

23 | Host容器：Tomcat如何实现热部署和热加载？

2019-07-02 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



讲述：李号双

时长 09:10 大小 7.36M



从这一期我们开始学习 Tomcat 的容器模块，来聊一聊各容器组件实现的功能，主要有热部署热加载、类加载机制以及 Servlet 规范的实现。最后还会谈到 Spring Boot 是如何与 Web 容器进行交互的。

今天我们首先来看热部署和热加载。要在运行的过程中升级 Web 应用，如果你不想重启系统，实现的方式有两种：热加载和热部署。

那如何实现热部署和热加载呢？它们跟类加载机制有关，具体来说就是：


热加载的实现方式是 Web 容器启动一个后台线程，定期检测类文件的变化，如果有变化，就重新加载类，在这个过程中不会清空 Session，一般用在开发环境。

热部署原理类似，也是由后台线程定时检测 Web 应用的变化，但它会重新加载整个 Web 应用。这种方式会清空 Session，比热加载更加干净、彻底，一般用在生产环境。

今天我们来学习一下 Tomcat 是如何用后台线程来实现热加载和热部署的。Tomcat 通过开启后台线程，使得各个层次的容器组件都有机会完成一些周期性任务。我们在实际工作中，往往也需要执行一些周期性的任务，比如监控程序周期性拉取系统的健康状态，就可以借鉴这种设计。

Tomcat 的后台线程

要说开启后台线程做周期性的任务，有经验的同学马上会想到线程池中的 `ScheduledThreadPoolExecutor`，它除了具有线程池的功能，还能够执行周期性的任务。Tomcat 就是通过它来开启后台线程的：

 复制代码

```
1 bgFuture = exec.scheduleWithFixedDelay(  
2     new ContainerBackgroundProcessor(), // 要执  
3     backgroundProcessorDelay, // 第一次执行延迟  
4     backgroundProcessorDelay, // 之后每次执行间隔  
5     TimeUnit.SECONDS); // 时间单位
```

上面的代码调用了 `scheduleWithFixedDelay` 方法，传入了四个参数，第一个参数就是要周期性执行的任务类 `ContainerBackgroundProcessor`，它是一个 `Runnable`，同时也是 `ContainerBase` 的内部类，`ContainerBase` 是所有容器组件的基类，我们来回忆一下容器组件有哪些，有 `Engine`、`Host`、`Context` 和 `Wrapper` 等，它们具有父子关系。

ContainerBackgroundProcessor 实现

我们接来看 `ContainerBackgroundProcessor` 具体是如何实现的。

```
1 protected class ContainerBackgroundProcessor implements
2
3     @Override
4     public void run() {
5         // 请注意这里传入的参数是 " 宿主类 " 的实例
6         processChildren(ContainerBase.this);
7     }
8
9     protected void processChildren(Container container)
10        try {
11            //1. 调用当前容器的 backgroundProcess 方法。
12            container.backgroundProcess();
13
14            //2. 遍历所有的子容器，递归调用 processChildr
15            // 这样当前容器的子孙都会被处理
16            Container[] children = container.findChildr
17            for (int i = 0; i < children.length; i++) {
18                // 这里请你注意，容器基类有个变量叫做 backgrou
19                if (children[i].getBackgroundProcessor() != null)
20                    processChildren(children[i]);
21            }
22        }
23        } catch (Throwable t) { ... }
```

上面的代码逻辑也是比较清晰的，首先 ContainerBackgroundProcessor 是一个 Runnable，它需要实现 run 方法，它的 run 很简单，就是调用了 processChildren 方法。这里有个小技巧，它把 “宿主

类”，也就是**ContainerBase** 的类实例当成参数传给了 **run** 方法。

而在 **processChildren** 方法里，就做了两步：调用当前容器的 **backgroundProcess** 方法，以及递归调用子孙的 **backgroundProcess** 方法。请你注意 **backgroundProcess** 是 **Container** 接口中的方法，也就是说所有类型的容器都可以实现这个方法，在这个方法里完成需要周期性执行的任务。

这样的设计意味着什么呢？我们只需要在顶层容器，也就是 **Engine** 容器中启动一个后台线程，那么这个线程**不但会执行 **Engine** 容器的周期性任务，它还会执行所有子容器的周期性任务。**

backgroundProcess 方法

上述代码都是在基类 **ContainerBase** 中实现的，那具体容器类需要做什么呢？其实很简单，如果有周期性任务要执行，就实现 **backgroundProcess** 方法；如果没有，就重用基类 **ContainerBase** 的方法。**ContainerBase** 的 **backgroundProcess** 方法实现如下：

```

1 public void backgroundProcess() {
2
3     //1. 执行容器中 Cluster 组件的周期性任务
4     Cluster cluster = getClusterInternal();
5     if (cluster != null) {
6         cluster.backgroundProcess();
7     }
8
9     //2. 执行容器中 Realm 组件的周期性任务
10    Realm realm = getRealmInternal();
11    if (realm != null) {
12        realm.backgroundProcess();
13    }
14
15    //3. 执行容器中 Valve 组件的周期性任务
16    Valve current = pipeline.getFirst();
17    while (current != null) {
18        current.backgroundProcess();
19        current = current.getNext();
20    }
21
22    //4. 触发容器的 " 周期事件 ", Host 容器的监听器 HostCc
23    fireLifecycleEvent(Lifecycle.PERIODIC_EVENT, null);
24 }

```

从上面的代码可以看到，不仅每个容器可以有周期性任务，每个容器中的其他通用组件，比如跟集群管理有关的 Cluster 组件、跟安全管理有关的 Realm 组件都可以有自己的周期性任务。

我在前面的专栏里提到过，容器之间的链式调用是通过 Pipeline-Valve 机制来实现的，从上面的代码你可以看到容器中的 Valve 也可以有周期性任务，并且被 ContainerBase 统一处理。

请你特别注意的是，在 backgroundProcess 方法的最后，还触发了容器的“周期事件”。我们知道容器的生命周期事件有初始化、启动和停止等，那“周期事件”又是什么呢？它跟生命周期事件一样，是一种扩展机制，你可以这样理解：


又一段时间过去了，容器还活着，你想做点什么吗？如果你想做点什么，就创建一个监听器来监听这个“周期事件”，事件到了我负责调用你的方法。

总之，有了 ContainerBase 中的后台线程和 backgroundProcess 方法，各种子容器和通用组件不需要各自弄一个后台线程来处理周期性任务，这样的设计显得优雅和整洁。

Tomcat 热加载

有了 ContainerBase 的周期性任务处理“框架”，作为具体容器子类，只需要实现自己的周期性任务就行。而

Tomcat 的热加载，就是在 Context 容器中实现的。
Context 容器的 backgroundProcess 方法是这样实现的：

 复制代码

```
1 public void backgroundProcess() {
2
3     //WebappLoader 周期性的检查 WEB-INF/classes 和 WEB-I
4     Loader loader = getLoader();
5     if (loader != null) {
6         loader.backgroundProcess();
7     }
8
9     //Session 管理器周期性的检查是否有过期的 Session
10    Manager manager = getManager();
11    if (manager != null) {
12        manager.backgroundProcess();
13    }
14
15    // 周期性的检查静态资源是否有变化
16    WebResourceRoot resources = getResources();
17    if (resources != null) {
18        resources.backgroundProcess();
19    }
20
21    // 调用父类 ContainerBase 的 backgroundProcess 方法
22    super.backgroundProcess();
23 }
```

从上面的代码我们看到 Context 容器通过 WebappLoader 来检查类文件是否有更新，通过 Session 管理器来检查是否


有 Session 过期，并且通过资源管理器来检查静态资源是否有更新，最后还调用了父类 ContainerBase 的 backgroundProcess 方法。

这里我们要重点关注，WebappLoader 是如何实现热加载的，它主要是调用了 Context 容器的 reload 方法，而 Context 的 reload 方法比较复杂，总结起来，主要完成了下面这些任务：

1. 停止和销毁 Context 容器及其所有子容器，子容器其实就是 Wrapper，也就是说 Wrapper 里面 Servlet 实例也被销毁了。
2. 停止和销毁 Context 容器关联的 Listener 和 Filter。
3. 停止和销毁 Context 下的 Pipeline 和各种 Valve。
4. 停止和销毁 Context 的类加载器，以及类加载器加载的类文件资源。
5. 启动 Context 容器，在这个过程中会重新创建前面四步被销毁的资源。

在这个过程中，类加载器发挥着关键作用。一个 Context 容器对应一个类加载器，类加载器在销毁的过程中会把它加载的所有类也全部销毁。Context 容器在启动过程中，会创建一个新的类加载器来加载新的类文件。

在 Context 的 reload 方法里，并没有调用 Session 管理器的 distroy 方法，也就是说这个 Context 关联的 Session 是没有销毁的。你还需要注意的是，Tomcat 的热加载默认是关闭的，你需要在 conf 目录下的 Context.xml 文件中设置 reloadable 参数来开启这个功能，像下面这样：

 复制代码

```
1 <Context reloadable="true"/>
```


Tomcat 热部署

我们再来看看热部署，热部署跟热加载的本质区别是，热部署会重新部署 Web 应用，原来的 Context 对象会整个被销毁掉，因此这个 Context 所关联的一切资源都会被销毁，包括 Session。

那么 Tomcat 热部署又是由哪个容器来实现的呢？应该不是由 Context，因为热部署过程中 Context 容器被销毁了，那么这个重担就落在 Host 身上了，因为它是 Context 的父容器。


跟 Context 不一样，Host 容器并没有在 backgroundProcess 方法中实现周期性检测的任务，而是

通过监听器 HostConfig 来实现的，HostConfig 就是前面提到的“周期事件”的监听器，那“周期事件”达到时，HostConfig 会做什么事呢？

 复制代码

```
1 public void lifecycleEvent(LifecycleEvent event) {
2     // 执行 check 方法。
3     if (event.getType().equals(Lifecycle.PERIODIC_EVENT
4         check());
5     }
6 }
```

它执行了 check 方法，我们接着来看 check 方法里做了什么。

 复制代码

```
1 protected void check() {
2
3     if (host.getAutoDeploy()) {
4         // 检查这个 Host 下所有已经部署的 Web 应用
5         DeployedApplication[] apps =
6             deployed.values().toArray(new DeployedAppli
7
8         for (int i = 0; i < apps.length; i++) {
9             // 检查 Web 应用目录是否有变化
10             checkResources(apps[i], false);
11         }
12 }
```

```
13         // 执行部署
14         deployApps();
15     }
16 }
```



其实 HostConfig 会检查 webapps 目录下的所有 Web 应用：

如果原来 Web 应用目录被删掉了，就把相应 Context 容器整个销毁掉。

是否有新的 Web 应用目录放进来了，或者有新的 WAR 包放进来了，就部署相应的 Web 应用。

因此 HostConfig 做的事情都是比较“宏观”的，它不会去检查具体类文件或者资源文件是否有变化，而是检查 Web 应用目录级别的变化。

本期精华

今天我们学习 Tomcat 的热加载和热部署，它们的目的是在不重启 Tomcat 的情况下实现 Web 应用的更新。

热加载的粒度比较小，主要是针对类文件的更新，通过创建新的类加载器来实现重新加载。而热部署是针对整个 Web

应用的，Tomcat 会将原来的 Context 对象整个销毁掉，再重新创建 Context 容器对象。

热加载和热部署的实现都离不开后台线程的周期性检查，Tomcat 在基类 ContainerBase 中统一实现了后台线程的处理逻辑，并在顶层容器 Engine 启动后台线程，这样子容器组件甚至各种通用组件都不需要自己去创建后台线程，这样的设计显得优雅整洁。

课后思考

为什么 Host 容器不通过重写 backgroundProcess 方法来实现热部署呢？

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 22 | 热点问题答疑（2）：内核如何阻塞与唤醒进程？

下一篇 24 | Context容器（上）：Tomcat如何打破双亲委托...

精选留言 (10)

写留言



曹宇

2019-07-03

为什么 Host 容器不通过重写 `backgroundProcess` 方法来实现热部署呢？

因为这种方式的传递方向是从父容器到子容器，而HOST容器部署依赖Context容器部署完毕，才能部署应用，也就是先要子容器Context完成热部署后才能Host容器进行部署。所以针对这种情况，提供了周期性事件机制。



W.T

2019-07-02

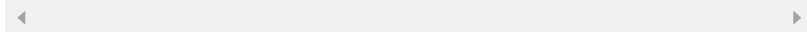
文章读完有两点疑惑：

1、文章提到热部署和热加载的本质是热部署的Context对象整个会被销毁掉，会重新创建一个Context对象；而前文讲热加载时提到WebLoader会调用Context的reload方法，也会销毁Context对象，是否前后矛盾了...

展开 ✓

作者回复: 1，reload只是调用了context组件的方法，没有将这个对象实例销毁掉。

2，父子容器 不是 父子类 的关系。这里的backgroundProccess是ContainerBase实现的各容器的公共逻辑，不是只有父容器才有的行为。

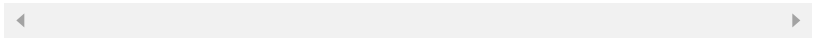


QQ怪

2019-07-02

老师，问个问题，我们平常在开发中，在已经运行的应用上改了代码之后然后编译，是不是就是主动热加载？

作者回复: 对的



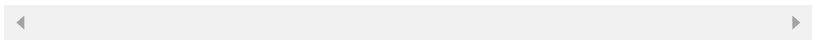
仙道

2019-07-03

能出个如何tomcat源码导入到idea 以及如何调试的教程吗

展开 ▾

作者回复: 建议看13答疑篇



nightmare

2019-07-03

Host热部署是父容器，如果也采用定时任务，那么context都重新加载了，也就没有热加载什么事情了



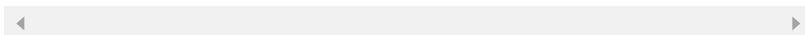


-W.LI-
2019-07-02

老师好!是因为基于事件，可以解耦么？

展开 ∨

作者回复: 这是其中一个原因，只要还是Host的周期性任务比较简单，只要检查部署是否有更新；而Context组件那样，周期性任务比较复杂，不得不重写父类的方法。



nightmare

2019-07-02

热加载的定位是某些文件更新，热部署是通过用户手动加了一个项目，热部署没有必要定时执行，只需要监听用户的热部署事件就行



刘章周

2019-07-02

backgroundProcess 方法主要用在加载粒度比较小，加载具体的资源、文件。而热部署是直接销毁Context,然后重新部署，所以不适合。





黎

2019-07-02

热加载这种设计感觉就是一个树结构，从根开始遍历执行到所有叶，学习了。



Liam

2019-07-02

热部署也会导致系统短暂地不可用吧？我理解它只不过是
通过后台线程周期性任务实现了一下自动重启功能？

作者回复: 对的，部署过程中不能访问

