

Tomcat 常见面试题总结

本文内容主要整理自：

- 《深入拆解 Tomcat & Jetty》
- 《Tomcat 架构解析》

感谢这两份资料，尤其是《深入拆解 Tomcat & Jetty》，写的非常赞，看了之后收货颇多。

虽然这篇文章的内容大部分都不是我的原创，但整理重要的知识点和面试题同样花了不少心思，希望对你有帮助！

Tomcat 介绍

什么是 Web 容器？

早期的 Web 应用主要用于浏览新闻等静态页面，HTTP 服务器（比如 Apache、Nginx）向浏览器返回静态 HTML，浏览器负责解析 HTML，将结果呈现给用户。

随着互联网的发展，我们已经不满足于仅仅浏览静态页面，还希望通过一些交互操作，来获取动态结果，因此也就需要一些扩展机制能够让 HTTP 服务器调用服务端程序。

于是 Sun 公司推出了 Servlet 技术。你可以把 Servlet 简单理解为运行在服务端的 Java 小程序，但是 Servlet 没有 main 方法，不能独立运行，因此必须把它部署到 Servlet 容器中，由容器来实例化并调用 Servlet。

Tomcat 就是一个 Servlet 容器。为了方便使用，Tomcat 同时具有 HTTP 服务器的功能。

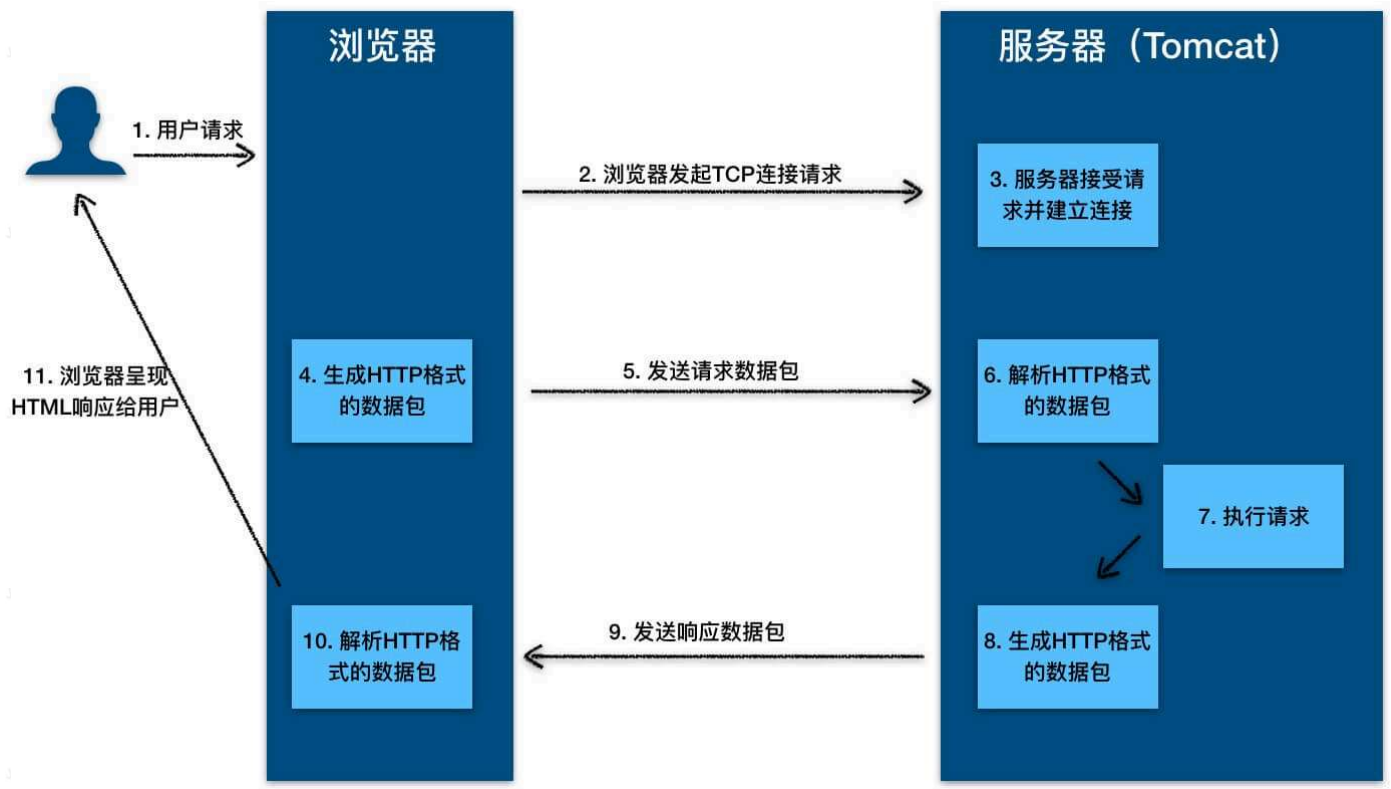
因此 Tomcat 就是一个“HTTP 服务器 + Servlet 容器”，我们也叫它 Web 容器。

什么是 Tomcat？

简单来说，Tomcat 就是一个“HTTP 服务器 + Servlet 容器”，我们通常也称呼 Tomcat 为 Web 容器。

- **HTTP 服务器**：处理 HTTP 请求并响应结果。
- **Servlet 容器**：HTTP 服务器将请求交给 Servlet 容器处理，Servlet 容器会将请求转发到具体的 Servlet（Servlet 容器用来加载和管理业务类）。

HTTP 服务器工作原理了解吗？



1. 用户通过浏览器进行了一个操作，比如输入网址并回车，或者是点击链接，接着浏览器获取了这个事件。
2. 浏览器向服务端发出 TCP 连接请求。
3. 服务端程序接受浏览器的连接请求，并经过 TCP 三次握手建立连接。
4. 浏览器将请求数据打包成一个 HTTP 协议格式的数据包。
5. 浏览器将该数据包推入网络，数据包经过网络传输，最终达到端服务程序。
6. 服务端程序拿到这个数据包后，同样以 HTTP 协议格式解包，获取到客户端的意图。
7. 得知客户端意图后进行处理，比如提供静态文件或者调用服务端程序获得动态结果。
8. 服务器将响应结果（可能是 HTML 或者图片等）按照 HTTP 协议格式打包。
9. 服务器将响应数据包推入网络，数据包经过网络传输最终达到浏览器。
10. 浏览器拿到数据包后，以 HTTP 协议的格式解包，然后解析数据，假设这里的数据是 HTML。
11. 浏览器将 HTML 文件展示在页面上。

什么是 Servlet?有什么作用?

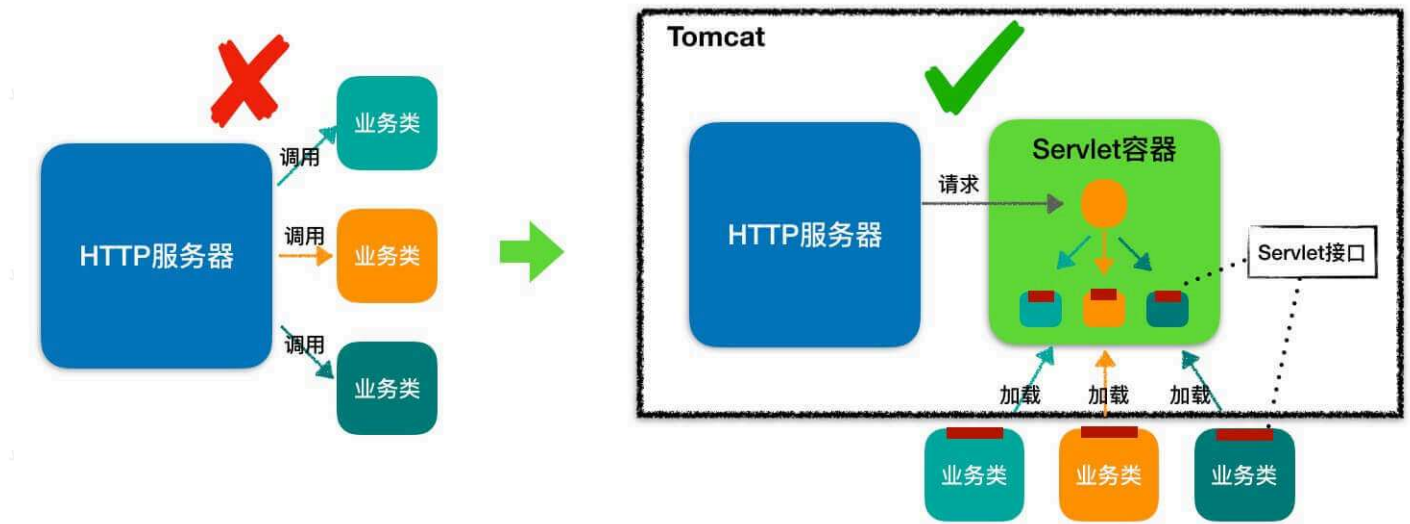
Servlet 指的是任何实现了 `Servlet` 接口的类。Servlet 主要用于处理客户端传来的 HTTP 请求，并返回一个响应。

Servlet 接口定义了下面五个方法：

```
1 public interface Servlet {
2     void init(ServletConfig config) throws ServletException;
3
4     ServletConfig getServletConfig();
5
6     void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;
7
8     String getServletInfo();
9
10    void destroy();
11 }
```

其中最重要的是 `service` 方法，具体业务类在这个方法里实现业务的具体处理逻辑。

Servlet 容器会根据 `web.xml` 文件中的映射关系，调用相应的 Servlet，Servlet 将处理的结果返回给 Servlet 容器，并通过 HTTP 服务器将响应传输给客户端。



几乎所有的 Java Web 框架（比如 Spring）都是基于 Servlet 的封装。

Tomcat 是如何创建 Servlet 的？

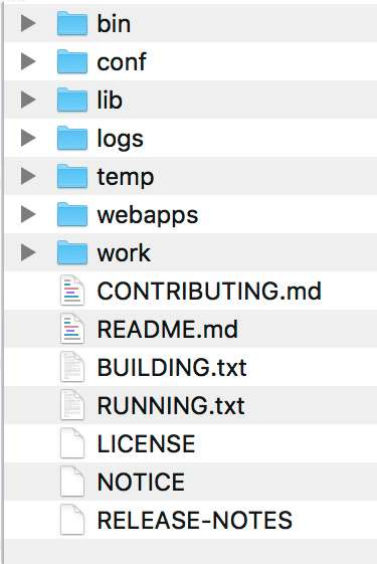
当容器启动时，会读取在 webapps 目录下所有的 web 应用中的 web.xml 文件，然后对 xml 文件进行解析，并读取 servlet 注册信息。然后，将每个应用中注册的 Servlet 类都进行加载，并通过反射的方式实例化。（有时候也是在第一次请求时实例化）。

`<load-on-startup>` 元素是 `<servlet>` 元素的一个子元素，它用于指定 Servlet 被加载的时机和顺序。在 `<load-on-startup>` 元素中，设置的值必须是一个整数。如果这个值是一个负数，或者没有设定这个元素，Servlet 容器将在客户端首次请求这个 Servlet 时加载它；如果这个值是正整数或 0，Servlet 容器将在 Web 应用启动时加载并初始化 Servlet，并且 `<load-on-startup>` 的值越小，它对应的 Servlet 就越先被加载。

具体配置方式如下所示：

```
XML |
1  <servlet>
2    <servlet-name>HelloWorldServlet</servlet-name>
3    <servlet-class>
4      cn.itcast.firstapp.servlet.HelloWorldServlet
5    </servlet-class>
6    <!-- 设置Servlet在Web应用启动时初始化 -->
7    <load-on-startup>1</load-on-startup>
8  </servlet>
9  <servlet-mapping>
10    <!-- HelloWorldServlet在Tomcat启动时就被自动加载并且初始化了。 -->
11    <servlet-name>HelloWorldServlet</servlet-name>
12    <url-pattern>/helloWorldServlet</url-pattern>
13  </servlet-mapping>
```

Tomcat 文件夹



- /bin: 存放 Windows 或 Linux 平台上启动和关闭 Tomcat 的脚本文件。
- /conf: 存放 Tomcat 的各种全局配置文件，其中最重要的是 server.xml。
- /lib: 存放 Tomcat 以及所有 Web 应用都可以访问的 JAR 文件。
- /logs: 存放 Tomcat 执行时产生的日志文件。
- /work: 存放 JSP 编译后产生的 Class 文件。
- /webapps: Tomcat 的 Web 应用目录，默认情况下把 Web 应用放在这个目录下。

bin 目录有什么作用？

```
Bash |
1  $ ls tomcat/bin
2  bootstrap.jar          configtest.bat  setclasspath.bat  tcnative-1.dll*   tool-wrapper.sh*
3  catalina.bat           configtest.sh*  setclasspath.sh*  tomcat8.exe*      version.bat
4  catalina.sh*           daemon.sh*      shutdown.bat      tomcat8w.exe*     version.sh*
5  catalina-tasks.xml     digest.bat      shutdown.sh*      tomcat-juli.jar
6  commons-daemon.jar     digest.sh*      startup.bat       tomcat-native.tar.gz
7  commons-daemon-native.tar.gz service.bat      startup.sh*       tool-wrapper.bat
```

bin 目录保存了对 Tomcat 进行控制的相关可执行程序。

上面的文件中，主要分为两类：.bat 和 .sh。 .bat 是 window 平台的批处理文件，用于在 window 中执行。而 .sh 则是在 Linux 或者 Unix 上执行的。

比较常用的是下面两个：

- startup.sh (startup.bat) 用来启动 Tomcat 服务器。
- shutdown.sh (shutdown.bat) 用来关闭已经运行的 Tomcat 服务器。

webapps 目录有什么作用？

webapps 目录用来存放应用程序，当 Tomcat 启动时会去加载 webapps 目录下的应用程序。可以以文件夹、war 包、jar 包的形式发布应用。

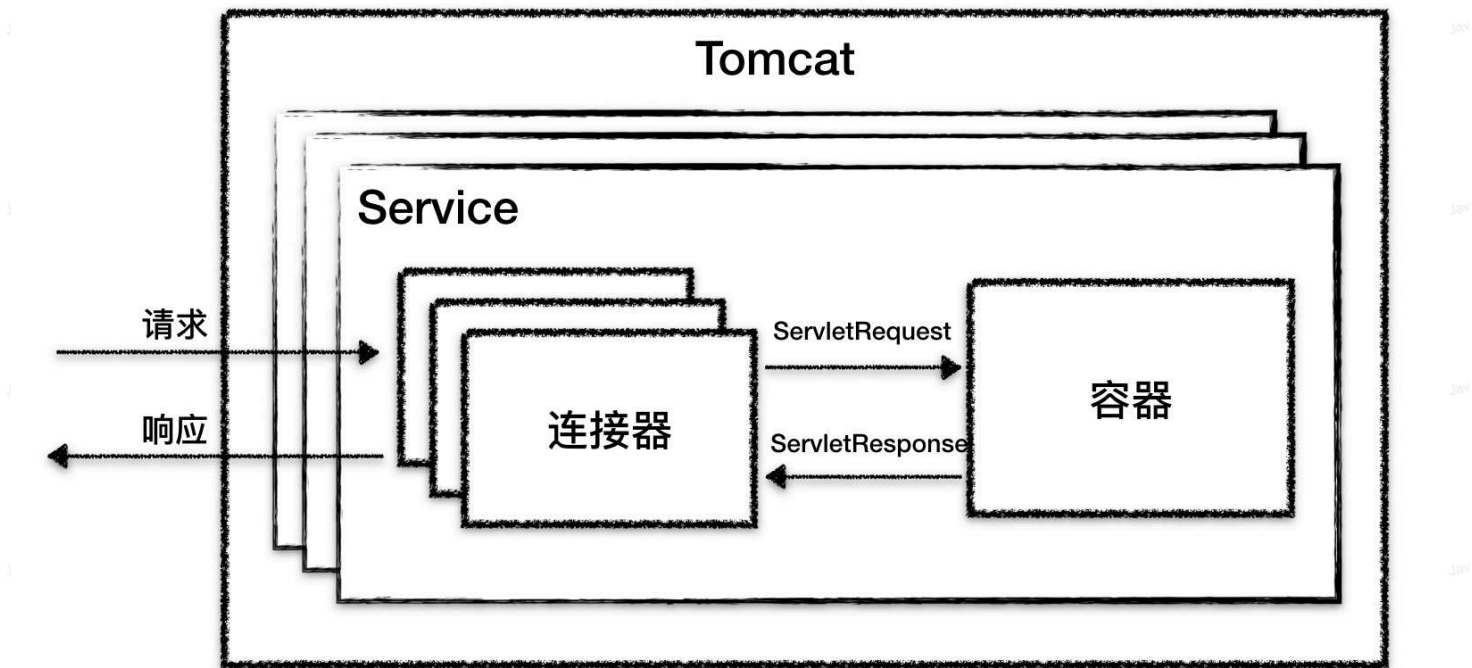
当然，你也可以把应用程序放置在磁盘的任意位置，在配置文件中映射好就行。

Tomcat 总体架构

Tomcat 要实现 2 个核心功能：

1. 处理 Socket 连接，负责网络字节流与 Request 和 Response 对象的转化。
2. 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。



连接器有什么作用？

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：

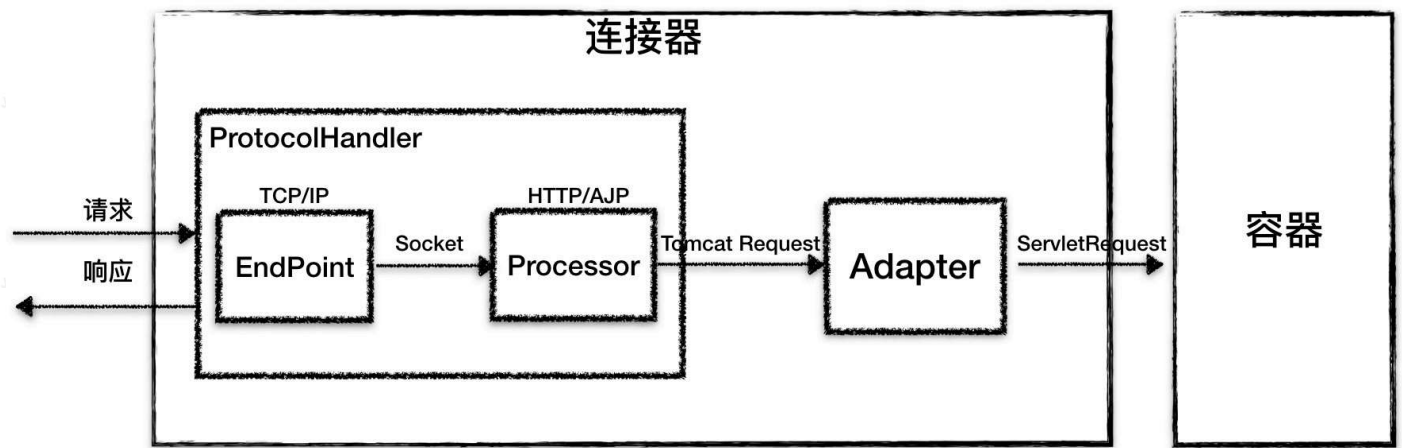
- 监听网络端口。
- 接受网络连接请求。
- 读取网络请求字节流。
- 根据具体应用层协议（HTTP/AJP）解析字节流，生成统一的 `Tomcat Request` 对象。
- 将 `Tomcat Request` 对象转成标准的 `ServletRequest`。
- 调用 `Servlet` 容器，得到 `ServletResponse`。
- 将 `ServletResponse` 转成 `Tomcat Response` 对象。
- 将 `Tomcat Response` 转成网络字节流。
- 将响应字节流写回给浏览器。

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：

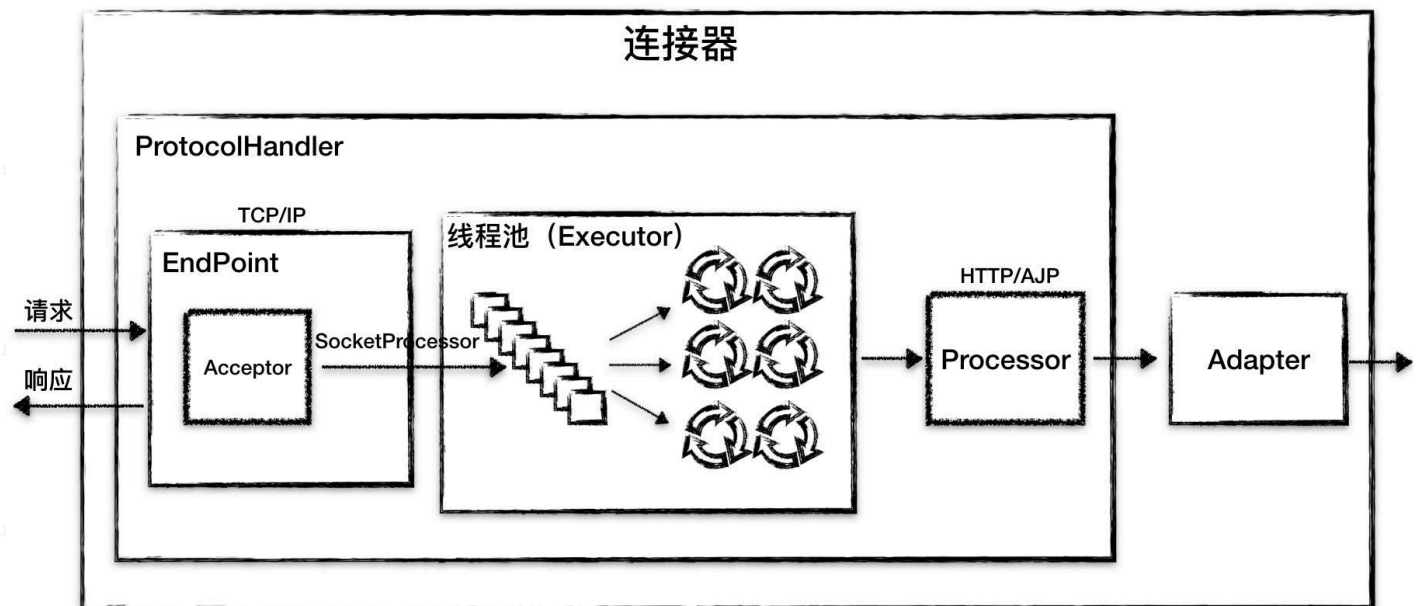
- 网络通信。
- 应用层协议解析。
- `Tomcat Request/Response` 与 `ServletRequest/ServletResponse` 的转化。

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**（适配器模式）。

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。



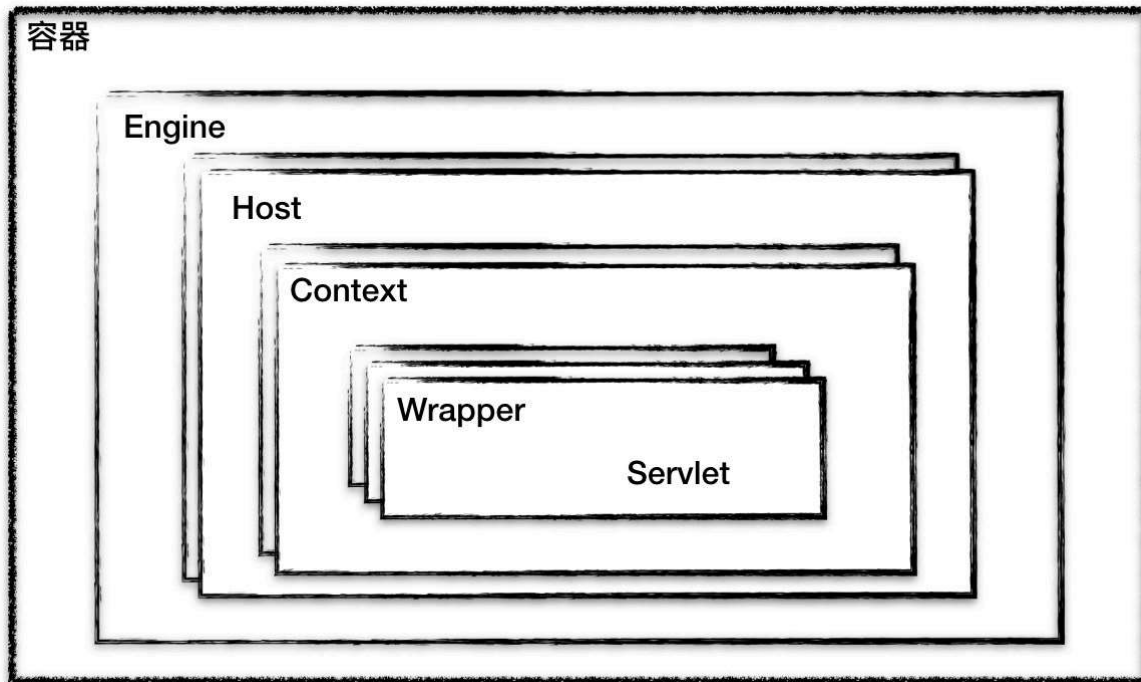
连接器用 `ProtocolHandler` 接口来封装通信协议和 I/O 模型的差异, `ProtocolHandler` 内部又分为 `Endpoint` 和 `Processor` 模块, `Endpoint` 负责底层 Socket 通信, `Processor` 负责应用层协议解析。连接器通过适配器 Adapter 调用容器。



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。

容器是怎么设计的?

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。



- **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>
```

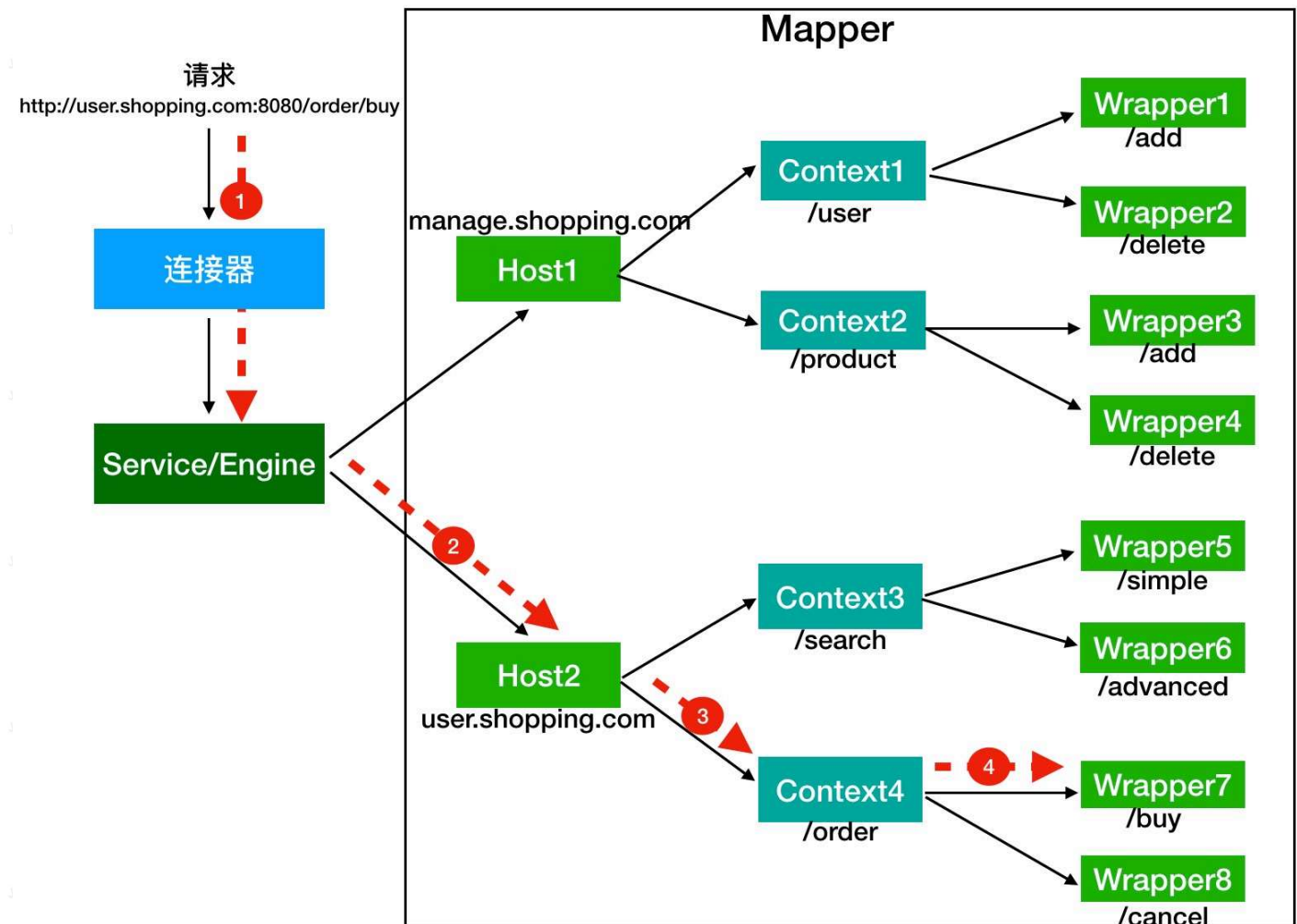
请求是如何定位到 Servlet 的?

Tomcat 是怎么确定请求是由哪个 Wrapper 容器里的 Servlet 来处理的呢?

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。

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。

举个例子: 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`。



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？

1. 根据协议和端口号选定 Service 和 Engine：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。
2. 根据域名选定 Host：域名是 user.shopping.com，因此 Mapper 会找到 Host2 这个容器。
3. 根据 URL 路径找到 Context 组件。
4. 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。

Tomcat 为什么要打破双亲委托机制？

Tomcat 自定义类加载器打破双亲委托机制的目的是为了优先加载 Web 应用目录下的类，然后再加载其他目录下的类，这也是 Servlet 规范的推荐做法。

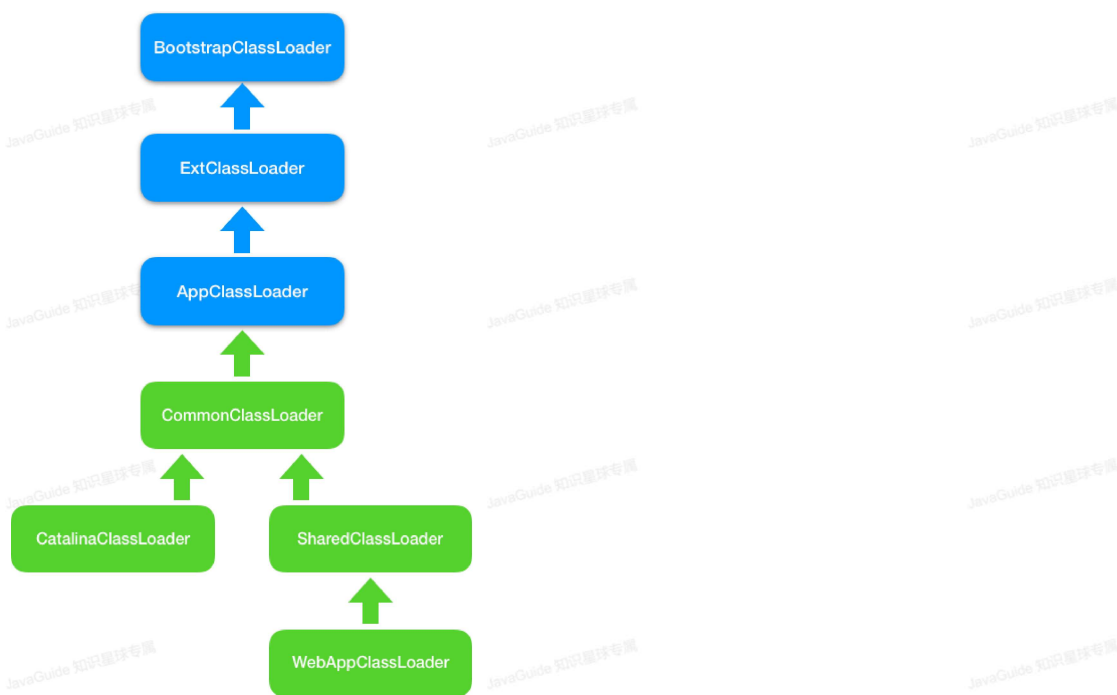
要打破双亲委托机制，需要继承 `ClassLoader` 抽象类，并且需要重写它的 `loadClass` 方法，因为 `ClassLoader` 的默认实现就是双亲委托。

Tomcat 如何隔离 Web 应用？

首先让我们思考一下这一个问题：

1. 假如我们在 Tomcat 中运行了两个 Web 应用程序，两个 Web 应用中有同名的 Servlet，但是功能不同，Tomcat 需要同时加载和管理这两个同名的 Servlet 类，保证它们不会冲突，因此 Web 应用之间的类需要隔离。
2. 假如两个 Web 应用都依赖同一个第三方的 JAR 包，比如 Spring，那 Spring 的 JAR 包被加载到内存后，Tomcat 要保证这两个 Web 应用能够共享，也就是说 Spring 的 JAR 包只被加载一次，否则随着依赖的第三方 JAR 包增多，JVM 的内存会膨胀。
3. 跟 JVM 一样，我们需要隔离 Tomcat 本身的类和 Web 应用的类。

为了解决上面这些问题，Tomcat 设计了类加载器的层次结构。



我们先来看第 1 个问题: Web 应用之间的类之间如何隔离?

假如我们使用 JVM 默认 AppClassLoader 来加载 Web 应用, AppClassLoader 只能加载一个 Servlet 类, 在加载第二个同名 Servlet 类时, AppClassLoader 会返回第一个 Servlet 类的 Class 实例, 这是因为在 AppClassLoader 看来, 同名的 Servlet 类只被加载一次。

Tomcat 的解决方案是自定义一个类加载器 WebAppClassLoader, 并且给每个 Web 应用创建一个类加载器实例。我们知道, Context 容器组件对应一个 Web 应用, 因此, 每个 Context 容器负责创建和维护一个 WebAppClassLoader 加载器实例。这背后的原理是, 不同的加载器实例加载的类被认为是不同的类, 即使它们的类名相同。这就相当于在 Java 虚拟机内部创建了一个个相互隔离的 Java 类空间, 每一个 Web 应用都有自己的类空间, Web 应用之间通过各自的类加载器互相隔离。

我们再来看第 2 个问题: 两个 Web 应用之间怎么共享库类, 并且不能重复加载相同的类?

我们知道, 在双亲委托机制里, 各个子加载器都可以通过父加载器去加载类, 那么把需要共享的类放到父加载器的加载路径下不就行了吗, 应用程序也正是通过这种方式共享 JRE 的核心类。因此 Tomcat 的设计者又加了一个类加载器 SharedClassLoader, 作为 WebAppClassLoader 的父加载器, 专门来加载 Web 应用之间共享的类。如果 WebAppClassLoader 自己没有加载到某个类, 就会委托父加载器 SharedClassLoader 去加载这个类, SharedClassLoader 会在指定目录下加载共享类, 之后返回给 WebAppClassLoader, 这样共享的问题就解决了。

我们再来看第 3 个问题:如何隔离 Tomcat 本身的类和 Web 应用的类?

我们知道, 要共享可以通过父子关系, 要隔离那就需要兄弟关系了。兄弟关系就是指两个类加载器是平行的, 它们可能拥有同一个父加载器, 但是两个兄弟类加载器加载的类是隔离的。基于此 Tomcat 又设计一个类加载器 CatalinaClassLoader, 专门来加载 Tomcat 自身的类。这样设计有问题, 那 Tomcat 和各 Web 应用之间需要共享一些类时该怎么办呢?

老办法, 还是再增加一个 CommonClassLoader, 作为 CatalinaClassLoader 和 SharedClassLoader 的父加载器。CommonClassLoader 能加载的类都可以被 CatalinaClassLoader 和 SharedClassLoader 使用, 而 CatalinaClassLoader 和 SharedClassLoader 能加载的类则与对方相互隔离。WebAppClassLoader 可以使用 SharedClassLoader 加载到的类, 但各个 WebAppClassLoader 实例之间相互隔离。

性能优化

如何监控 Tomcat 性能?

Tomcat 的关键的性能指标主要有 吞吐量、响应时间、错误数、线程池、CPU 以及 JVM 内存。

1. 通过 JConsole 监控 Tomcat
2. 命令行查看 Tomcat 指标
3. prometheus + grafana

JVM GC 原理及调优的基本思路

Tomcat 基于 Java, 也是跑在 JVM 中, 因此, 我们要对 Tomcat 进行调优的话, 先要了解 JVM 调优的原理。

JVM 调优主要是对 JVM 垃圾收集的优化。一般来说是因为有问题才需要优化，所以对于 JVM GC 来说，如果你观察到 Tomcat 进程的 CPU 使用率比较高，并且在 GC 日志中发现 GC 次数比较频繁、GC 停顿时间长，这表明你需要对 GC 进行优化了。

在对 GC 调优的过程中，我们不仅需要知道 GC 的原理，更重要的是要熟练使用各种监控和分析工具，具备 GC 调优的实战能力。

CMS 和 G1 是时下使用率比较高的两款垃圾收集器，从 Java 9 开始，采用 G1 作为默认垃圾收集器，而 G1 的目标也是逐步取代 CMS。

如何选择 IO 模型？

I/O 调优实际上是连接器类型的选择，一般情况下默认都是 NIO，在绝大多数情况下都是够用的，除非你的 Web 应用用到了 TLS 加密传输，而且对性能要求极高，这个时候可以考虑 APR，因为 APR 通过 OpenSSL 来处理 TLS 握手和加 / 解密。OpenSSL 本身用 C 语言实现，它还对 TLS 通信做了优化，所以性能比 Java 要高。

那你可能会问那什么时候考虑选择 NIO.2？

- 如果你的 Tomcat 跑在 Windows 平台上，并且 HTTP 请求的数据量比较大，可以考虑 NIO.2，这是因为 Windows 从操作系统层面实现了真正意义上的异步 I/O，如果传输的数据量比较大，异步 I/O 的效果就能显现出来。
- 如果你的 Tomcat 跑在 Linux 平台上，建议使用 NIO，这是因为 Linux 内核没有很完善地支持异步 I/O 模型，因此 JVM 并没有采用原生的 Linux 异步 I/O，而是在应用层面通过 epoll 模拟了异步 I/O 模型，只是 Java NIO 的使用者感觉不到而已。因此可以这样理解，在 Linux 平台上，Java NIO 和 Java NIO.2 底层都是通过 epoll 来实现的，但是 Java NIO 更加简单高效。