

26 | Context容器（下）：Tomcat如何实现Servlet规范？

2019-07-09 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



讲述：李号双

时长 09:21 大小 7.50M



我们知道，Servlet 容器最重要的任务就是创建 Servlet 的实例并且调用 Servlet，在前面两期我谈到了 Tomcat 如何定义自己的类加载器来加载 Servlet，但加载 Servlet 的类不等于创建 Servlet 的实例，类加载只是第一步，类加载好了才能创建类的实例，也就是说 Tomcat 先加载 Servlet 的类，然后在 Java 堆上创建了一个 Servlet 实例。

一个 Web 应用里往往有多个 Servlet，而在 Tomcat 中一个 Web 应用对应一个 Context 容器，也就是说一个 Context 容器需要管理多个 Servlet 实例。但 Context 容器并不直接持有 Servlet 实例，而是通过子容器 Wrapper 来管理 Servlet，你可以把 Wrapper 容器看作是 Servlet 的包装。

那为什么需要 Wrapper 呢？Context 容器直接维护一个 Servlet 数组不就行了吗？这是因为 Servlet 不仅仅是一个类实例，它还有相关的配置信息，比如它的 URL 映射、它的初始

化参数，因此设计出了一个包装器，把 Servlet 本身和它相关的数据包起来，没错，这就是面向对象的思想。

那管理好 Servlet 就完事大吉了吗？别忘了 Servlet 还有两个兄弟：Listener 和 Filter，它们也是 Servlet 规范中的重要成员，因此 Tomcat 也需要创建它们的实例，也需要在合适的时机去调用它们的方法。

说了那么多，下面我们就来聊一聊是 Tomcat 如何做到上面这些事的。

Servlet 管理

前面提到，Tomcat 是用 Wrapper 容器来管理 Servlet 的，那 Wrapper 容器具体长什么样子呢？我们先来看看它里面有哪些关键的成员变量：

 复制代码

```
1 protected volatile Servlet instance = null;
```

毫无悬念，它拥有一个 Servlet 实例，并且 Wrapper 通过 loadServlet 方法来实例化 Servlet。为了方便你阅读，我简化了代码：

 复制代码

```
1 public synchronized Servlet loadServlet() throws ServletException {
2     Servlet servlet;
3
4     //1. 创建一个 Servlet 实例
5     servlet = (Servlet) instanceManager.newInstance(servletClass);
6
7     //2. 调用了 Servlet 的 init 方法，这是 Servlet 规范要求的
8     initServlet(servlet);
9
10    return servlet;
11 }
```


其实 loadServlet 主要做了两件事：创建 Servlet 的实例，并且调用 Servlet 的 init 方法，因为这是 Servlet 规范要求的。

那接下来的问题是，什么时候会调到这个 `loadServlet` 方法呢？为了加快系统的启动速度，我们往往会采取资源延迟加载的策略，Tomcat 也不例外，默认情况下 Tomcat 在启动时不会加载你的 Servlet，除非你把 Servlet 的 `loadOnStartup` 参数设置为 `true`。

这里还需要你注意的是，虽然 Tomcat 在启动时不会创建 Servlet 实例，但是会创建 Wrapper 容器，就好比尽管枪里面还没有子弹，先把枪造出来。那子弹什么时候造呢？是真正需要开枪的时候，也就是说有请求来访问某个 Servlet 时，这个 Servlet 的实例才会被创建。

那 Servlet 是被谁调用的呢？我们回忆一下专栏前面提到过 Tomcat 的 Pipeline-Valve 机制，每个容器组件都有自己的 Pipeline，每个 Pipeline 中有一个 Valve 链，并且每个容器组件有一个 BasicValve（基础阀）。Wrapper 作为一个容器组件，它也有自己的 Pipeline 和 BasicValve，Wrapper 的 BasicValve 叫 **StandardWrapperValve**。

你可以想到，当请求到来时，Context 容器的 BasicValve 会调用 Wrapper 容器中 Pipeline 中的第一个 Valve，然后会调用到 StandardWrapperValve。我们先来看看它的 `invoke` 方法是如何实现的，同样为了方便你阅读，我简化了代码：

 复制代码

```
1 public final void invoke(Request request, Response response) {
2
3     //1. 实例化 Servlet
4     servlet = wrapper.allocate();
5
6     //2. 给当前请求创建一个 Filter 链
7     ApplicationFilterChain filterChain =
8         ApplicationFilterFactory.createFilterChain(request, wrapper, servlet);
9
10    //3. 调用这个 Filter 链，Filter 链中的最后一个 Filter 会调用 Servlet
11    filterChain.doFilter(request.getRequest(), response.getResponse());
12
13 }
```

StandardWrapperValve 的 `invoke` 方法比较复杂，去掉其他异常处理的一些细节，本质上就是三步：

第一步，创建 Servlet 实例；

第二步，给当前请求创建一个 Filter 链；

第三步，调用这个 Filter 链。


你可能会问，为什么需要给每个请求创建一个 Filter 链？这是因为每个请求的请求路径都不一样，而 Filter 都有相应的路径映射，因此不是所有的 Filter 都需要来处理当前的请求，我们需要根据请求的路径来选择特定的一些 Filter 来处理。

第二个问题是，为什么没有看到调到 Servlet 的 service 方法？这是因为 Filter 链的 doFilter 方法会负责调用 Servlet，具体来说就是 Filter 链中的最后一个 Filter 会负责调用 Servlet。

接下来我们来看 Filter 的实现原理。


Filter 管理

我们知道，跟 Servlet 一样，Filter 也可以在 web.xml 文件里进行配置，不同的是，Filter 的作用域是整个 Web 应用，因此 Filter 的实例是在 Context 容器中进行管理的，Context 容器用 Map 集合来保存 Filter。

 复制代码

```
1 private Map<String, FilterDef> filterDefs = new HashMap<>();
```

那上面提到的 Filter 链又是什么呢？Filter 链的存活期很短，它是跟每个请求对应的。一个新的请求来了，就动态创建一个 Filter 链，请求处理完了，Filter 链也就被回收了。理解它的原理也非常关键，我们还是来看看源码：

 复制代码

```
1 public final class ApplicationFilterChain implements FilterChain {
2
3     //Filter 链中有 Filter 数组，这个好理解
4     private ApplicationFilterConfig[] filters = new ApplicationFilterConfig[0];
5
6     //Filter 链中的当前的调用位置
7     private int pos = 0;
8
9     // 总共有多少了 Filter
10    private int n = 0;
```

```
11
12 // 每个 Filter 链对应一个 Servlet，也就是它要调用的 Servlet
13 private Servlet servlet = null;
14
15 public void doFilter(ServletRequest req, ServletResponse res) {
16     internalDoFilter(request,response);
17 }
18
19 private void internalDoFilter(ServletRequest req,
20                               ServletResponse res){
21
22     // 每个 Filter 链在内部维护了一个 Filter 数组
23     if (pos < n) {
24         ApplicationFilterConfig filterConfig = filters[pos++];
25         Filter filter = filterConfig.getFilter();
26
27         filter.doFilter(request, response, this);
28         return;
29     }
30
31     servlet.service(request, response);
32
33 }
```

从 ApplicationFilterChain 的源码我们可以看到几个关键信息：

1. Filter 链中除了有 Filter 对象的数组，还有一个整数变量 pos，这个变量用来记录当前被调用的 Filter 在数组中的位置。
2. Filter 链中有个 Servlet 实例，这个好理解，因为上面提到了，每个 Filter 链最后都会调用到一个 Servlet。
3. Filter 链本身也实现了 doFilter 方法，直接调用了一个内部方法 internalDoFilter。
4. internalDoFilter 方法的实现比较有意思，它做了一个判断，如果当前 Filter 的位置小于 Filter 数组的长度，也就是说 Filter 还没调完，就从 Filter 数组拿下一个 Filter，调用它的 doFilter 方法。否则，意味着所有 Filter 都调到了，就调用 Servlet 的 service 方法。

但问题是，方法体里没看到循环，谁在不停地调用 Filter 链的 doFilter 方法呢？Filter 是怎么依次调到的呢？

答案是**Filter 本身的 doFilter 方法会调用 Filter 链的 doFilter 方法**，我们还是来看看代码就明白了：

```
1 public void doFilter(ServletRequest request, ServletResponse response,
2     FilterChain chain){
3
4     ...
5
6     // 调用 Filter 的方法
7     chain.doFilter(request, response);
8
9 }
```

注意 Filter 的 doFilter 方法有个关键参数 FilterChain，就是 Filter 链。并且每个 Filter 在实现 doFilter 时，必须要调用 Filter 链的 doFilter 方法，而 Filter 链中保存当前 Filter 的位置，会调用下一个 Filter 的 doFilter 方法，这样链式调用就完成了。

Filter 链跟 Tomcat 的 Pipeline-Valve 本质都是责任链模式，但是在具体实现上稍有不同，你可以细细体会一下。

Listener 管理

我们接着聊 Servlet 规范里 Listener。跟 Filter 一样，Listener 也是一种扩展机制，你可以监听容器内部发生的事件，主要有两类事件：

第一类是生命状态的变化，比如 Context 容器启动和停止、Session 的创建和销毁。

第二类是属性的变化，比如 Context 容器某个属性值变了、Session 的某个属性值变了以及新的请求来了等。

我们可以在 web.xml 配置或者通过注解的方式来添加监听器，在监听器里实现我们的业务逻辑。对于 Tomcat 来说，它需要读取配置文件，拿到监听器类的名字，实例化这些类，并且在合适的时机调用这些监听器的方法。


Tomcat 是通过 Context 容器来管理这些监听器的。Context 容器将两类事件分开来管理，分别用不同的集合来存放不同类型事件的监听器：

```
1 // 监听属性值变化的监听器
2 private List<Object> applicationEventListenersList = new CopyOnWriteArrayList<>();
```



```
3
4 // 监听生命事件的监听器
5 private Object applicationLifecycleListenersObjects[] = new Object[0];
```

剩下的事情就是触发监听器了，比如在 Context 容器的启动方法里，就触发了所有的 ServletContextListener：

 复制代码

```
1 //1. 拿到所有的生命周期监听器
2 Object instances[] = getApplicationLifecycleListeners();
3
4 for (int i = 0; i < instances.length; i++) {
5     //2. 判断 Listener 的类型是不是 ServletContextListener
6     if (!(instances[i] instanceof ServletContextListener))
7         continue;
8
9     //3. 触发 Listener 的方法
10    ServletContextListener lr = (ServletContextListener) instances[i];
11    lr.contextInitialized(event);
12 }
```

需要注意的是，这里的 ServletContextListener 接口是一种留给用户的扩展机制，用户可以实现这个接口来定义自己的监听器，监听 Context 容器的启停事件。Spring 就是这么做的。ServletContextListener 跟 Tomcat 自己的生命周期事件 LifecycleListener 是不同的。LifecycleListener 定义在生命周期管理组件中，由基类 LifeCycleBase 统一管理。

本期精华

Servlet 规范中最重要的就是 Servlet、Filter 和 Listener “三兄弟”。Web 容器最重要的职能就是把它们创建出来，并在适当的时候调用它们的方法。

Tomcat 通过 Wrapper 容器来管理 Servlet，Wrapper 包装了 Servlet 本身以及相应的参数，这体现了面向对象中“封装”的设计原则。

Tomcat 会给**每个请求生成一个 Filter 链**，Filter 链中的最后一个 Filter 会负责调用 Servlet 的 service 方法。

对于 Listener 来说，我们可以定制自己的监听器来监听 Tomcat 内部发生的各种事件：包括 Web 应用级别的、Session 级别的和请求级别的。Tomcat 中的 Context 容器统一维护了这些监听器，并负责触发。

最后小结一下这 3 期内容，Context 组件通过自定义类加载器来加载 Web 应用，并实现了 Servlet 规范，直接跟 Web 应用打交道，是一个核心的容器组件。也因此我用了很重的篇幅去讲解它，也非常建议你花点时间阅读一下它的源码。

课后思考

Context 容器分别用了 CopyOnWriteArrayList 和对象数组来存储两种不同的监听器，为什么要这样设计，你可以思考一下背后的原因。

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 25 | Context容器（中）：Tomcat如何隔离Web应用？

下一篇 27 | 新特性：Tomcat 9.0.0 中的新特性

精选留言 (7)

写留言



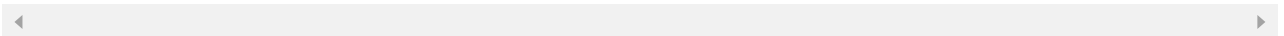
-W.LI-

2019-07-09

属性值变化listener能动态配置，所以用CopyOnWriteArray。生命周期事件listener，不能动态改变没有线程安全问题？

展开 ▾

作者回复: 说的对



4



nightmare

2019-07-09

生命周期相关的类比如session一个用户分配一个，用完了就会销毁，用对象数组，可以适应增删改操作，而属性变化，写不会那么频繁，读取比较频繁

展开 ▾



2



Monday

2019-07-11

思考题两种数据结构我能区分开来，但是还是回答不上来。。。



1



z.l

2019-07-12

反向推导猜下，生命状态变化都在单线程中，没有并发问题。属性变化有线程安全问题，但又不频繁。



飞翔

2019-07-09

老师我需要手动加一个filter，不能用annotation或是xml、我是不是需要调用standardcontext里边的addfilterdef和addfiltermap这两个函数就行了？

展开 ▾



非想

2019-07-09

老师您好，看你的文章servlet容器中的三个组件servlet，filter，listener都是由context容器管理的对吗？

作者回复: filter是wrapper组件管理的



Liam

2019-07-09

我记得pip-val是用单向链表实现的，filters chain是用数组实现的；另外，pipe节点自己负责调用下一个节点，而filter则是委派chain来调度

展开 ▾

