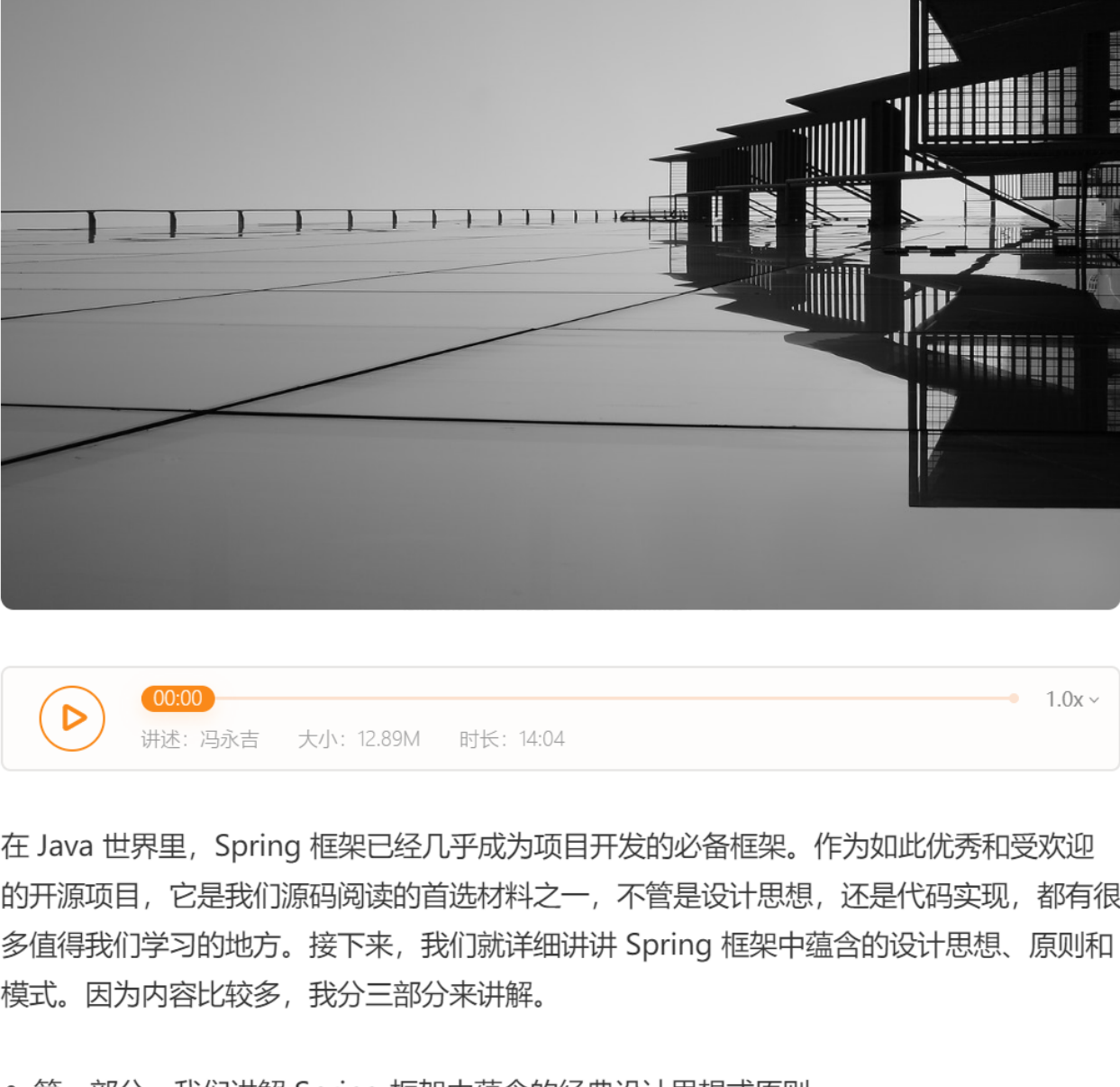
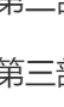



84 | 开源实战四（上）：剖析Spring框架中蕴含的经典设计思想或原则

王争 2020-05-15



00:00

讲述：冯永吉 大小：12.89M 时长：14:04

1.0x 

在 Java 世界里，Spring 框架已经几乎成为项目开发的必备框架。作为如此优秀和受欢迎的开源项目，它是我们源码阅读的首选材料之一，不管是设计思想，还是代码实现，都有很多值得我们学习的地方。接下来，我们就详细讲讲 Spring 框架中蕴含的设计思想、原则和模式。因为内容比较多，我分三部分来讲解。

- 第一部分，我们讲解 Spring 框架中蕴含的经典设计思想或原则。
- 第二部分，我们讲解 Spring 框架中用来支持扩展的两种设计模式。
- 第三部分，我们总结罗列 Spring 框架中用到的其他十几种设计模式。

今天，我们就讲下第一部分：Spring 框架中蕴含的一些设计思想或原则，这其中就包括：约定大于配置、低侵入松耦合、模块化轻量级等。这些设计思想都很通用，掌握之后，我们可以借鉴用到其他框架的开发中。


话不多说，让我们正式开始今天的学习吧！

Spring 框架简单介绍

考虑到你可能不熟悉 Spring，我这里对它做下简单介绍。我们常说的 Spring 框架，是指 Spring Framework 基础框架。Spring Framework 是整个 Spring 生态（也被称作 Spring 全家桶）的基石。除了 Spring Framework，Spring 全家桶中还有更多基于 Spring Framework 开发出来的、整合更多功能的框架，比如 Spring Boot、Spring Cloud。

在 Spring 全家桶中，Spring Framework 是最基础、最底层的一部分。它提供了最基础、最核心的 IOC 和 AOP 功能。当然，它包含的功能还不仅如此，还有其他比如事务管理（Transactions）、MVC 框架（Spring MVC）等很多功能。下面这个表格，是我从 Spring 官网上找的，关于 Spring Framework 的功能介绍，你可以大略地看下有个印象。

Core	IoC Container, Events, Resources, i18n, Validation, Data Binding, Type Conversion, SpEL, AOP.
Testing	Mock Objects, TestContext Framework, Spring MVC Test, WebTestClient.
Data Access	Transactions, DAO Support, JDBC, O/R Mapping, XML Marshalling.
Web Servlet	Spring MVC, WebSocket, SockJS, STOMP Messaging.
Web Reactive	Spring WebFlux, WebClient, WebSocket.
Integration	Remoting, JMS, JCA, JMX, Email, Tasks, Scheduling, Caching.
Languages	Kotlin, Groovy, Dynamic Languages.

 极客时间

在 Spring Framework 中，Spring MVC 出镜率很高，经常被单独拎出来使用。它是支持 Web 开发的 MVC 框架，提供了 URL 路由、Session 管理、模板引擎等跟 Web 开发相关的一系列功能。

Spring Boot 是基于 Spring Framework 开发的。它更加专注于微服务开发。之所以名字里带有“Boot”一词，跟它的设计初衷有关。Spring Boot 的设计初衷是快速启动一个项目，利用它可以快速地完成一个项目的开发、部署和运行。Spring Boot 支持的所有功能都是围绕这个初衷设计的，比如：集成很多第三方开发包、简化配置（比如，规约优于配置）、集成内嵌 Web 容器（比如，Tomcat、Jetty）等。

单个的微服务开发，使用 Spring Boot 就足够了，但是，如果要构建整个微服务集群，就需要用到 Spring Cloud 了。Spring Cloud 主要负责微服务集群的服务治理工作，包含很多独立的功能组件，比如 Spring Cloud Sleuth 调用链追踪、Spring Cloud Config 配置中心等。

从 Spring 看框架的作用

如果你使用过一些框架来做开发，你应该能感受到使用框架开发的优势。这里我稍微总结一下。利用框架的好处有：解耦业务和非业务开发、让程序员聚焦在业务开发上；隐藏复杂实现细节、降低开发难度、减少代码 bug；实现代码复用、节省开发时间；规范化标准化项目开发、降低学习和维护成本等等。实际上，如果要用一句话来总结一下的话，那就是简化开发！

对于刚刚的总结，我们再详细解释一下。

相比单纯的 CRUD 业务代码开发，非业务代码开发要更难一些。所以，将一些非业务的通用代码开发为框架，在项目中复用，除了节省开发时间之外，也降低了项目开发的难度。除此之外，框架经过多个项目的多次验证，比起每个项目都重新开发，代码的 bug 会相对少一些。而且，不同的项目使用相同的框架，对于研发人员来说，从一个项目切换到另一个项目的学习成本，也会降低很多。

接下来，我们再拿常见的 Web 项目开发来举例说明一下。

通过在项目中引入 Spring MVC 开发框架，开发一个 Web 应用，我们只需要创建 Controller、Service、Repository 三层类，在其中填写相应的业务代码，然后做些简单的配置，告知框架 Controller、Service、Repository 类之间的调用关系，剩下的非业务相关的工作，比如，对象的创建、组装、管理，请求的解析、封装，URL 与 Controller 之间的映射，都由框架来完成。

不仅如此，如果我们直接引入功能更强大的 Spring Boot，那将应用部署到 Web 容器的工作都省掉了。Spring Boot 内嵌了 Tomcat、Jetty 等 Web 容器。在编写完代码之后，我们用一条命令就能完成项目的部署、运行。

Spring 框架蕴含的设计思想

在 Google Guava 源码讲解中，我们讲到开发通用功能模块的一些比较普适的开发思想，比如产品意识、服务意识、代码质量意识、不要重复造轮子等。今天，我们剖析一下 Spring 框架背后的一些经典设计思想（或开发技巧）。这些设计思想并非 Spring 独有，都比较通用，能借鉴应用在很多通用功能模块的设计开发中。这也是我们学习 Spring 源码的价值所在。

1. 约定优于配置

在使用 Spring 开发的项目中，配置往往会比较复杂、繁琐。比如，我们利用 Spring MVC 来开发 Web 应用，需要配置每个 Controller 类以及 Controller 类中的接口对应的 URL。

如何来简化配置呢？一般来讲，有两种方法，一种是基于注解，另一种是基于约定。这两种配置方式在 Spring 中都有用到。Spring 在最小化配置方面做得淋漓尽致，有很多值得我们借鉴的地方。

基于注解的配置方式，我们在指定类上使用指定的注解，来替代集中的 XML 配置。比如，我们使用 @RequestMapping 注解，在 Controller 类或者接口上，标注对应的 URL；使用 @Transaction 注解表明支持事务等。

基于约定的配置方式，也常叫作“约定优于配置”或者“规约优于配置”（Convention over Configuration）。通过约定的代码结构或者命名来减少配置。说白了，就是提供配置的默认值，优先使用默认值。程序员只需要设置那些偏离约定的配置就可以了。

比如，在 Spring JPA（基于 ORM 框架、JPA 规范的基础上，封装的一套 JPA 应用框架）中，我们约定类名默认跟表名相同，属性名默认跟表字段名相同，String 类型对应数据库中的 varchar 类型，long 类型对应数据库中的 bigint 类型等等。

基于刚刚的约定，代码中定义的 Order 类就对应数据库中的“order”表。只有在偏离这一约定的时候，例如数据库中表命名为“order_info”而非“order”，我们才需要显示地去配置类与表的映射关系（Order 类 -> order_info 表）。

实际上，约定优于配置，很好地体现了“二八法则”。在平时的项目开发中，80% 的配置使用默认配置就可以了，只有 20% 的配置必须用户显式地去设置。所以，基于约定来配置，在没有牺牲配置灵活性的前提下，节省了我們大量编写配置的时间，省掉了很多不动脑子的纯体力劳动，提高了开发效率。除此之外，基于相同的约定来做开发，也减少了项目的学习成本和维护成本。

2. 低侵入、松耦合

框架的侵入性是衡量框架好坏的重要指标。所谓低侵入指的是，框架代码很少耦合在业务代码中。低侵入意味着，当我们要替换一个框架的时候，对原有的业务代码改动会很少。相反，如果一个框架是高度侵入的，代码高度侵入到业务代码中，那替换成另一个框架的成本将非常高，甚至几乎不可能。这也是一些长期维护的老项目，使用的框架、技术比较老旧，又无法更新的一个很重要的原因。

实际上，低侵入是 Spring 框架遵循的一个非常重要的设计思想。

Spring 提供的 IOC 容器，在不需要 Bean 继承任何父类或者实现任何接口的情况下，仅仅通过配置，就能将它们纳入进 Spring 的管理中。如果我们换一个 IOC 容器，也只是重新配置一下就可以了，原有的 Bean 都不需要任何修改。

除此之外，Spring 提供的 AOP 功能，也体现了低侵入的特性。在项目中，对于非业务功能，比如请求日志、数据采点、安全校验、事务等等，我们没必要将它们侵入进业务代码中。因为一旦侵入，这些代码将分散在各个业务代码中，删除、修改的成本就变得很高。而基于 AOP 这种开发模式，将非业务代码集中放到切面中，删除、修改的成本就变得很低了。

3. 模块化、轻量级

我们知道，十几年前，EJB 是 Java 企业级应用的主流开发框架。但是，它非常臃肿、复杂，侵入性、耦合性高，开发、维护和学习成本都不低。所以，为了替代笨重的 EJB，Rod Johnson 开发了一套开源的 Interface21 框架，提供了最基本的 IOC 功能。实际上，Interface21 框架就是 Spring 框架的前身。

但是，随着不断的发展，Spring 现在也不单单只是一个只包含 IOC 功能的小框架了，它显然已经壮大成了一个“平台”或者叫“生态”，包含了各种五花八门的功能。尽管如此，但它也并没有重蹈覆辙，变成一个像 EJB 那样的庞大难用的框架。那 Spring 是怎么做到的呢？

这就要归功于 Spring 的模块化设计思想。我们先看一张图，如下所示，它是 Spring Framework 的模块和分层介绍图。



从图中我们可以看出，Spring 在分层、模块化方面做得非常好。每个模块都只负责一个相对独立的功能。模块之间关系，仅有上层对下层的依赖关系，而同层之间以及下层对上层，几乎没有依赖和耦合。除此之外，在依赖 Spring 的项目中，开发者可以有选择地引入某几个模块，而不会因为需要一个小功能，就被强迫引入整个 Spring 框架。所以，尽管 Spring Framework 包含的模块很多，已经有二十几个，但每个模块都非常轻量级，都可以单独拿来使用。正因如此，到现在，Spring 框架仍然可以被称为是一个轻量级的开发框架。

4. 再封装、再抽象

Spring 不仅仅提供了各种 Java 项目开发的常用功能模块，而且还对市面上主流的中间件、系统的访问类库，做了进一步的封装和抽象，提供了更高层次、更统一的访问接口。

比如，Spring 提供了 spring-data-redis 模块，对 Redis Java 开发类库（比如 Jedis、Lettuce）做了进一步的封装，适配 Spring 的访问方式，让编程访问 Redis 更加简单。

还有我们下节课要讲的 Spring Cache，实际上也是一种再封装、再抽象。它定义了统一、抽象的 Cache 访问接口，这些接口不依赖具体的 Cache 实现（Redis、Guava Cache、Caffeine 等）。在项目中，我们基于 Spring 提供的抽象统一的接口来访问 Cache。这样，我们就能在不修改代码的情况下，实现不同 Cache 之间的切换。

除此之外，还记得我们之前在模板模式中，讲过的 JdbcTemplate 吗？实际上，它也是对 JDBC 的进一步封装和抽象，为的是进一步简化数据库编程。不仅如此，Spring 对 JDBC 异常也做了进一步的封装。封装的数据库异常继承自 DataAccessException 运行时异常。这类异常在开发中无需强制捕获，从而减少了不少不必要的异常捕获和处理。除此之外，Spring 封装的数据库异常，还屏蔽了不同数据库异常的细节（比如，不同的数据库对同一报错定义了不同的错误码），让异常的处理更加简单。

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

借助 Spring 框架，我们总结了框架的作用：解耦业务和非业务开发、让程序员聚焦在业务开发上；隐藏复杂实现细节、降低开发难度、减少代码 bug；实现代码复用、节省开发时间；规范化标准化项目开发、降低学习和维护成本等。实际上，如果要用一句话来总结一下的话，那就是简化开发！

除此之外，我们还重点讲解了 Spring 背后蕴含的一些经典设计思想，主要有：约定优于配置，低侵入、松耦合，模块化、轻量级，再封装、再抽象。这些设计思想都比较通用，我们可以借鉴到其他框架的开发中。

课堂讨论

1. “约定优于配置”在很多开发场景中都有体现，比如 Maven、Gradle 构建工具，它们约定了一套默认的项目目录结构，除此之外，你还能想到体现这条设计思想的其他哪些开发场景吗？
2. 参照 Spring 的设计思想，分析一个你熟悉框架、类库、功能组件背后的设计思想。

欢迎留言和我分享你的想法，如果有收获，也欢迎你把这篇文章分享给你的朋友。