

08 | Tomcat的“高层们”都负责做什么？

2019-05-28 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)

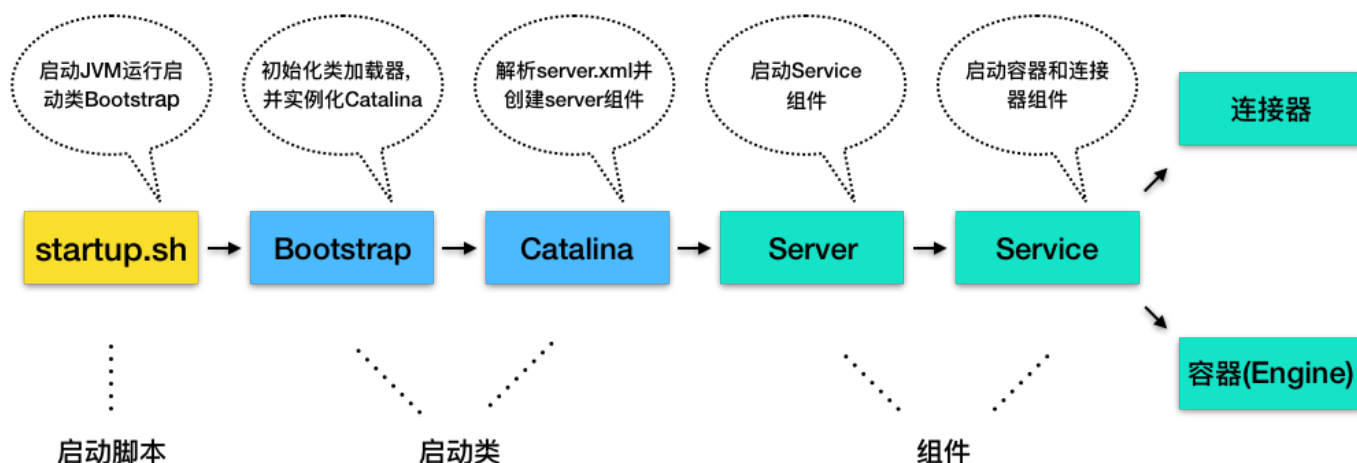


讲述：李号双

时长 08:58 大小 8.22M



使用过 Tomcat 的同学都知道，我们可以通过 Tomcat 的 /bin 目录下的脚本 startup.sh 来启动 Tomcat，那你是否知道我们执行了这个脚本后发生了什么呢？你可以通过下面这张流程图来了解一下。



1.Tomcat 本质上是一个 Java 程序，因此 startup.sh 脚本会启动一个 JVM 来运行 Tomcat 的启动类 Bootstrap。

2.Bootstrap 的主要任务是初始化 Tomcat 的类加载器，并且创建 Catalina。关于 Tomcat 为什么需要自己的类加载器，我会在专栏后面详细介绍。

3.Catalina 是一个启动类，它通过解析 server.xml、创建相应的组件，并调用 Server 的 start 方法。

4.Server 组件的职责就是管理 Service 组件，它会负责调用 Service 的 start 方法。

5.Service 组件的职责就是管理连接器和顶层容器 Engine，因此它会调用连接器和 Engine 的 start 方法。

这样 Tomcat 的启动就算完成了。下面我来详细介绍一下上面这个启动过程中提到的几个非常关键的启动类和组件。

你可以把 Bootstrap 看作是上帝，它初始化了类加载器，也就是创造万物的工具。

如果我们把 Tomcat 比作是一家公司，那么 Catalina 应该是公司创始人，因为 Catalina 负责组建团队，也就是创建 Server 以及它的子组件。

Server 是公司的 CEO，负责管理多个事业群，每个事业群就是一个 Service。

Service 是事业群总经理，它管理两个职能部门：一个是对外的市场部，也就是连接器组件；另一个是对内的研发部，也就是容器组件。

Engine 则是研发部经理，因为 Engine 是最顶层的容器组件。


你可以看到这些启动类或者组件不处理具体请求，它们的任务主要是“管理”，管理下层组件的生命周期，并且给下层组件分配任务，也就是把请求路由到负责“干活儿”的组件。因此我把它比作 Tomcat 的“高层”。

今天我们就来看看这些“高层”的实现细节，目的是让我们逐步理解 Tomcat 的工作原理。另一方面，软件系统中往往都有一些起管理作用的组件，你可以学习和借鉴 Tomcat

是如何实现这些组件的。


Catalina

Catalina 的主要任务就是创建 Server，它不是直接 new 一个 Server 实例就完事了，而是需要解析 server.xml，把在 server.xml 里配置的各种组件一一创建出来，接着调用 Server 组件的 init 方法和 start 方法，这样整个 Tomcat 就启动起来了。作为“管理者”，Catalina 还需要处理各种“异常”情况，比如当我们通过“Ctrl + C”关闭 Tomcat 时，Tomcat 将如何优雅的停止并且清理资源呢？因此 Catalina 在 JVM 中注册一个“关闭钩子”。

 复制代码

```
1 public void start() {
2     //1. 如果持有的 Server 实例为空，就解析 server.xml 创建出来
3     if (getServer() == null) {
4         load();
5     }
6     //2. 如果创建失败，报错退出
7     if (getServer() == null) {
8         log.fatal(sm.getString("catalina.noServer"));
9         return;
10    }
11
12    //3. 启动 Server
13    try {
14        getServer().start();
15    } catch (LifecycleException e) {
16        return;
17    }
18
19    // 创建并注册关闭钩子
20    if (useShutdownHook) {
21        if (shutdownHook == null) {
22            shutdownHook = new CatalinaShutdownHook();
23        }
24        Runtime.getRuntime().addShutdownHook(shutdownHook);
25    }
26
27    // 用 await 方法监听停止请求
28    if (await) {
29        await();
30        stop();
31    }
32 }
```

那什么是“关闭钩子”，它又是做什么的呢？如果我们需要在 JVM 关闭时做一些清理工作，比如将缓存数据刷到磁盘上，或者清理一些临时文件，可以向 JVM 注册一个“关闭钩子”。“关闭钩子”其实就是一个线程，JVM 在停止之前会尝试执行这个线程的 run 方法。下面我们来看看 Tomcat 的“关闭钩子” CatalinaShutdownHook 做了些什么。


 复制代码

```
1 protected class CatalinaShutdownHook extends Thread {
2
3     @Override
4     public void run() {
5         try {
6             if (getServer() != null) {
7                 Catalina.this.stop();
8             }
9         } catch (Throwable ex) {
10             ...
11         }
12     }
13 }
```

从这段代码中你可以看到，Tomcat 的“关闭钩子”实际上就执行了 Server 的 stop 方法，Server 的 stop 方法会释放和清理所有的资源。

Server 组件

Server 组件的具体实现类是 StandardServer，我们来看下 StandardServer 具体实现了哪些功能。Server 继承了 LifecycleBase，它的生命周期被统一管理，并且它的子组件是 Service，因此它还需要管理 Service 的生命周期，也就是说在启动时调用 Service 组件的启动方法，在停止时调用它们的停止方法。Server 在内部维护了若干 Service 组件，它是以数组来保存的，那 Server 是如何添加一个 Service 到数组中的呢？

 复制代码

```
1 @Override
2 public void addService(Service service) {
3
4     service.setServer(this);
5
6     synchronized (servicesLock) {
7         // 创建一个长度 +1 的新数组
8         Service results[] = new Service[services.length + 1];
9     }
```

```

10      // 将老的数据复制过去
11      System.arraycopy(services, 0, results, 0, services.length);
12      results[services.length] = service;
13      services = results;
14
15      // 启动 Service 组件
16      if (getState().isAvailable()) {
17          try {
18              service.start();
19          } catch (LifecycleException e) {
20              // Ignore
21          }
22      }
23
24      // 触发监听事件
25      support.firePropertyChange("service", null, service);
26  }
27
28 }

```

从上面的代码你能看到，它并没有一开始就分配一个很长的数组，而是在添加的过程中动态地扩展数组长度，当添加一个新的 Service 实例时，会创建一个新数组并把原来数组内容复制到新数组，这样做的目的其实是为了节省内存空间。

除此之外，Server 组件还有一个重要的任务是启动一个 Socket 来监听停止端口，这就是为什么你能通过 shutdown 命令来关闭 Tomcat。不知道你留意到没有，上面 Caralina 的启动方法的最后一行代码就是调用了 Server 的 await 方法。

在 await 方法里会创建一个 Socket 监听 8005 端口，并在一个死循环里接收 Socket 上的连接请求，如果有新的连接到来就建立连接，然后从 Socket 中读取数据；如果读到的数据是停止命令“SHUTDOWN”，就退出循环，进入 stop 流程。

Service 组件

Service 组件的具体实现类是 StandardService，我们先来看看它的定义以及关键的成员变量。

 复制代码

```

1 public class StandardService extends LifecycleBase implements Service {
2     // 名字
3     private String name = null;

```

```

4
5 //Server 实例
6 private Server server = null;
7
8 // 连接器数组
9 protected Connector connectors[] = new Connector[0];
10 private final Object connectorsLock = new Object();
11
12 // 对应的 Engine 容器
13 private Engine engine = null;
14
15 // 映射器及其监听器
16 protected final Mapper mapper = new Mapper();
17 protected final MapperListener mapperListener = new MapperListener(this);

```

StandardService 继承了 LifecycleBase 抽象类，此外 StandardService 中还有一些我们熟悉的组件，比如 Server、Connector、Engine 和 Mapper。

那为什么还有一个 MapperListener？这是因为 Tomcat 支持热部署，当 Web 应用的部署发生变化时，Mapper 中的映射信息也要跟着变化，MapperListener 就是一个监听器，它监听容器的变化，并把信息更新到 Mapper 中，这是典型的观察者模式。

作为“管理”角色的组件，最重要的是维护其他组件的生命周期。此外在启动各种组件时，要注意它们的依赖关系，也就是说，要注意启动的顺序。我们来看看 Service 启动方法：

 复制代码

```

1 protected void startInternal() throws LifecycleException {
2
3 //1. 触发启动监听器
4 setState(LifecycleState.STARTING);
5
6 //2. 先启动 Engine，Engine 会启动它子容器
7 if (engine != null) {
8     synchronized (engine) {
9         engine.start();
10    }
11 }
12
13 //3. 再启动 Mapper 监听器
14 mapperListener.start();
15
16 //4. 最后启动连接器，连接器会启动它子组件，比如 Endpoint
17 synchronized (connectorsLock) {
18     for (Connector connector: connectors) {


```

```
19         if (connector.getState() != LifecycleState.FAILED) {
20             connector.start();
21         }
22     }
23 }
24 }
```

从启动方法可以看到，Service 先启动了 Engine 组件，再启动 Mapper 监听器，最后才是启动连接器。这很好理解，因为内层组件启动好了才能对外提供服务，才能启动外层的连接器组件。而 Mapper 也依赖容器组件，容器组件启动好了才能监听它们的变化，因此 Mapper 和 MapperListener 在容器组件之后启动。组件停止的顺序跟启动顺序正好相反的，也是基于它们的依赖关系。


Engine 组件

最后我们再来看看顶层的容器组件 Engine 具体是如何实现的。Engine 本质是一个容器，因此它继承了 ContainerBase 基类，并且实现了 Engine 接口。

 复制代码


```
1 public class StandardEngine extends ContainerBase implements Engine {
2 }
```

我们知道，Engine 的子容器是 Host，所以它持有了一个 Host 容器的数组，这些功能都被抽象到了 ContainerBase 中，ContainerBase 中有这样一个数据结构：

 复制代码

```
1 protected final HashMap<String, Container> children = new HashMap<>();
```

ContainerBase 用 HashMap 保存了它的子容器，并且 ContainerBase 还实现了子容器的“增删改查”，甚至连子组件的启动和停止都提供了默认实现，比如 ContainerBase 会用专门的线程池来启动子容器。

 复制代码

```
1 for (int i = 0; i < children.length; i++) {
2     results.add(startStopExecutor.submit(new StartChild(children[i])));
3 }
```

所以 Engine 在启动 Host 子容器时就直接重用了这个方法。

那 Engine 自己做了什么呢？我们知道容器组件最重要的功能是处理请求，而 Engine 容器对请求的“处理”，其实就是把请求转发给某一个 Host 子容器来处理，具体是通过 Valve 来实现的。

通过专栏前面的学习，我们知道每一个容器组件都有一个 Pipeline，而 Pipeline 中有一个基础阀（Basic Valve），而 Engine 容器的基础阀定义如下：

 复制代码

```
1 final class StandardEngineValve extends ValveBase {
2
3     public final void invoke(Request request, Response response)
4         throws IOException, ServletException {
5
6         // 拿到请求中的 Host 容器
7         Host host = request.getHost();
8         if (host == null) {
9             return;
10        }
11
12        // 调用 Host 容器中的 Pipeline 中的第一个 Valve
13        host.getPipeline().getFirst().invoke(request, response);
14    }
15
16 }
```

这个基础阀实现非常简单，就是把请求转发到 Host 容器。你可能好奇，从代码中可以看到，处理请求的 Host 容器对象是从请求中拿到的，请求对象中怎么会有 Host 容器呢？这是因为请求到达 Engine 容器中之前，Mapper 组件已经对请求进行了路由处理，Mapper 组件通过请求的 URL 定位了相应的容器，并且把容器对象保存到了请求对象中。

本期精华

今天我们学习了 Tomcat 启动过程，具体是由启动类和“高层”组件来完成的，它们都承担着“管理”的角色，负责将子组件创建出来，并把它们拼装在一起，同时也掌握子组件的“生杀大权”。

所以当我们在设计这样的组件时，需要考虑两个方面：

首先要选用合适的数据结构来保存子组件，比如 Server 用数组来保存 Service 组件，并且采取动态扩容的方式，这是因为数组结构简单，占用内存小；再比如 ContainerBase 用 HashMap 来保存子容器，虽然 Map 占用内存会多一点，但是可以通过 Map 来快速的查找子容器。因此在实际的工作中，我们也需要根据具体的场景和需求来选用合适的数据结构。

其次还需要根据子组件依赖关系来决定它们的启动和停止顺序，以及如何优雅的停止，防止异常情况下的资源泄漏。这正是“管理者”应该考虑的事情。

课后思考

Server 组件的在启动连接器和容器时，都分别加了锁，这是为什么呢？

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 07 | Tomcat如何实现一键式启停？

下一篇 09 | 比较：Jetty架构特点之Connector组件

精选留言 (22)

写留言



一路远行

2019-05-28

7

加锁通常的场景是存在多个线程并发操作不安全的数据结构。

不安全的数据结构：

Server本身包含多个Service，内部实现上用数组来存储services，数组的并发操作(包含扩容，扩容)是不安全的。所以，在并发操作(添加/修改/删除/遍历等)services数组时，需...

展开



allea

2019-05-28

3

老师的专栏思路清晰，讲解的深入但又通俗易懂，感谢大佬带我飞😊

展开 ▾



W.T

2019-05-28

👍 2

老师，以上源码是基于tomcat的哪个版本？

展开 ▾

作者回复: 最新9.x版

◀ ▶



刘章周

2019-06-04

👍 1

老师，1.catalina创建组件，是把所有的对象都new出来了吧，只是各个组件之间没有相互注入吧。

2.为什么catalina直接调用server的start方法？不是先init吗？

3.容器之间是什么时候注入进去的？还有listener是什么时候注入到组件中去的？

展开 ▾

作者回复: 1，对的，直接new出来

2，start方法里调了init方法

3，父容器应该是在构造函数里new了子容器

◀ ▶



why

2019-06-02

👍 1

- Tomcat 本质是 Java 程序, startup.sh 启动 JVM 运行 Tomcat 启动类 bootstrap

- Bootstrap 初始化类加载器, 创建 Catalina

- Catalina 解析 server.xml, 创建相应组件, 调用 Server start 方法

- Server 组件管理 Service 组件并调用其 start 方法...

展开 ▾



清风

2019-05-29

👍 1

调试源码，会遇到在执行不到对应的断点的情况，看调用栈是被之前的断点拦住了，这个

有什么好的断点调试方法吗，这样调试太费劲了

作者回复: IDE一般有disable断点的功能

◀ ▶



轩恒

2019-05-29

👍 1

有个常见问题请教一下，在实际应用场景中，tomcat在shutdown的时候，无法杀死java进程，还得kill，这是为何呢？

作者回复: Tomcat会调用Web应用的代码来处理请求，可能Web应用代码阻塞在某个地方。

◀ ▶



Monday

2019-05-29

👍 1

listener对应的中文（三个字）竟然是敏感词。。。

展开 ▾



Monday

2019-05-29

👍 1

根据老师的给出的Github上Tomcat源码调试Tomcat的启动过程，遇到以下这个问题。经debug发现，运行完Catalina.load()方法的第566行digester.parse(inputSource)初始化了Server对象。但是我单步进入第566行，各种操作都没有跟踪到具体是哪一行初始化了Server对象。莫非有Listener？

展开 ▾

作者回复: digester应该是通过反射的方式创建Server对象的。

◀ ▶



allea

2019-05-28

👍 1

如果映射关系不变，而是某个具体的Servlet的方法处理逻辑变了，热部署也可以解决重启tomcat的尴尬吗

作者回复: 那不叫热部署, 叫热加载。
热部署和热加载都不需要重启Tomcat。



wwm

2019-05-28

1

老师, 请教一个问题:

在Bootstrap中, 基于什么原因用反射的方式创建Catalina实例, 之后继续基于反射方式调用load、init、start这些方法? 为什么不是直接new Catalina实例后通过实例直接调用这些方法?

作者回复: Tomcat有自己的类加载器体系, Catalina相关的类都是由专门的类加载器catalinaLoader来加载:

```
Class<?> startupClass = catalinaLoader.loadClass("org.apache.catalina.startup.Catalina")
```

Tomcat的类加载器体系会有专门的文章来详细解释。



大卫

2019-06-05

1

老师好, tomcat一般生产环境线程数大小建议怎么设置呢

展开 ∨

作者回复: 理论上:

线程数 = ((线程阻塞时间 + 线程忙碌时间) / 线程忙碌时间) * cpu核数

如果线程始终不阻塞, 一直忙碌, 会一直占用一个CPU核, 因此可以直接设置 线程数=CPU核数。

但是现实中线程可能会被阻塞, 比如等待IO。因此根据上面的公式确定线程数。

那怎么确定线程的忙碌时间和阻塞时间? 要经过压测, 在代码中埋点统计, 专栏后面的调优环节会涉及到。



佑儿

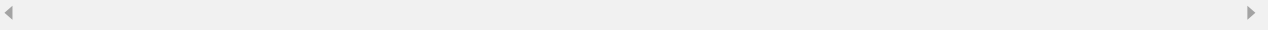
2019-06-04



Mapping和MapperListener会重点讲嘛？

展开 ∨

作者回复: 在Tomcat系统架构下篇有Mapper的原理介绍，怎么将一个请求映射到具体容器来处理。



802.11

2019-06-02



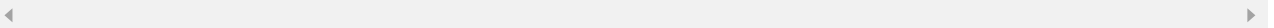
abel614: {

label613: {

try {

老师，在tomcat出现的，这是什么语法呀

作者回复: 可能是你的IDE没有把源码下载下来，这是反编译的代码：)



Geek_00d56...

2019-05-30



今天我们学习了 Tomcat 启动过程，具体是由启动类和“高层”组件来完成的，它们都承担着“管理”的角色，负责将子组件创建出来，并把它们拼装在一起，同时也掌握子组件的“生杀大权”。

加锁🔒，是因为有并发。...

展开 ∨



锦

2019-05-29



加锁可能会有多个连接器和容器并发创建。

展开 ∨



王智

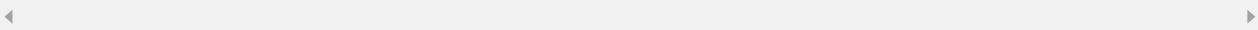
2019-05-29



老师,具体的注入是个什么样的概念,上节课说的子组件注入到父组件,内层组件注入到外层组件,这是个什么样的操作,在上面的代码中并没有看到具体的操作呢?

展开 ▾

作者回复: 其实就是父组件持有子组件的引用, new出来就好了。



-W.LI-

2019-05-29



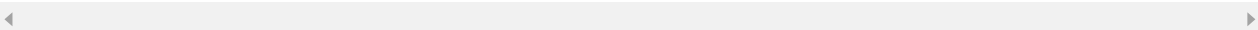
老师好!catalina的start()方法末尾那部分不太理解能帮忙讲解下么。

// 用 await 方法监听停止请求

```
if (await) {  
    await();  
    stop();...
```

展开 ▾

作者回复: 不怪把, 这里的考虑是怎么让Tomcat这个程序不退出, 否则程序执行完了就终止了。



发条橙子 ...

2019-05-29



老师, 对于你的问题, 实际上我也不理解为何要加锁。

首先, 按理说server对每一个service开一个线程去初始化。 应该不会多个线程对一个service同时初始化吧。

再者, 这块同步如果是要防止重复初始化, 那应该在start()方法中做, 否则等释放锁后, 下一个线程获得锁还是会执行start()方法。 ...

展开 ▾

作者回复: 不管外面怎么调, 加了锁这个方法就是线程安全的了



QQ怪

2019-05-29



因为还用了线程不安全的hashmap

展开 ✓