

07 | Tomcat如何实现一键式启停？

2019-05-25 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)

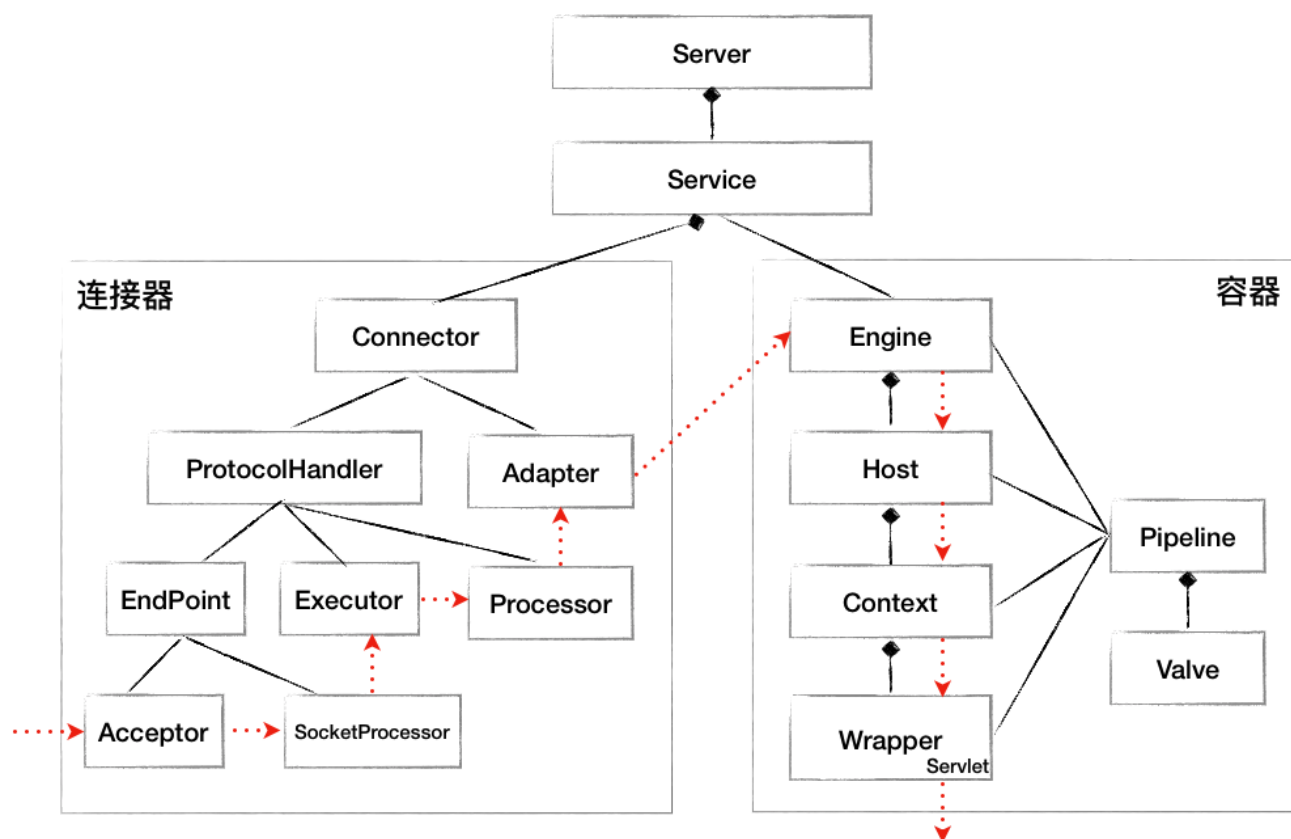


讲述：李号双

时长 11:12 大小 10.27M



通过前面的学习，相信你对 Tomcat 的架构已经有所了解，知道了 Tomcat 都有哪些组件，组件之间是什么样的关系，以及 Tomcat 是怎么处理一个 HTTP 请求的。下面我们通过一张简化的类图来回顾一下，从图上你可以看到各种组件的层次关系，图中的虚线表示一个请求在 Tomcat 中流转的过程。



上面这张图描述了组件之间的静态关系，如果想让一个系统能够对外提供服务，我们需要创建、组装并启动这些组件；在服务停止的时候，我们还需要释放资源，销毁这些组件，因此这是一个动态的过程。也就是说，Tomcat 需要动态地管理这些组件的生命周期。

在我们实际的工作中，如果你需要设计一个比较大的系统或者框架时，你同样也需要考虑这几个问题：如何统一管理组件的创建、初始化、启动、停止和销毁？如何做到代码逻辑清晰？如何方便地添加或者删除组件？如何做到组件启动和停止不遗漏、不重复？

今天我们就来解决上面的问题，在这之前，先来看看组件之间的关系。如果你仔细分析过这些组件，可以发现它们具有两层关系。

第一层关系是组件有大有小，大组件管理小组件，比如 Server 管理 Service，Service 又管理连接器和容器。

第二层关系是组件有外有内，外层组件控制内层组件，比如连接器是外层组件，负责对外交流，外层组件调用内层组件完成业务功能。也就是说，**请求的处理过程是由外层组件来驱动的。**

这两层关系决定了系统在创建组件时应该遵循一定的顺序。

第一个原则是先创建子组件，再创建父组件，子组件需要被“注入”到父组件中。

第二个原则是先创建内层组件，再创建外层组件，内层组建需要被“注入”到外层组件。

因此，最直观的做法就是将图上所有的组件按照先小后大、先内后外的顺序创建出来，然后组装在一起。不知道你注意到没有，这个思路其实很有问题！因为这样不仅会造成代码逻辑混乱和组件遗漏，而且也不利于后期的功能扩展。

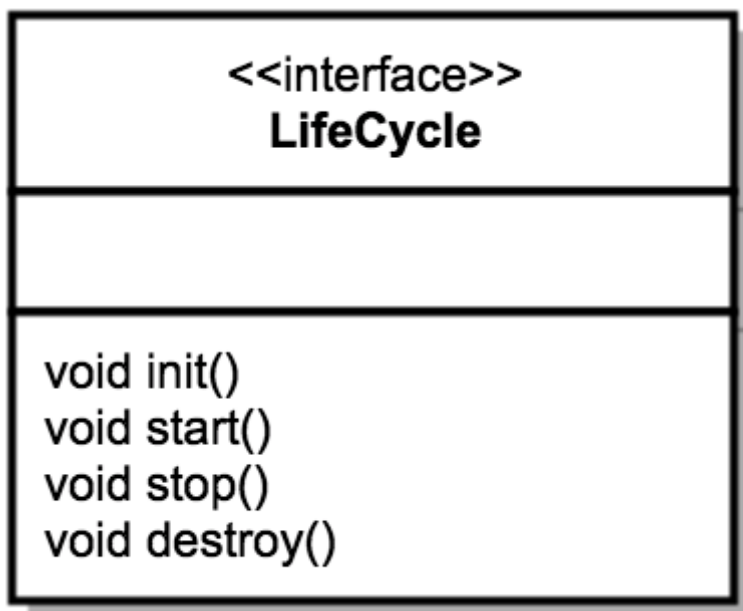
为了解决这个问题，我们希望找到一种通用的、统一的方法来管理组件的生命周期，就像汽车“一键启动”那样的效果。

一键式启停：LifeCycle 接口

我在前面说到过，设计就是要找到系统的变化点和不变点。这里的不变点就是每个组件都要经历创建、初始化、启动这几个过程，这些状态以及状态的转化是不变的。而变化点是每个具体组件的初始化方法，也就是启动方法是不一样的。

因此，我们把不变点抽象出来成为一个接口，这个接口跟生命周期有关，叫作 LifeCycle。LifeCycle 接口里应该定义这么几个方法：init()、start()、stop() 和 destroy()，每个具体的组件去实现这些方法。

理所当然，在父组件的 init() 方法里需要创建子组件并调用子组件的 init() 方法。同样，在父组件的 start() 方法里也需要调用子组件的 start() 方法，因此调用者可以无差别的调用各组件的 init() 方法和 start() 方法，这就是**组合模式**的使用，并且只要调用最顶层组件，也就是 Server 组件的 init() 和 start() 方法，整个 Tomcat 就被启动起来了。下面是 LifeCycle 接口的定义。

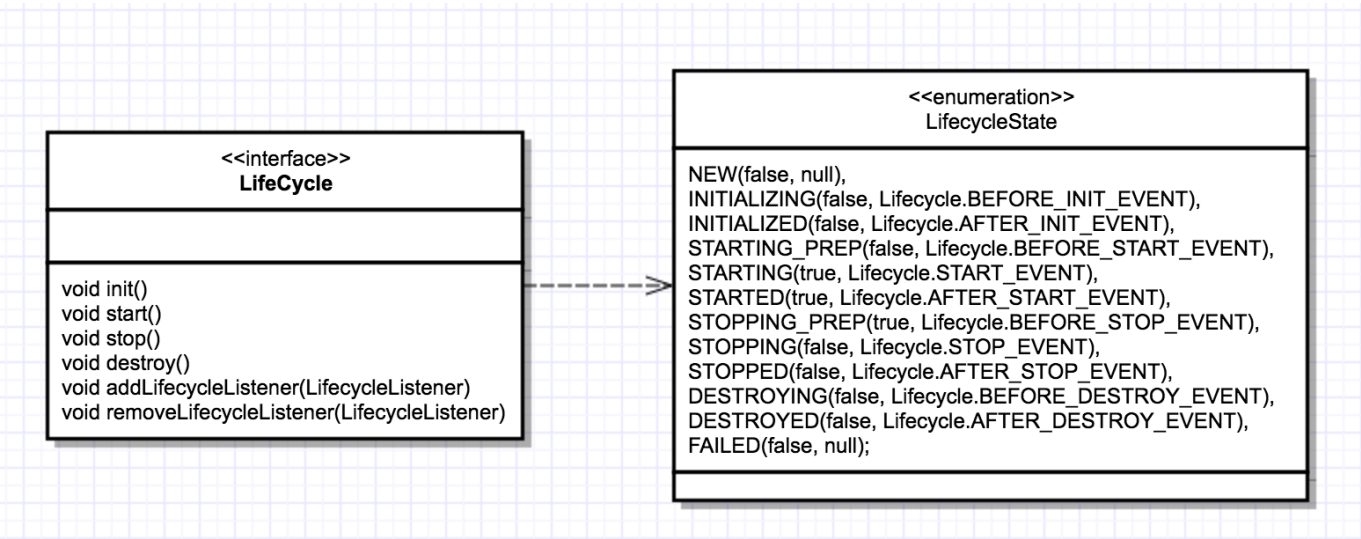


可扩展性：LifeCycle 事件

我们再来考虑另一个问题，那就是系统的可扩展性。因为各个组件 `init()` 和 `start()` 方法的具体实现是复杂多变的，比如在 Host 容器的启动方法里需要扫描 webapps 目录下的 Web 应用，创建相应的 Context 容器，如果将来需要增加新的逻辑，直接修改 `start()` 方法？这样会违反开闭原则，那如何解决这个问题呢？开闭原则说的是为了扩展系统的功能，你不能直接修改系统中已有的类，但是你可以定义新的类。

我们注意到，组件的 `init()` 和 `start()` 调用是由它的父组件的状态变化触发的，上层组件的初始化会触发子组件的初始化，上层组件的启动会触发子组件的启动，因此我们把组件的生命周期定义成一个个状态，把状态的转变看作是一个事件。而事件是有监听器的，在监听器里可以实现一些逻辑，并且监听器也可以方便的添加和删除，这就是典型的**观察者模式**。

具体来说就是在 LifeCycle 接口里加入两个方法：添加监听器和删除监听器。除此之外，我们还需要定义一个 Enum 来表示组件有哪些状态，以及处在什么状态会触发什么样的事件。因此 LifeCycle 接口和 LifecycleState 就定义成了下面这样。



从图上你可以看到，组件的生命周期有 NEW、INITIALIZING、INITIALIZED、STARTING_PREP、STARTING、STARTED 等，而一旦组件到达相应的状态就触发相应的事件，比如 NEW 状态表示组件刚刚被实例化；而当 `init()` 方法被调用时，状态就变成 INITIALIZING 状态，这个时候，就会触发 BEFORE_INIT_EVENT 事件，如果有监听器在监听这个事件，它的方法就会被调用。

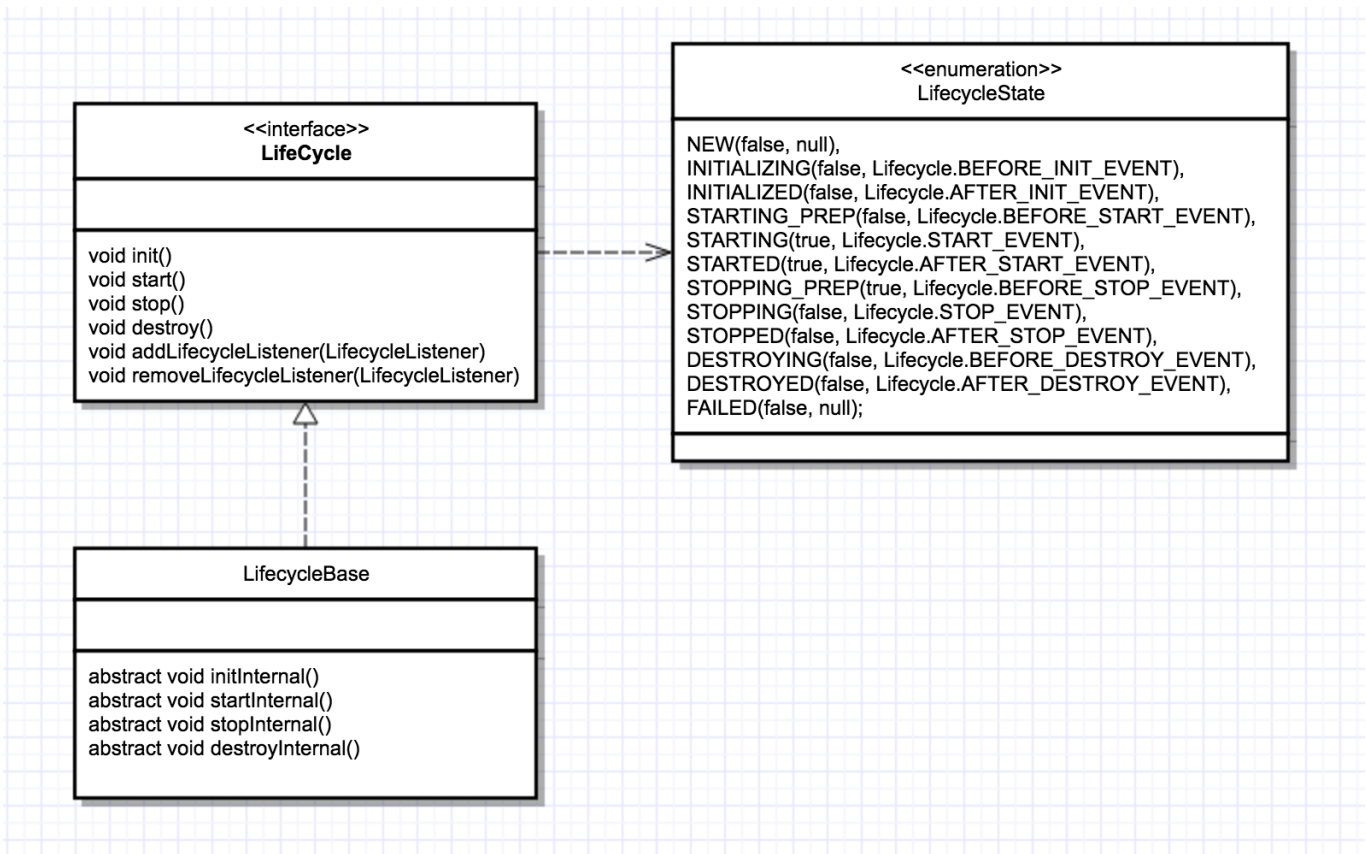
重用性：LifecycleBase 抽象基类

有了接口，我们就要用类去实现接口。一般来说实现类不止一个，不同的类在实现接口时往往会有一些相同的逻辑，如果让各个子类都去实现一遍，就会有重复代码。那子类如何重用这部分逻辑呢？其实就是定义一个基类来实现共同的逻辑，然后让各个子类去继承它，就达到了重用的目的。

而基类中往往会定义一些抽象方法，所谓的抽象方法就是说基类不会去实现这些方法，而是调用这些方法来实现骨架逻辑。抽象方法是留给各个子类去实现的，并且子类必须实现，否则无法实例化。


比如宝马和荣威的底盘和骨架其实是一样的，只是发动机和内饰等配套是不一样的。底盘和骨架就是基类，宝马和荣威就是子类。仅仅有底盘和骨架还不是一辆真正意义上的车，只能算是半成品，因此在底盘和骨架上会留出一些安装接口，比如安装发动机的接口、安装座椅的接口，这些就是抽象方法。宝马或者荣威上安装的发动机和座椅是不一样的，也就是具体子类对抽象方法有不同的实现。

回到 LifeCycle 接口，Tomcat 定义一个基类 LifeCycleBase 来实现 LifeCycle 接口，把一些公共的逻辑放到基类中去，比如生命状态的转变与维护、生命事件的触发以及监听器的添加和删除等，而子类就负责实现自己的初始化、启动和停止等方法。为了避免跟基类中的方法同名，我们把具体子类的实现方法改个名字，在后面加上 Internal，叫 initInternal()、startInternal() 等。我们再来看引入了基类 LifeCycleBase 后的类图：



从图上可以看到，LifeCycleBase 实现了 LifeCycle 接口中所有的方法，还定义了相应的抽象方法交给具体子类去实现，这是典型的**模板设计模式**。

我们还是看一看代码，可以帮你加深理解，下面是 LifeCycleBase 的 init() 方法实现。

 复制代码

```
1 @Override
2 public final synchronized void init() throws LifecycleException {
3     //1. 状态检查
4     if (!state.equals(LifecycleState.NEW)) {
5         invalidTransition(Lifecycle.BEFORE_INIT_EVENT);
6     }
7
8     try {
9         //2. 触发 INITIALIZING 事件的监听器
10        setStateInternal(LifecycleState.INITIALIZING, null, false);
11
12        //3. 调用具体子类的初始化方法
13        initInternal();
14
15        //4. 触发 INITIALIZED 事件的监听器
16        setStateInternal(LifecycleState.INITIALIZED, null, false);
17    } catch (Throwable t) {
18        ...
19    }
20 }
```

这个方法逻辑比较清楚，主要完成了四步：

第一步，检查状态的合法性，比如当前状态必须是 NEW 然后才能进行初始化。

第二步，触发 INITIALIZING 事件的监听器：


 复制代码

```
1 setStateInternal(LifecycleState.INITIALIZING, null, false);
```

在这个 setStateInternal 方法里，会调用监听器的业务方法。

第三步，调用具体子类实现的抽象方法 `initInternal()` 方法。我在前面提到过，为了实现一键式启动，具体组件在实现 `initInternal()` 方法时，又会调用它的子组件的 `init()` 方法。

第四步，子组件初始化后，触发 `INITIALIZED` 事件的监听器，相应监听器的业务方法就会被调用。

 复制代码

```
1 setStateInternal(LifecycleState.INITIALIZED, null, false);
```

总之，`LifeCycleBase` 调用了抽象方法来实现骨架逻辑。讲到这里，你可能好奇，`LifeCycleBase` 负责触发事件，并调用监听器的方法，那是什么时候、谁把监听器注册进来的呢？

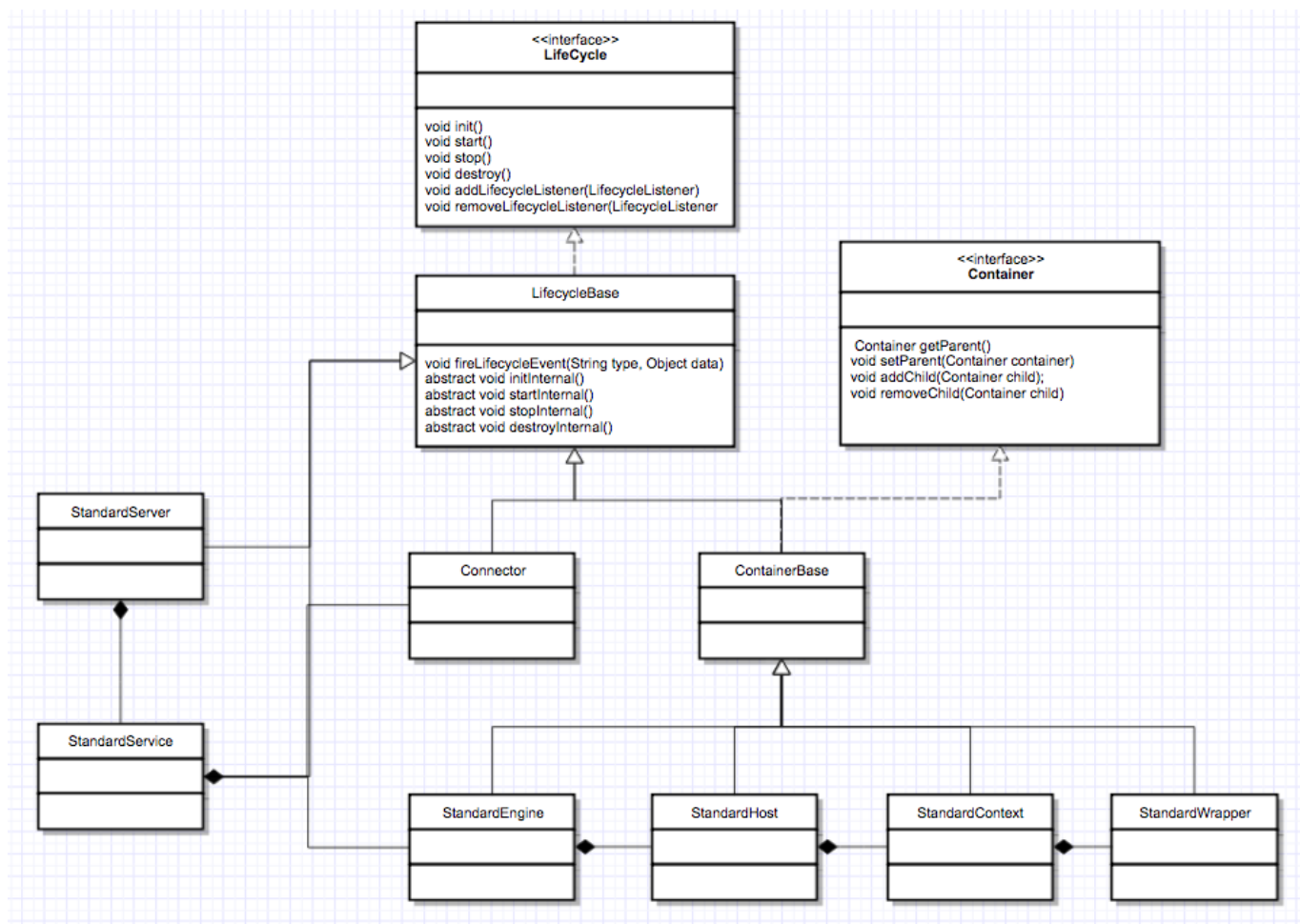
分为两种情况：

Tomcat 自定义了一些监听器，这些监听器是父组件在创建子组件的过程中注册到子组件的。比如 `MemoryLeakTrackingListener` 监听器，用来检测 `Context` 容器中的内存泄漏，这个监听器是 `Host` 容器在创建 `Context` 容器时注册到 `Context` 中的。

我们还可以在 `server.xml` 中定义自己的监听器，Tomcat 在启动时会解析 `server.xml`，创建监听器并注册到容器组件。

生命周期管理总体类图

通过上面的学习，我相信你对 Tomcat 组件的生命周期的管理有了深入的理解，我们再来看一张总体类图继续加深印象。



这里请你注意，图中的 StandardServer、StandardService 等是 Server 和 Service 组件的具体实现类，它们都继承了 LifecycleBase。

StandardEngine、StandardHost、StandardContext 和 StandardWrapper 是相应容器组件的具体实现类，因为它们都是容器，所以继承了 ContainerBase 抽象基类，而 ContainerBase 实现了 Container 接口，也继承了 LifecycleBase 类，它们的生命周期管理接口和功能接口是分开的，这也符合设计中**接口分离的原则**。

本期精华

Tomcat 为了实现一键式启停以及优雅的生命周期管理，并考虑到了可扩展性和可重用性，将面向对象思想和设计模式发挥到了极致，分别运用了**组合模式**、**观察者模式**、**骨架抽象类**和**模板方法**。

如果你需要维护一堆具有父子关系的实体，可以考虑使用组合模式。

观察者模式听起来“高大上”，其实就是当一个事件发生后，需要执行一连串更新操作。传统的实现方式是在事件响应代码里直接加更新逻辑，当更新逻辑加多了之后，代码会变得臃

肿，并且这种方式是紧耦合的、侵入式的。而观察者模式实现了低耦合、非侵入式的通知与更新机制。

而模板方法在抽象基类中经常用到，用来实现通用逻辑。

课后思考

从文中最后的类图上你会看到所有的容器组件都扩展了 ContainerBase，跟 LifecycleBase 一样，ContainerBase 也是一个骨架抽象类，请你思考一下，各容器组件有哪些“共同的逻辑”需要 ContainerBase 由来实现呢？

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，欢迎你把它的分享给你的朋友。



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 06 | Tomcat系统架构（下）：聊聊多层容器的设计

下一篇 08 | Tomcat的“高层们”都负责做什么？

精选留言 (24)

写留言



allean

2019-05-25

11

原理理解之后特别想看看源码是怎么写的，不看源码总感觉不踏实🤔，老师在介绍组件原理之后可不可以指明怎么启动Tomcat源码，并debug啊，多谢

作者回复: 建议跟SpringBoot那样，用嵌入式方式启动Tomcat，这里有例子：

<https://github.com/heroku/devcenter-embedded-tomcat>



一路远行

2019-05-26

10

ContainerBase提供了针对Container接口的通用实现，所以最重要的职责包含两个：

- 1) 维护容器通用的状态数据
- 2) 提供管理状态数据的通用方法

容器的关键状态信息和方法有：...

展开 ▾

作者回复: 🙏



刘冬

2019-05-25

5

老师太拼了，周末凌晨发新的课程。太感动了🐱

展开 ▾



Monday

2019-05-25

4

思考题

- 1, 容器的创建/初始化/销毁
- 2, 容器添加/删除子容器
- 3, 如果还要监听容器状态变化的话还需要有添加/移除事件的方法。

请指正。...

展开 ▾

作者回复: 你思考题回答的不错啊, 还可以想想实际工作中是不是也有类似的场景, 可以模仿这种设计。



yi_jun

2019-05-27

👍 3

看了这篇文章对tomcat中用到的设计模式又有了新的理解.

看到评论里有问tomcat启动的同学, 和大家分享一篇介绍Tomcat启动的文章, 从startup.bat的源码开始分析的.

<https://www.cnblogs.com/tanshaoshenghao/p/10932306.html>



nsn_huang

2019-05-27

👍 3

这是Tomcat8.5版本的源码, 基于Maven和IDEA, 希望大家一起学习, 一起进步。

https://blog.csdn.net/sougou_1323/article/details/90597079

作者回复: 谢谢分享!



-W.LI-

2019-05-25

👍 2

老师好!之前设计模式看过好几遍, 总感觉用不上, 虽然知道是自己对设计思想的理解不够深入导致的。又苦于找不到方法, 看了老师的分析对设计模式, 和设计原则又有了进一步的了解。问题1:对于变于不变的界定标准, 哪些方法需要抽象为接口。老师有啥好的建议么。问题2:spring的event, 发布一个事件时会把事件放入map.然后轮训所有的所有的观察者。观察者和event很多的时候, 内存等开销, 会成为性能瓶颈么?比如处理事件的逻辑...

展开 ▾

作者回复: 1.这个要根据具体场景来的, 简单来说如果你需要用if-else来实现某个逻辑, 这是可能是变化点。

2.这种情况下, 对event的处理考虑用线程池。



why

2019-05-30

👍 1

- 系统启动时会创建，组装，启动组件，服务停止时释放资源销毁组件。
- 组件间由两层关系
 - 组件有大有小, 大组件管理小组件(Server 管 Service)
 - 组件有内有外, 外层控制内层(连接器控制容器)
- 关系决定创建应有的顺序...

展开 ▾

作者回复: 👍



刘冬

2019-05-26

👍 1

强烈呼吁老师能够讲解一下，怎么启动Tomcat的源码、调试。怎么读源码。
另外，有些Interface的实现是在单独的Jar中，用IntelliJ无法直接看到Implementation，请问有什么好的办法看到这部分的源码吗？

展开 ▾

作者回复: 建议用嵌入式的方式来启动Tomcat。这里有个例子：

<https://github.com/heroku/devcenter-embedded-tomcat>



QQ怪

2019-05-26

👍 1

还好订阅老师专栏之前看完大话设计模式，正好根据tomcat原理来体会设计模式的精髓，不然都会看不懂啊，哈哈哈

作者回复: ☺



K.Zhou

👍 1



2019-05-25

这篇干货十足，几种设计模式的经典实际运用！

展开 ∨



木木木

2019-06-02



目前找到的debug资料都是tomcat8的，老师提供的例子也是tomcat8的，有没有基于新代码的呢，目前也是github上拉了最新的代码，试着构建了下。一个是debug tomcat本身，还有就是部署的应用的，两个应用不一样的吧。

展开 ∨

作者回复: 你可以改下pom.xml里面Tomcat的版本，改成最新的9.0.20把。

用Embedded方式把应用跑起来后，IDE有直接下载源码的功能，在Tomcat源码中下断点都不是问题。



疯狂咸鱼

2019-06-02



老师，会不会考虑出一门课：深入拆解spring?我特别想听老师讲Spring架构的设计，和里面用到的设计模式

作者回复: 掌握了分析源码的方法，再分析Spring都不是难事。



Geek_00d56...

2019-05-29



Tomcat 为了实现一键式启停以及优雅的生命周期管理，并考虑到了可扩展性和可重用性，将面向对象思想和设计模式发挥到了极致，分别运用了组合模式、观察者模式、骨架抽象类和模板方法。

展开 ∨



WL

2019-05-27



请问一下老师我看LifecycleMBeanBase中的initInternal()方法想问下这个方法里面的Registry是起到什么作用, 我不太理解?

作者回复: 将组件注册到mbean server, 用JMX管起来



轩

2019-05-27



想debug Tomcat源码的同学可以参考下这边博文。

<https://www.cnblogs.com/hjy9420/p/5017583.html>



Mr.差不多

2019-05-27



您好, 老师, 我想问一下Tomcat是如何通过start.sh 来启动项目的, 换句话说就是怎么找到了Bootstrap.java类进行启动的。

作者回复: startup.sh -> catalina.sh

catalina.sh里执行了Bootstrap的main方法



吖蒲

2019-05-26



Tomcat生命周期的三种设计模式运用

①.基于LifecycleBase抽象骨架类实现的子类, 使用组合模式管理起来, 只需要调用一个顶层组件, 即可对所有子类进行管理。

②.使用观察者模式可跟踪顶层组件的状态改变, 通知子类更新生命周期的状态。

③.继承LifecycleBase的子类实现的initInternal()、startInternal()等方法, 根据步骤②获...

展开 ∨

作者回复: ☺



WL

2019-05-26



请问一下老师Tomcat为什么要在LifecycleBase中定义一系列的***internal()让子类取调用, 为什么不是子类实现接口的init()方法呢, 这一点我不是很理解, 希望老师指点一下。

展开 ▾

作者回复: init方法里有一些”通用的逻辑“对各个子类都适用,如果让每个子类都重复实现一遍,代码不久重复了吗,于是这些”逻辑“让LifeCycleBase来做,其实就是实现了模板,子类再在这个模板上填充自己的内容。



流光

2019-05-26



应该有: 除了具体实现类的通用处理逻辑以外的方法,可以用模板方法进行定义通用实现算法,如处理前的方法,处理后的方法. 还有下级节点的注册方法