tomcat Request 对象转成标准的 ServletRequest。

调用 Servlet 容器，得到 ServletResponse。

将 ServletResponse 转成 Tomcat Response 对象。

将 Tomcat Response 转成网络字节流。

将响应字节流写回给浏览器。

需求列清楚后，我们要考虑的下一个问题是，连接器应该有哪些子模块？优秀的模块化设计应该考虑**高内聚、低耦合**。

高内聚是指相关度比较高的功能要尽可能集中，不要分散。

低耦合是指两个相关的模块要尽可能减少依赖的部分和降低依赖的程度，不要让两个模块产生强依赖。

通过分析连接器的详细功能列表，我们发现连接器需要完成 3 个**高内聚**的功能：

网络通信。

应用层协议解析。

Tomcat Request/Response 与 ServletRequest/ServletResponse 的转化。

因此 Tomcat 的设计者设计了 3 个组件来实现这 3 个功能，分别是 EndPoint、Processor 和 Adapter。

组件之间通过抽象接口交互。这样做还有一个好处是**封装变化。**这是面向对象设计的精髓，将系统中经常变化的部分和稳定的部分隔离，有助于增加复用性，并降低系统耦合度。

网络通信的 I/O 模型是变化的，可能是非阻塞 I/O、异步 I/O 或者 APR。应用层协议也是变化的，可能是 HTTP、HTTPS、AJP。浏览器端发送的请求信息也是变化的。

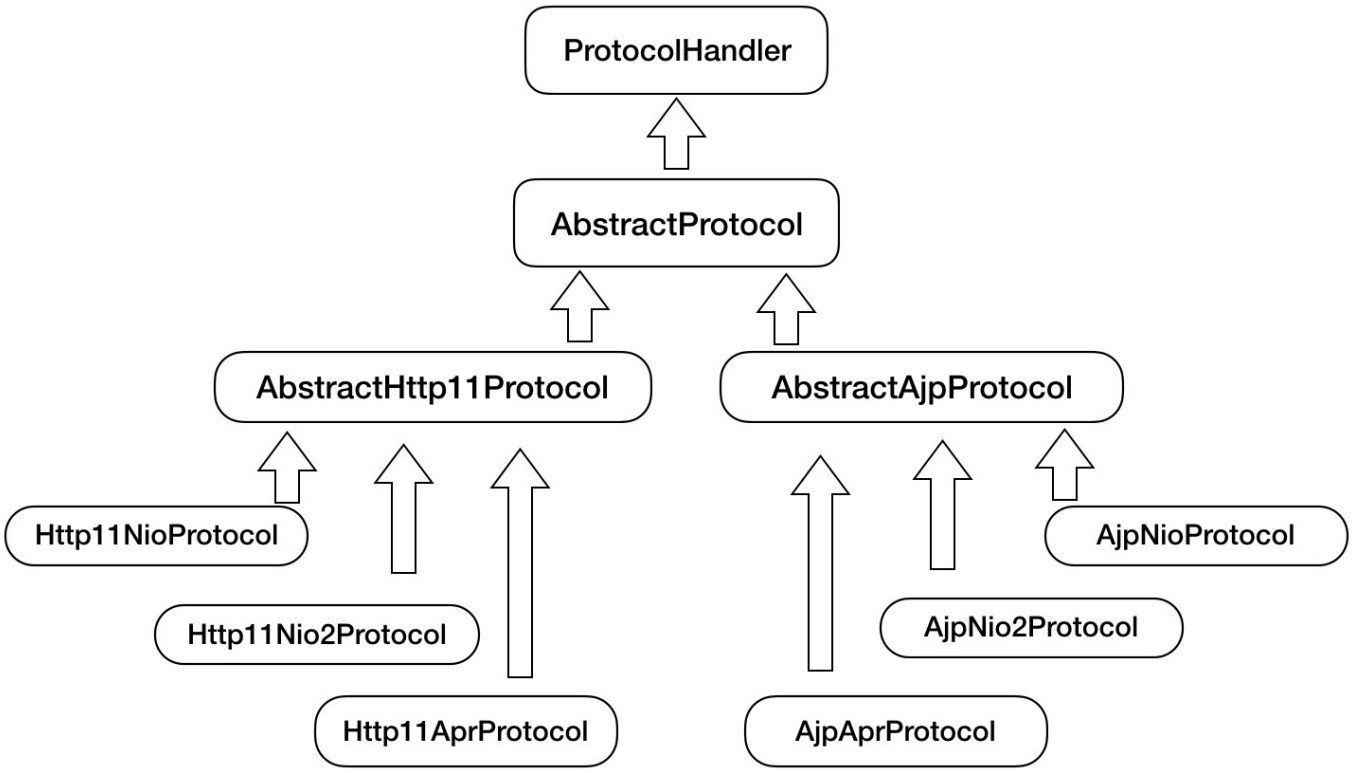
但是整体的处理逻辑是不变的，EndPoint 负责提供字节流给 Processor，Processor 负责提供 Tomcat Request 对象给 Adapter，Adapter 负责提供 ServletRequest 对象给容器。

如果要支持新的 I/O 方案、新的应用层协议，只需要实现相关的具体子类，上层通用的处理逻辑是不变的。

由于 I/O 模型和应用层协议可以自由组合，比如 NIO + HTTP 或者 NIO2 + AJP。Tomcat 的设计者将网络通信和应用层协议解析放在一起考虑，设计了一个叫 ProtocolHandler 的

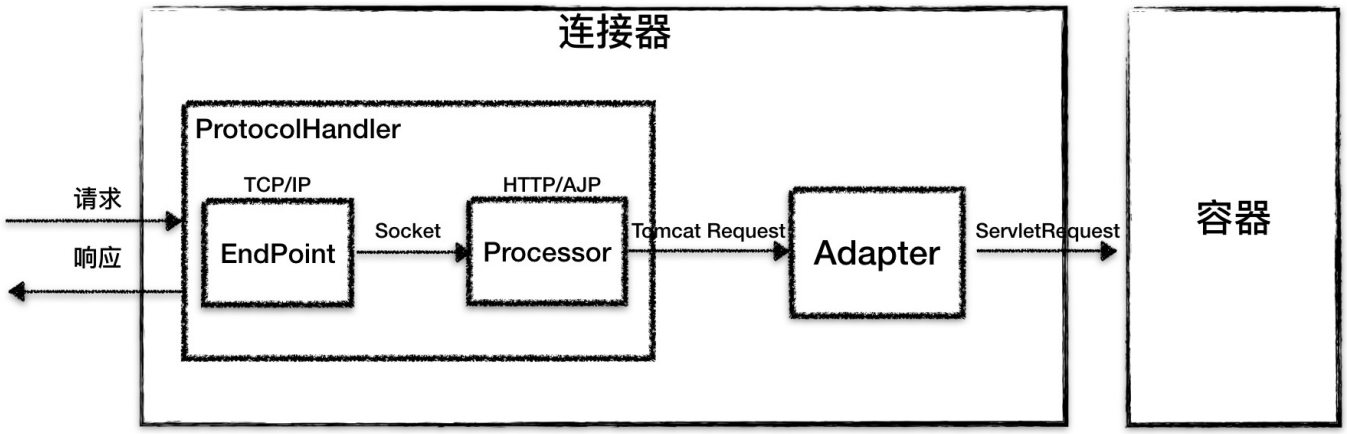
接口来封装这两种变化点。各种协议和通信模型的组合有相应的具体实现类。比如：Http11NioProtocol 和 AjpNioProtocol。

除了这些变化点，系统也存在一些相对稳定的部分，因此 Tomcat 设计了一系列抽象基类来**封装这些稳定的部分**，抽象基类 AbstractProtocol 实现了 ProtocolHandler 接口。每一种应用层协议有自己的抽象基类，比如 AbstractAjpProtocol 和 AbstractHttp11Protocol，具体协议的实现类扩展了协议层抽象基类。下面我整理一下它们的继承关系。



通过上面的图，你可以清晰地看到它们的继承和层次关系，这样设计的目的是尽量将稳定的部分放到抽象基类，同时每一种 I/O 模型和协议的组合都有相应的具体实现类，我们在使用时可以自由选择。

小结一下，连接器模块用三个核心组件：Endpoint、Processor 和 Adapter 来分别做三件事情，其中 Endpoint 和 Processor 放在一起抽象成了 ProtocolHandler 组件，它们的关系如下图所示。



下面我来详细介绍这两个顶层组件 ProtocolHandler 和 Adapter。

ProtocolHandler 组件

由上文我们知道，连接器用 ProtocolHandler 来处理网络连接和应用层协议，包含了 2 个重要部件：EndPoint 和 Processor，下面我来详细介绍它们的工作原理。

EndPoint

EndPoint 是通信端点，即通信监听的接口，是具体的 Socket 接收和发送处理器，是对传输层的抽象，因此 EndPoint 是用来实现 TCP/IP 协议的。

EndPoint 是一个接口，对应的抽象实现类是 AbstractEndpoint，而 AbstractEndpoint 的具体子类，比如在 NioEndpoint 和 Nio2Endpoint 中，有两个重要的子组件：Acceptor 和 SocketProcessor。

其中 Acceptor 用于监听 Socket 连接请求。SocketProcessor 用于处理接收到的 Socket 请求，它实现 Runnable 接口，在 Run 方法里调用协议处理组件 Processor 进行处理。为了提高处理能力，SocketProcessor 被提交到线程池来执行。而这个线程池叫作执行器（Executor)，我在后面的专栏会详细介绍 Tomcat 如何扩展原生的 Java 线程池。

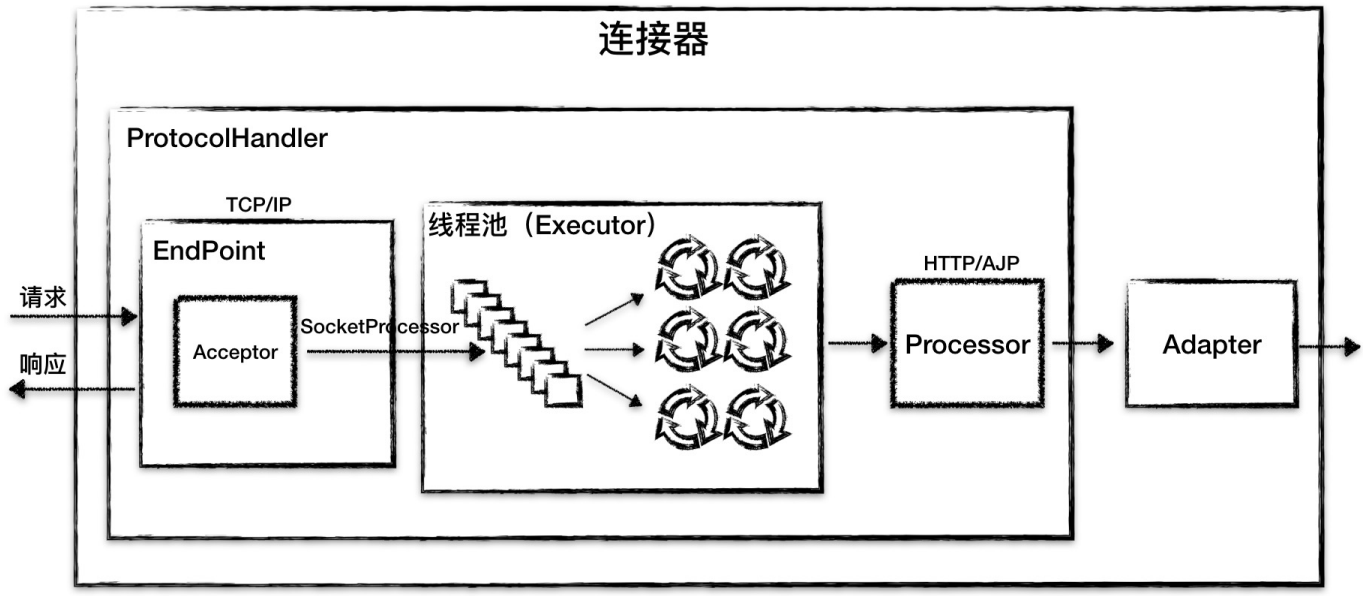
Processor

如果说 EndPoint 是用来实现 TCP/IP 协议的，那么 Processor 用来实现 HTTP 协议，Processor 接收来自 EndPoint 的 Socket，读取字节流解析成 Tomcat Request 和

Response 对象，并通过 Adapter 将其提交到容器处理，Processor 是对应用层协议的抽象。

Processor 是一个接口，定义了请求的处理等方法。它的抽象实现类 AbstractProcessor 对一些协议共有的属性进行封装，没有对方法进行实现。具体的实现有 AJPProcessor、HTTP11Processor 等，这些具体实现类实现了特定协议的解析方法和请求处理方式。

我们再来看看连接器的组件图：



从图中我们看到，EndPoint 接收到 Socket 连接后，生成一个 SocketProcessor 任务提交到线程池去处理，SocketProcessor 的 Run 方法会调用 Processor 组件去解析应用层协议，Processor 通过解析生成 Request 对象后，会调用 Adapter 的 Service 方法。

到这里我们学习了 ProtocolHandler 的总体架构和工作原理，关于 EndPoint 的详细设计，后面我还会专门介绍 EndPoint 是如何最大限度地利用 Java NIO 的非阻塞以及 NIO2 的异步特性，来实现高并发。

Adapter 组件

我在前面说过，由于协议不同，客户端发过来的请求信息也不尽相同，Tomcat 定义了自己的 Request 类来“存放”这些请求信息。ProtocolHandler 接口负责解析请求并生成 Tomcat Request 类。但是这个 Request 对象不是标准的 ServletRequest，也就意味着，不能用 Tomcat Request 作为参数来调用容器。Tomcat 设计者的解决方案是引入 CoyoteAdapter，这是适配器模式的经典运用，连接器调用 CoyoteAdapter 的 Sevice 方

法，传入的是 Tomcat Request 对象，CoyoteAdapter 负责将 Tomcat Request 转成 ServletRequest，再调用容器的 Service 方法。

本期精华

Tomcat 的整体架构包含了两个核心组件连接器和容器。连接器负责对外交流，容器负责内部处理。连接器用 ProtocolHandler 接口来封装通信协议和 I/O 模型的差异，ProtocolHandler 内部又分为 EndPoint 和 Processor 模块，EndPoint 负责底层 Socket 通信，Proccesor 负责应用层协议解析。连接器通过适配器 Adapter 调用容器。

通过对 Tomcat 整体架构的学习，我们可以得到一些设计复杂系统的基本思路。首先要分析需求，根据高内聚低耦合的原则确定子模块，然后找出子模块中的变化点和不变点，用接口和抽象基类去封装不变点，在抽象基类中定义模板方法，让子类自行实现抽象方法，也就是具体子类去实现变化点。

课后思考

回忆一下你在工作中曾经独立设计过的系统，或者你碰到过的设计类面试题，结合今天专栏的内容，你有没有一些新的思路？

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。

上一篇　04 | 实战：纯手工打造和运行一个Servlet

下一篇　06 | Tomcat系统架构（下）：聊聊多层容器的设计

精选留言 (59)

写留言

电光火石
2019-05-21

21

对Tomcat的结构有个清晰的了解，其中有两个问题：
1. PorotocolHandler的继承关系是不是太重了，看起来像典型的多维度扩展，nio2在apj和1HTTP11都要做一遍，用组合会不会更好
2. 为什么要多一层adapter，在processor直接转换为容器的servletrequest和servletresponse不是更好，为什么要先转化Tomcat的request和response，再用adapt...
展开

作者回复: 1，说的对，能用组合就不用继承，这里我感觉Tomcat设计者考虑的是通过多层继承来尽量重用一些通用的逻辑。另外I/O模型和应用层协议的个数也是可控的，用户可以在server.xml中直接指定想要的连接器类型：比如Http11NioProtocol和Http11Nio2Protocol。

2，这里的考虑是，如果连接器直接创建ServletRequest和ServletResponse对象的话，就和Servlet协议耦合了，设计者认为连接器尽量保持独立性，它不一定要跟Servlet容器工作的。另外对象转化的性能消耗还是比较少的，Tomcat对HTTP请求体采取了延迟解析的策略，也就是说，TomcatRequest对象转化成ServletRequest的时候，请求体的内容都还没读取呢，直到容器处理这个请求的时候才读取的。

囍
2019-05-22

4

“EndPoint 是通信端点，即通信监听的接口，是具体的 Socket 接收和发送处理器，是对传输层的抽象，因此 EndPoint 是用来实现 TCP/IP 协议的。”，【EndPoint是用来实现TCP/IP协议的】这个没有太明白，据我有限的知识所知，TCP/IP协议是【由操作系统实现】的，而socket只是在TCP/IP之上展现给用户层的一个接口，而EndPoint又用到了socket接口（我瞎猜的）。所以，我是否可以把这句话理解为，EndPoint利用Socket接...

展开

作者回复: 理解正确👍

-W.LI-
2019-05-21

3

老师好!Tomcat配置的并发数是文中endpoint里那个线程池么?IO方面知识比较薄弱，希望老师后期讲解时多花点心思。

作者回复: 是的

新世界
2019-05-21

3

对tomcat的结构的连接器部分收获不少，有一问题，tomcat的endpoint的功能和netty的实现功能很多方面一样，tomcat为什么没有用netty作为底层通讯框架?

展开

作者回复: Tomcat在I/O模型和线程模型方面跟Netty很相似，后面会详细分析。

325G
2019-05-21

2

Adapter一层使用的是适配器设计模式，好处是当容器版本升级只修改Adaper组件适配到新版本容器就可以了，protocal handler组件代码不需要改动

听雨
2019-05-21

2

一个service对应tomcat中部署的一个项目，一个连接器对应一个请求，这样理解对吗

作者回复: 一个Service中可以部署多个项目呢，一个连接器对应一个监听端口，不是一个请求，一个端口上可以接收多个请求。

李海明
2019-05-21

2

看懂了，但是没有一个形象化的记忆点。

展开

作者回复: 😑后面还会有整体架构图。

永光
2019-06-05

1

老师，你看这样理解对不，
采用何种I/O模式（NIO、NIO2、ARP），以及采用何种应用协议（HTTP1.1、AJP、HTTP/2)都是在processor这一层决定的。EndPoint只负责接收连接，并读取网络字节流但是不对字节流本身就进行任何解析。

作者回复: 对的

王智
2019-05-23

1

老师您好,我有两个问题,
一个是上面说到一个容器对接多个连接器,也就是service,这个具体是不是可以在tomcat的conf目录下的server.xml中发现呢? 但是一般情况下,也就是默认的,tomcat的一个server下只会有一个service组件,而connector就是在service组件中配置的呢?
另一个是一个server中有一个或多个service,一个service中有多个连接器和一个容器,这里...
展开

作者回复: 1.对的，默认是一个service
2.容器就是装载Servlet的箱子，Tomcat的容器分层次，大箱子里有小箱子，最大的箱子是Engine，下一篇会讲到。

Geek_ebda9...
2019-05-23

1

老师请教两个问题
1.应用层的i/o模型和http1，ajp等协议是指在endpoint接受网络请求后，对请求内容解析才会用到吧，就是在processor里面，这里面就是根据请求的协议类型，采用指定i/o读取网络流，是不是这样?
2.ajp也是指一种网络协议么，类似于http这种，processor里面是根据什么来判定请求的...
展开

作者回复: 1. 对的
2.AJP可以理解为应用层协议，它是用二进制的方式来传输文本，比如HTTP的请求头“accept-language”有15个字符，如果用二进制0xA004表示只有2个字节，效率大大提升。
3. Endpoint中的Acceptor有多个，每个Acceptor跑在单独的线程里，后面会详细分析为什么要这样做。

馒头
2019-05-22

1

现在不抓源码，只去了解整个项目的架构，自己也快要做项目经理了，需要带团队自己搭建项目，要从tomcat里学习经典框架!

allean
2019-05-22

1

可以理解为一个连接器对应一个应用吗

展开

作者回复: 不是的，一个连接器对应一个监听端口，比如一扇门，一个web应用是一个业务部门，进了这个门后你可以到各个业务部门去办事。

allean 1
2019-05-22

一个service有多个连接器和一个容器，多个Service就可能有n个连接器和n个容器吗？还是有且只有一个容器，所有的连接器都指向这个容器，请老师解答，谢谢！

展开

作者回复: Service是对外提供服务的单位，一个Service对应一个容器，多个Service就有多个容器。

z.l 1
2019-05-21

“如果说 EndPoint 是用来实现 TCP/IP 协议的，那么 Processor 用来实现 HTTP 协议”，这句话不太理解，TCP/IP协议不是由linux系统内核实现的么？

作者回复: 这里可以理解为Endpoint负责Socket网络通信，跟TCP/IP协议紧密相关。“实现”这个词确实有点误导。：）

Monday 1
2019-05-21

从上一节突然跳转到本节，感觉跳跃性很大。突然进入整体架构后，即使我花了大量时间多次阅读本节，也很难消化。真的捉急！
不知道老师上面提到的类名，是基于Tomcat的哪个版本。
今天我刻意花时间把tomcat7.0.94的源码下载下来，导入IDEA。发现org.apache.tomcat.util.net.AbstractEndpoint是一个抽象类，既没有实现EndPoint，...

展开

作者回复: 刚开始是需要适应一下^_^，其实不难的，我用的最新版的代码：https://github.com/apache/tomcat，AbstractEndpoint本身确实没有内部类，是它的子类Nio2Endpoint中包含了两个内部类：Nio2Acceptor 和 SocketProcessor。已经在下面的回复中已经纠正了。

锦
2019-05-21

1

两个问题请教一下老师
第一，如何debug源码呢?
第二，tomcat和netty有什么区别呢？为什么netty常常用做底层通讯模块，而tomcat作为web容器呢?

作者回复: 1）软件系统本质是对信息的处理，要跟踪信息在流动过程中的经过的关键环节，并在这些地方下断点，看看变量的值是什么。比如你可以在业务代码中下个断点，看看调用栈，看Tomcat和Spring是怎么调到你的代码的，然后在这个调用栈中的关键函数里上下都看看，先熟悉个大概，然后带着问题去深入调试。
2）你可以把Netty理解成Tomcat中的连接器，它们都负责网络通信，都利用了Java NIO非阻塞特性。但Netty素以高性能高并发著称，为什么Tomcat不把连接器替换成Netty呢？第一个原因是Tomcat的连接器性能已经足够好了，同样是Java NIO编程，套路都差不多。第二个原因是Tomcat做为Web容器，需要考虑到Servlet规范，Servlet规范规定了对HTTP Body的读写是阻塞的，因此即使用到了Netty，也不能充分发挥它的优势。所以Netty一般用在非HTTP协议和Servlet的场景下。

zhycaree...
2019-05-21

1

老师，源码如何阅读效果好啊？现在源码一大堆，不知从何下手。谢谢

展开

作者回复: 抓主线，抓主干，每个系统中都有一个关键的核心类，紧紧抓住这些类，先不要分散，在逐步看旁枝，等你学习弄明白一个经典的系统，很多套路你就明白了。

泉清

1

2019-05-21

一、课后题：刚好最近独立设计开发过一个以第三方为标准的项目，其实无论谁为标准，要做的事情非自己系统与别人系统高度耦合的定制项目，都可以进行抽离。
1、自己先把流程分析，确定那些是固定的，哪些是变的；
2、定义自己标准版的数据结构，对于任何来自第三方的数据都可以用适配器模式转化为自己标准版的。...
展开

作者回复: 源码在这里：https://github.com/apache/tomcat

你说的对，AbstractEndpoint中没有内部类，是它的子类Nio2Endpoint包含了两个内部类：Nio2Acceptor 和 SocketProcessor。

chp 1
2019-05-21

老师，请求来的时候，源码入口在哪里？
展开

作者回复: 在Acceptor的run方法里：

socket = endpoint.serverSocketAccept();

这句话用来接收一个新的连接

nimil 1
2019-05-21

这篇很棒。受益匪浅，谢谢老师
展开