Spring 框架参考文档-5.0.0-中文完整版

Authors
Rod Johnson , Juergen Hoeller , Keith Donald , Colin Sampaleanu , Rob Harrop , Thomas Risberg , Alef Arendsen , Darren Davison , Dmitriy Kopylenko , Mark Pollack , Thierry Templier , Erwin Vervaet , Portia Tung , Ben Hale , Adrian Colyer , John Lewis , Costin Leau , Mark Fisher , Sam Brannen , Ramnivas Laddad , Arjen Poutsma , Chris Beams , Tareq Abedrabbo , Andy Clement , Dave Syer , Oliver Gierke , Rossen Stoyanchev , Phillip Webb , Rob Winch , Brian Clozel , Stephane Nicoll , Sebastien Deleuze

版本号：5.0.0.RELEASE

文档官网：
https://docs.spring.io/spring/docs/5.0.0.RELEASE/spring-framework-reference/index.html

现在官网的 5.0.0 已经在 2017 年 9 月 28 日出了 release 版，为此翻译了 5.0.0 版本（从 4.3.10 升级到此版本，如果还在使用 4.3.10 版本，请看本人的前一个版本），翻译前后历时 15 天，十多次的修改和校对。

翻译特色：
1.翻译可能不会 100%表达原意，但是翻译过程是逐段、逐句、逐字翻译的，尽量在接近原文意；
2.完整的引用标注，上下文引用指示到节，网站引用使用脚注，脚注总量达 400 多，总页数 923 页；
3.代码使用 Markdown 编辑；
4.对于有异议的内容也已标注，【】里的内容是本人自行添加的；
5.5.0.0 版本的内容也已标注。

翻译声明：
1.此 Spring 框架参考文档由本人独立全部翻译和校对，本人对此翻译版本享有使用权和解释权；
2.如果有任何问题或者建议，欢迎来邮件交流：sekift@163.com。

陆印章
2017 年 10 月 14 日

目录

第一部分. Spring 框架概述

Spring 框架是一个轻量级的解决方案，可以一站式构建企业级应用。然而，Spring 是模块化的，允许你只使用那些你需要的部分，而不必把其余的带进来。你可以在任何框架之上去使用 IoC 容器，但你也可以只使用 Hibernate 集成代码（第 20.3 节）或 JDBC 抽象层（第 19.1 节）。Spring 框架支持声明式事务管理、通过 RMI 或 Web 服务远程访问您的逻辑，以及用于保存数据的各种选项。它提供了功能完备的 Web 框架，如 Spring WebMVC（第 22 章）和 Spring WebFlux（第 26 章）；开发者能够将 AOP 透明地集成到软件中。

本文档是 Spring 框架功能的参考指南。对 Framework 本身的问题应该到 StackOverflow 请问（见 https://spring.io/questions[1]）。

1. Spring 框架概述[5.0.0 版本]

Spring 使创建 Java 企业应用程序变得更加容易。它提供了在企业环境中接受 Java 语言所需的一切,并支持 Groovy 和 Kotlin 作为 JVM 上的替代语言,并根据应用程序的需要灵活地创建多种体系结构。在 Spring 框架 5.0 中,Spring 需要 jdk 8+(Java SE 8+),并且已经为 jdk 9 提供了现成的支持。

Spring 支持各种应用场景。在大型企业中,应用程序通常存在很长时间,并且必须在 JDK 和应用服务器上运行,其升级周期超出了开发人员的控制范围。其他人可能作为单一的 jar 运行、嵌入到服务器中,也可能在云环境中。但其他可能是不需要服务器的独立应用程序(如批处理或集成的工作任务)。

Spring 是开源的。它拥有庞大而且活跃的社区，提供不同范围的、真实用户的持续反馈。这也帮助 Spring 不断地改进、不断地发展。

1.1 “Spring”意味着什么

术语“Spring”在不同的上下文有着不同的意义。它可以指 Spring 框架项目本身，这是创建它的初衷。随着时间的推移，其它很多项目也已经从 Spring 框架中建立起来。通常，当说到 Spring 时，特指整个 Spring 的家族。本参考侧重于 Spring 的基础：也就是 Spring 框架本身。

整个 Spring 框架被分成多个模块。应用程序可以选择需要的部分。core 容器是最中心的模块，包括配置模块和依赖注入机制。Spring 框架还为不同的应用程序体系结构提供了基础支持,包括消息传递、事务性数据和持久性以及 Web。还包括基于 Servlet 的 Spring MVC Web 框架,还有在并行世界使用的 Spring WebFlux 响应式的 Web 框架。

关于模块的说明:Spring 的框架 jar 允许部署到 JDK 9 的模块路径("Jigsaw")。为了在启用 Jigsaw 的应用程序中使用,Spring 5。x jar 带有"自动-模块-名称"清单条目,它定义稳定的语言级模块名称(例如"spring.core","spring.context"等)独立于 jar 工件名称(jar 遵循相同的命名模式使用"-"号代替"."号,例如"spring-core"和"spring-context")。当然,Spring 的框架 jar 在 JDK 8 和 9 的类路径上保持正常工作。

1.2 Spring 和 Spring 框架的历史

Spring 的初版发布在 2003 年，是克服早期 J2EE 规范的复杂性的产品。虽然有些人认为 Java EE 和 Spring 是竞争的，但是 Spring 实际上对 Java EE 的补充。Spring 编程模型不受 Java EE 的平台制约;相反,它与精心挑选的个别规范的 EE 项目结合:

[1] https://spring.io/questions

a)Servlet API (JSR 340[2])
b)WebSocket API (JSR 356[3])
c)Concurrency Utilities (JSR 236[4])
d)JSON Binding API (JSR 367[5])
e)Bean Validation (JSR 303[6])
f)JPA (JSR 338[7])
g)JMS (JSR 914[8])
h)as well as JTA/JCA setups for transaction coordination, if necessary.
【简单的不翻译。】

Spring 框架还支持依赖注入(JSR 330[9])和通用注解(JSR 250[10])规范，应用程序开发人员可以选择使用 Spring 框架提供的特定于 Spring 的机制。

在 Spring 框架 5.0 版本中，Spring 最低要求使用 Java EE 7 的版本（例如 Servlet 3.1+,JPA 2.1+），同时在运行时能与使用 Java EE 8 的最新 API 集成（例如 Servlet 4.0、JSON 绑定 API）。这使得 Spring 能完全兼容最新的容器，例如 Tomcat 8/9、WebSphere 9 或者 JBoss EAP 7 等等。

随着时间的不断推移，Java EE 在应用程序开发中越发重要，也不断发展、改善。在 Java EE 和 Spring 的早期，应用程序被创建为部署到服务器的应用。如今，在有 Spring Boot 的帮助后，应用可以创建在 devops 或云端，而 Servlet 容器的嵌入和一些琐碎的东西也发生了变化。在 Spring 框架 5 中，WebFlux 应用程序甚至可以不直接使用 Servlet 的 API，并且可以在非 Servlet 容器的服务器（如 Netty）上运行。

Spring 还在继续创新和发展。如今，除了 Spring 框架以外，还加入了其他项目，例如 Spring Boot、Spring Security、Spring Data、Spring Cloud、Spring Batch,等等的。请记住，每一个 Spring 项目都有自己的源代码库、问题跟踪以及发布版本。请上 spring.io/projects[11]查看所有 Spring 家族的项目名单。

1.3 设计理念

当你学习一个框架时，不仅需要知道他是如何运作的，更需要知道他遵循什么样的原则。以下是 Spring 框架遵循的原则：

a)Spring 在各个层面上都提供可选项。Spring 允许你延迟设计策略。例如，你可以在不更改代码的情况下通过配置来切换持久性的供给。对于其他基础架构的问题以及与第三方 API 的集成也是如此。

b)包含多个视角。Spring 的灵活性非常高，而不是规定了某部分只能做某一件事。他以不同的视角支持广泛的应用需求。

c)保持向后兼容。Spring 的发展经过了静心的管理，在不同版本之间保持与前版本的兼容。Spring 支持一系列精心挑选的 JDK 版本和第三方库，以方便维护依赖于 Spring 的应用程序和库。

d)关心 API 的设计。Spring 团队投入了大量的思想和时间来制作直观的 API，并在许多版本和许多年中都保持不变。

[2] https://jcp.org/en/jsr/detail?id=340
[3] https://www.jcp.org/en/jsr/detail?id=356
[4] https://www.jcp.org/en/jsr/detail?id=236
[5] https://jcp.org/en/jsr/detail?id=367
[6] https://jcp.org/en/jsr/detail?id=303
[7] https://jcp.org/en/jsr/detail?id=338
[8] https://jcp.org/en/jsr/detail?id=914
[9] https://www.jcp.org/en/jsr/detail?id=330
[10] https://jcp.org/en/jsr/detail?id=250
[11] https://spring.io/projects

e)高标准的代码质量。Spring 框架提供了强有力的、精确的、即时的 Javadoc。Spring 这种要求干净、简洁的代码结构、包内没有循环依赖的项目，在 Java 界是少有的。

1.4 反馈和贡献

对于 how-to 问题或诊断、调试问题，我们建议使用 Stackoverflow，Spring 在此网站上有一个专用的页面[12]，列出了建议使用的标记。如果你相当确定 Spring 框架存在问题，或者想建议添加功能等等，可以使用 JIRA 的问题追踪[13]。

如果你在考虑解决方案或者是建议的程序代码，你可以在 Github 上提交请求[14]。但是，请记住，除了最琐碎的问题之外，我们希望在此之前，你可以在问题跟踪页面先提交一波，在那里可以进行讨论并留下记录以作参考。

在你提交共享时，希望你先查看我们在 Github 上的 Wiki[15]。虽然这个 Wiki 是只显示当前的版本，并随 Spring 框架一起做分发，但 Wiki 包含的信息并不特定于任何一个版本。例如，它具有早期版本的迁移注释、有关跨版本的新增内容等等一堆详细信息，还有参与者指南、Spring 框架代码样式和其他信息。

1.5 入门

如果你是刚开始使用 Spring 的，你可能希望使用基于 Spring Boot[16]来编写应用程序，由此开始使用 Spring 框架。Spring Boot 提供了一种快速的（固定设置的）方式来创建即时能用的 Spring 应用程序。它基于 Spring 框架、利用已有的约束配置，目的是让你的程序尽快跑起来。

你可以使用 start.spring.io[17]里的步骤来生成基本项目，或者参考“入门”[18]指南，例如开始创建 RESTful 风格的 Web 服务[19]。除了易于理解，这些指南都是非常重点的任务，其中大多数是基于 Spring Boot 的。当然，里面还囊括了 Spring 很多的其他项目，你可能再需要某些功能时用得上。

1. 开启 Spring 之旅

本参考指南提供了关于 Spring 框架的详细信息。提供它的所有功能全面的文档，以及 Spring 所涵盖的一些关于底层方面的背景资料（如“Dependency Injection（依赖注入）”）。

如果你是刚刚开始接触 Spring，你可能要开始使用 Spring 框架创建基于 Spring Boot[20]的应用。Spring Boot 提供了一种快速、自动配置 Spring 各种组件的方法来创建用于生产环境的 Spring 的应用程序。它是基于 Spring 框架的，约定多于配置，目的是快速搭建可以运行应用。

您可以使用 start.spring.io[21]创建一个基本项目或遵循类似于 “动手创建 RESTful Web 服务”[22]的“入门”[23]指南。这些指南都非常专注于具体的任务，其中大部分是基于 Spring Boot，这部分内容非常容易理解。而且还涵盖了你可能想解决一个特定问题时要考虑到的其他 Spring 项目。

[12] https://spring.io/questions
[13] https://jira.spring.io/browse/spr
[14] https://github.com/spring-projects/spring-framework
[15] https://github.com/spring-projects/spring-framework/wiki
[16] https://projects.spring.io/spring-boot/
[17] https://start.spring.io/
[18] https://spring.io/guides
[19] https://spring.io/guides/gs/rest-service/
[20] http://projects.spring.io/spring-boot/
[21] http://start.spring.io/
[22] https://spring.io/guides/gs/rest-service/
[23] https://spring.io/guides

2. Spring 框架简介

Spring 框架是一个提供完善的基础设施用来支持来开发 Java 应用程序的 Java 平台。Spring 负责基础设施功能，而您可以专注于您的应用。

使用 Spring，你可以用“简单的 Java 对象”（POJO）构建应用程序，并且将企业服务非侵入性的应用到 POJO。此功能适用于 Java SE 编程模型和完全或者部分的 Java EE。

举例，作为一个应用程序的开发者，你可以从 Spring 平台获得以下好处：

a)使 Java 方法可以执行数据库事务而不用去处理事务 API。

b)使本地 Java 方法可以执行远程过程而不用去处理远程 API。

c)使本地 Java 方法可以拥有管理操作而不用去处理 JMX API。

d)使本地 Java 方法可以执行消息处理而不用去处理 JMS API。

2.1 依赖注入和控制反转

Java 应用程序--运行在各个松散的领域,从受限的嵌入式应用程序，到 N 层架构的服务端企业级应用程序--通常由来自应用适当的对象进行组合合作。因此，对象在应用程序中是彼此依赖。

尽管 Java 平台提供了丰富的应用程序开发功能,但它缺乏来组织基本构建块成为一个完整的方法。这个任务留给了架构师和开发人员。虽然您可以使用设计模式，例如 Factory,Abstract Factory，Builder，Decorator，和 Service Locator 来组合各种类和对象实例构成应用程序，这些模式是：给出最佳实践的名字，描述什么模式，哪里需要应用它，它要解决什么问题等等。模式是形式化的最佳实践，但你必须在应用程序中自己来实现。

Spring 框架的 Inversion of Control(IoC)组件旨在通过提供正规化的方法来组合不同的组件成为一个完整的可用的应用。Spring 框架将规范化的设计模式作为一等的对象，您可以集成到自己的应用程序。许多组织和机构使用 Spring 框架以这种方式来开发健壮的、可维护的应用程序。

IoC 背景

“问题是，[他们]哪些方面的控制被反转？”这个问题由 Martin Fowler 在他的 Inversion of Control(IoC)网站于 2004 年提出[24]。Fowler 建议重新命名这个说法,使得他更加好理解,并且提出了 Dependency Injection（依赖注入）这个新的说法。

2.2 模块化

Spring 框架的功能被组织成了 20 来个模块。这些模块分成 Core Container ,Data Access/Integration,Web,AOP(Aspect Oriented Programming),Instrumentation,Messaging,和 Test，如下图：

[24] http://martinfowler.com/articles/injection.html

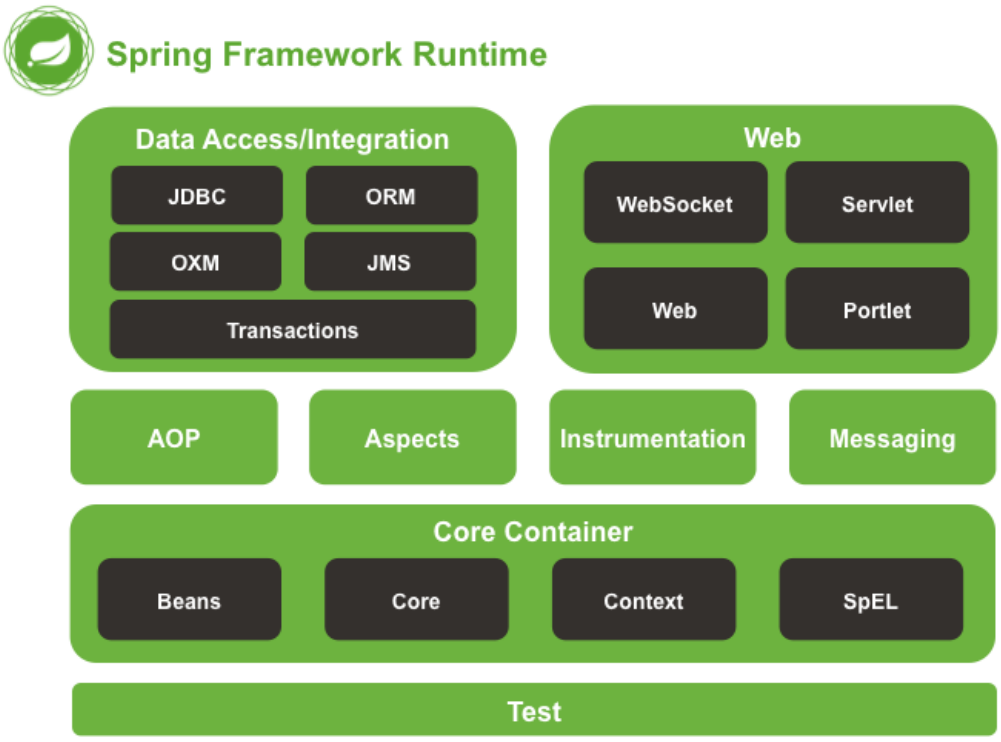

图 2.1 Spring 框架概览

下面章节会列出可用的模块、名称与功能及相关的主题。组件名称与组件的 ID 相关(第 2.3.1 节)。

2.2.1 核心容器

核心容器（第 7.1 节）由 spring-core、spring-beans、spring-context、spring-context-support 和 spring-expression 模块组成。

spring-core 和 spring-beans 提供框架的基础部分（第 7.1 节），包括 IoC 和 Dependency Injection 功能。BeanFactory 是一个复杂的工厂模式的实现。不需要可编程的单例，允许您将配置和特定的依赖从你的实际程序逻辑中解耦。

Context(spring-context)（第 7.15 节）模块建立且提供于在 Core 和 Beans（第 7.1 节）模块的基础上，它是一种在框架类型下实现对象存储操作的手段，就像 JNDI 注册。Context 继承了 Beans 模块的特性，并且增加了对国际化的支持（例如用在资源包中）、事件广播、资源加载和创建上下文（例如 Servlet 容器）。Context 模块也支持例如 EJB，JMX 和基础远程这样的 Java EE 特性。ApplicationContext 是 Context 模块的重点。spring-context-support 提供对常见第三方库的支持，集成到 Spring 应用上下文，如缓存(EhCache,Guava,JCache)，通信(JavaMail)，调度(CommonJ,Quartz)和模板引擎(FreeMarker ,JasperReports,Velocity)。

spring-expression 模块提供了一个强大的 Expression Language（表达式语言，第 10 章）用来在运行时查询和操作对象图。这是作为 JSP 2.1 规范所指定的统一表达式语言（unified EL）的一种延续。这种语言支持对属性值、属性参数、方法调用、数组内容存储、收集器和索引、逻辑和算数操作及命名变量，并且通过名称从 Spring 的控制反转容器中取回对象。表达式语言模块也支持 List 的映射和选择，正如像常见的列表汇总一样。

2.2.2 AOP 和设备模块

spring-aop 模块提供 AOP Alliance-compliant（联盟兼容，第 11.1 节）的面向切面编程实现，允许你自定义注解，比如，方法拦截器和切入点完全分离代码。使用源码级别元数据的功能，你也可以在你的代码中加入行为信息，在某种程度上类似于.NET 属性。

单独的 spring-aspects 模块提供了 AspectJ 的集成和使用。

spring-instrument 模块提供了类 instrumentation 的支持和在某些应用程序服务器使用类加载器实现。spring-instrument-tomcat 用于 Tomcat Instrumentation 代理。

2.2.3 消息组件

Spring 框架 4 包含了 spring-messaging 模块，从 Spring 集成项目中抽象出来，比如 Message,MessageChannel,MessageHandler 及其他用来提供基于消息的基础服务。该模块还包括一组消息映射方法的注解，类似于基于编程模型 Spring MVC 的注解。

2.2.4 数据访问/集成

数据访问和集成层由 JDBC、ORM、OXM、JMS 和事务模块组成。

spring-jdbc 模块提供了不需要编写冗长的 JDBC 代码（第 19.1 节）和解析数据库厂商特有的错误代码的 JDBC-抽象层。

spring-tx 模块的支持可编程和声明式事务管理（第 17 章），用于实现了特殊的接口的和你所有的 POJO 类（Plain Old Java Objects）。

spring-orm 模块提供了流行的 object-relational mapping（对象/关系映射，第 20.1 节）API 集成层，其包含 JPA（第 20.5 节），JDO（第 20.4 节），Hibernate（第 20.3 节）。使用 ORM 包，你可以使用所有的 O/R 映射框架结合所有 Spring 提供的特性，例如前面提到的简单声明式事务管理功能。

spring-oxm 模块提供抽象层用于支持 Object/XML mapping（对象/XML 映射，第 21 章）的实现，如 JAXB、Castor、XMLBeans、JiBX 和 XStream 等。

spring-jms 模块（Java Messaging Service，第 30 章）包含生产和消费信息的功能。从 Spring 框架 4.1 开始提供集成 spring-messaging 模块。

2.2.5 Web

Web 层包括 spring-web、spring-webmvc、spring-websocket 和 spring-webmvc-portlet 模块组成。

spring-web 模块提供了基本的面向 Web 开发的集成功能，例如多方文件上传、使用 Servlet listeners 和 Web 开发应用程序上下文初始化 IoC 容器。它也包含 HTTP 客户端以及 Spring 远程访问的支持的 Web 相关的部分。

spring-webmvc 模块（也被称为 Web Servlet 模块）包含 Spring 的 model-view-controller（模型-视图-控制器（MVC，第 22.1 节）和 REST Web Services 实现的 Web 应用程序。Spring 的 MVC 框架提供了 domain model(领域模型)代码和 web form(网页)之间的完全分离，并且集成了 Spring 框架所有的其他功能。

spring-webmvc-portlet 模块（也被称为 Web-Portlet 模块）提供了 MVC 模式的实现是用 Portlet 的环境和 spring-webmvc 模块功能的镜像。

2.2.6 测试

spring-test 模块支持通过组合 JUnit 或 TestNG 来进行单元测试（第 14 章）和集成测试（第 15 章）。它提供了连续的加载（第 15.5.4 节）ApplicationContext 并且缓存（第 15.5.4.10 节）这些上下文。它还提供了 mock object（模仿对象，第 14.1 节），您可以使用隔离测试你的代码。

2.3 应用场景

前面的构建模块描述在很多的情况下使 Spring 是很合理的选择，从资源受限的嵌入式程序到成熟的企业应用程序都可以使用 Spring 事务管理功能和 Web 框架集成。

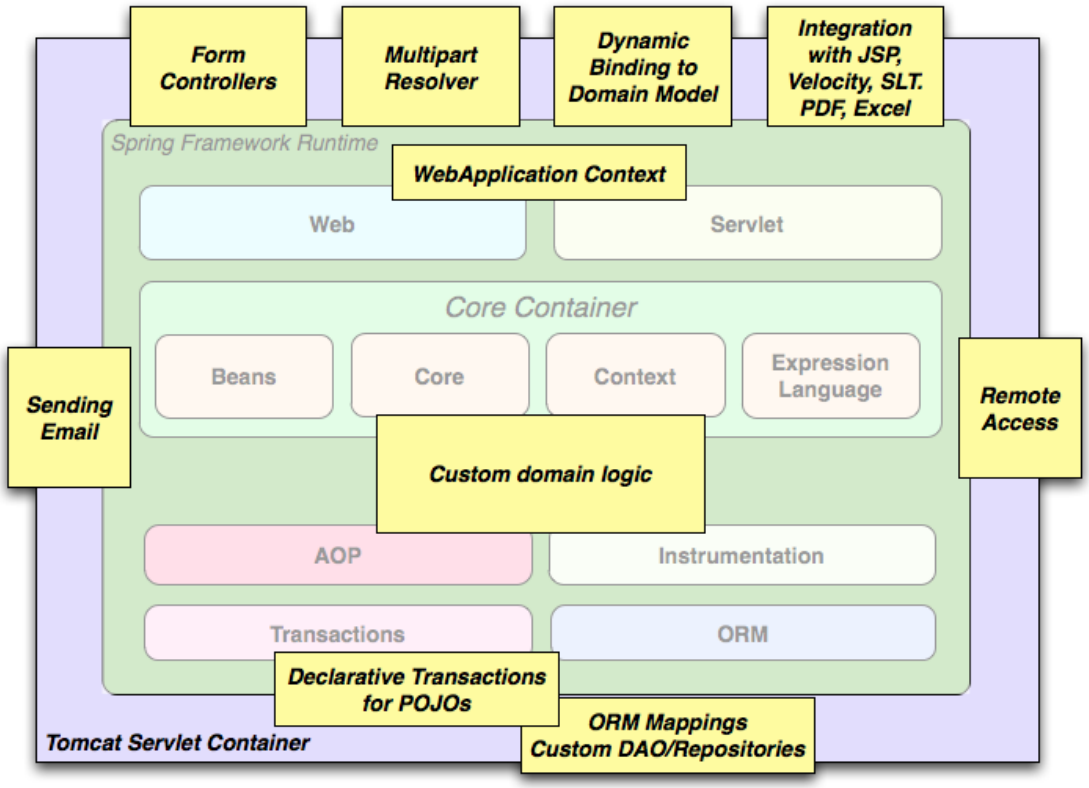

图 2.2 典型的成熟型的 Spring Web 应用

Spring 声明式事务管理特性（第 17.5 节）使 Web 应用程序拥有完全的事务，就像你使用 EJB 容器管理的事务。所有的自定义业务逻辑可以用简单的 POJO 实现，用 Spring 的 IoC 容器进行管理。额外的服务包括发送电子邮件和验证，是独立的网络层的支持，它可以让你选择在何处执行验证规则。Spring 的 ORM 支持集成 JP A，Hibernate，JDO；例如，当使用 Hibernate，您可以继续使用现有的映射文件和标准的 Hibernate 的 SessionFactory 配置。表单控制器将 Web 层和领域模型无缝集成，消除 ActionForms 或其他类似于变换 HTTP 参数成为您的域模型值的需要。

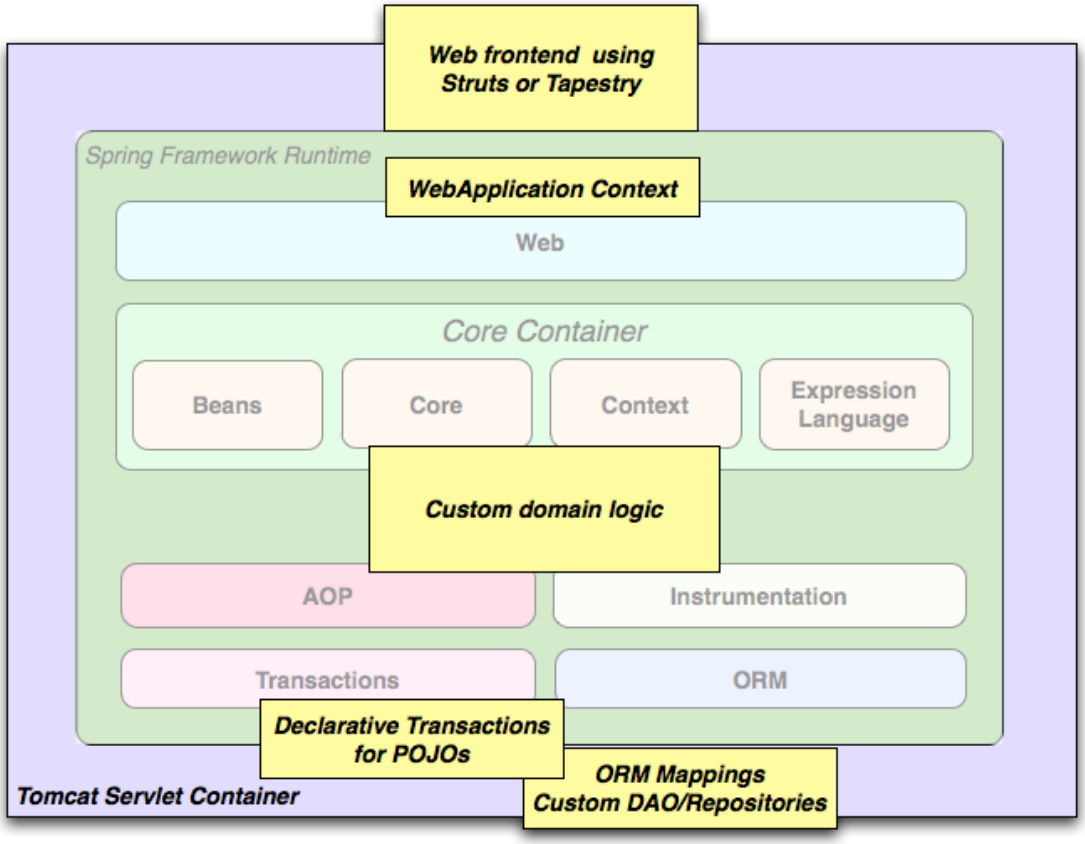

图 2.3 Spring 使用第三方 Web 框架工具的中间层

有时，不允许你完全切换到不同的框架。Spring 框架不强制使用它,它不是全有或全无的解决方案。现有的前端 Struts,Tapestry ,JSF 或其他 UI 框架，可以集成基于 Spring 中间件，它允许你使用 Spring 事务的功能。你只需要将业务逻辑连接使用 ApplicationContext 和使用 WebApplicationContext 集成到你的 Web 层。

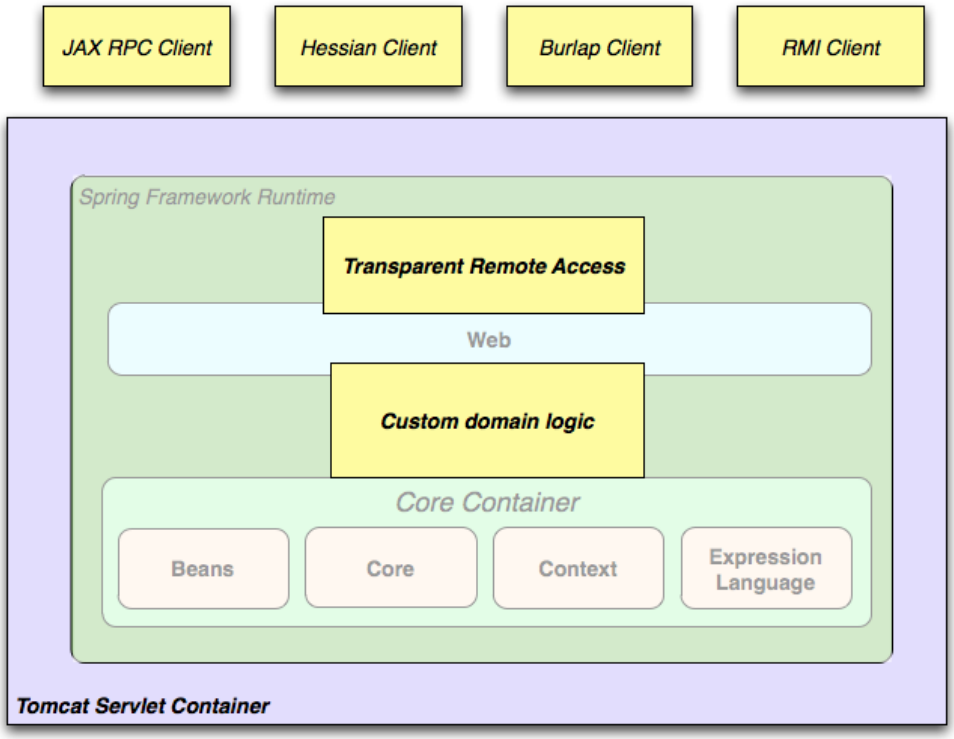

图 2.4 远程使用场景

当你需要通过 Web 服务访问现有代码时，可以使用 Spring 的 Hessian-,Burlap-,Rmi-或 JaxRpcProxyFactory 类。启用远程访问现有的应用程序并不难。

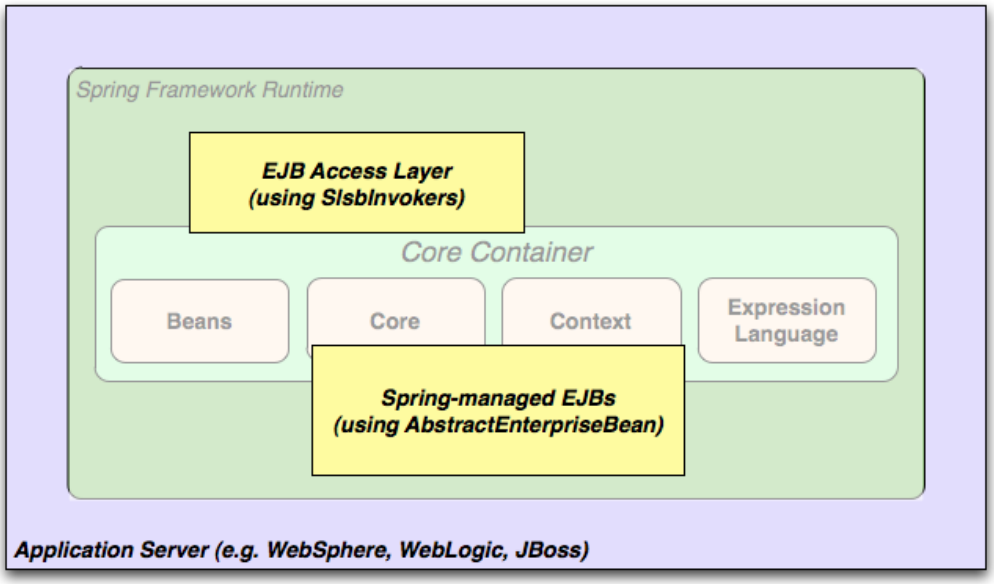

图 2.5 EJBs—包裹已存在的 POJO

Spring 框架也提供了 Enterprise JavaBeans 访问和抽象层(第 29 章)使你能重用现有的 POJOs，并且在需要声明安全的时候打包他们成为无状态的 bean 用于可伸缩，安全的 Web 应用里。

2.3.1 依赖管理和命名规范

依赖管理和依赖注入是不同的概念。为了让 Spring 的这些不错的功能运用到运用程序中（比如依赖注入），你需要收集所有的需要的库（JAR 文件），并且在编译、运行的时候将它们放到你的类路径

中。这些依赖不是虚拟组件的注入，而是物理的资源在文件系统中（通常）。依赖管理过程包括定位这些资源，存储它们并添加它们到类路径。依赖可以是直接（如我的应用程序运行时依赖于 Spring），或者是间接（如我的应用程序依赖 commons-dbcp，而 commons-dbcp 又依赖于 commons-pool）。间接的依赖也被称为“transitive（传递）”，它是最难识别和管理的依赖。

如果你将使用 Spring，你需要复制哪些包含你需要的 Spring 功能的 jar 包。为了使这个过程更加简单，Spring 被打包为一组模块，这些模块尽可能多的分开依赖关系。例如，如果不想写 Web 应用程序，你就不需要引入 Spring-web 模块。为了在本指南中标记 Spring 库模块我们使用了速记命名约定 spring-* 或者 spring-*.jar，其中 * 代表模块的短名（比如 spring-core,spring-webmvc,spring-jms 等）。实际的 jar 文件的名字，通常是用模块名字和版本号级联（如 spring-core-4.3.0.RELEASE.jar）

每个 Spring 框架发行版本将会放到下面的位置：

a)Maven Central（Maven 中央库），这是 Maven 查询的默认库，而不需要任何特殊的配置就能使用。许多常用的 Spring 的依赖库也存在与 Maven Central，并且 Spring 社区的很大一部分都使用 Maven 进行依赖管理，所以这是最方便他们的。jar 命名格式是 spring-*-<version>.jar，Maven groupId 是 org.springframework。

b)公共 Maven 仓库还拥有 Spring 专有的库。除了最终的 GA 版本，这个库还保存开发的快照和里程碑。JAR 文件的名字是和 Maven Central 相同的形式，所以这是让 Spring 的开发版本使用其它部署在 Maven Central 库的有用的地方。该库还包含一个用于发布的 zip 文件包含所有 Spring jar，方便下载。

所以首先，你要决定用什么方式管理你的依赖，通常建议你使用自动系统像 Maven,Gradle 或 Ivy，当然你也可以下载 jar。

下面是 Spring 中的组件列表。更多描述，详见第 2.2 节“框架模块”。

表 2.1 Spring 框架构件

GroupId	ArtifactId	Description
org.springframework	spring-aop	基于代理的 AOP 支持
org.springframework	spring-aspects	基于 AspectJ 的切面
org.springframework	spring-beans	Beans 的支持，包括 Groovy
org.springframework	spring-context	应用程序上下文运行，包括调度和远程抽象
org.springframework	spring-context-support	将公共第三方库集成到 Spring 应用程序上下文中的支持类
org.springframework	spring-core	许多其他 Spring 模块使用的核心实用程序
org.springframework	spring-expression	Spring 表达式语言(SpEL)
org.springframework	spring-instrument	用于 JVM 引导的检测代理
org.springframework	spring-instrument-tomcat	用于 Tomcat 的检测代理
org.springframework	spring-jdbc	JDBC 支持包，包括数据源设置和 JDBC 访问支持
org.springframework	spring-jms	JMS 支持包，包括用于发送和接收 JMS 消息的帮助类
org.springframework	spring-messaging	对消息传递体系结构和协议的支持
org.springframework	spring-orm	对象/关系映射，包括 JPA 和

GroupId	ArtifactId	Description
		Hibernate 支持
org.springframework	spring-oxm	对象/XML 映射
org.springframework	spring-test	支持单元测试和集成测试 Spring 组件
org.springframework	spring-tx	事务架构,包括 DAO 支持和 JCA 集成
org.springframework	spring-web	WEB 支持包,包括客户端和 WEB 远程处理
org.springframework	spring-webmvc	WEB 应用程序的 REST WEB 服务和模型-视图-控制器实现
org.springframework	spring-webmvc-portlet	要在 Portlet 环境中使用的 MVC 实现
org.springframework	spring-websocket	WebSocket 和 SockJS 实现，包括 STOMP 支持

2.3.1.1 Spring 的依赖和依赖 Spring

虽然 Spring 提供了集成在很大范围的企业和其他外部工具的支持，它故意保持其强制性依赖关系降到最低：在简单的用例里，你无需查找并下载（甚至自动）一大批 jar 库来使用 Spring。基本的依赖注入只有一个外部强制性的依赖，这是用来做日志的（见下面更详细地描述日志选项）。

接下来我们将一步步展示如果配置依赖 Spring 的程序，首先用 Maven 然后用 Gradle 和最后用 Ivy。在所有的情况下，如果有不清楚的地方，查看的依赖性管理系统的文档，或看一些示例代码。Spring 本身是使用 Gradle 来管理依赖的，我们的很多示例也是使用 Gradle 或 Maven。

2.3.1.2 Maven 依赖管理

如果您使用的是 Maven[25]的依赖管理你甚至不需要明确提供日志依赖。例如，要创建应用程序的上下文和使用依赖注入来配置应用程序，你的 Maven 依赖将看起来像这样：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.10.RELEASE</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

这样就可以了。注意 scope 可声明为 runtime（运行时）如果你不需要编译 Spring API，这通常是基本的依赖注入案例。

以上与 Maven Central 存储库工程实例。使用 Spring Maven 存储库（如里程碑或开发者快照），你需要在你的 Maven 配置指定的存储位置。如下：

```
<repositories>
```

[25] https://maven.apache.org/

```
    <repository>
        <id>io.spring.repo.maven.release</id>
        <url>http://repo.spring.io/release/</url>
        <snapshots><enabled>false</enabled></snapshots>
    </repository>
</repositories>
```

里程碑版本如下：

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.milestone</id>
        <url>http://repo.spring.io/milestone/</url>
        <snapshots><enabled>false</enabled></snapshots>
    </repository>
</repositories>
```

快照版本如下：

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.snapshot</id>
        <url>http://repo.spring.io/snapshot/</url>
        <snapshots><enabled>true</enabled></snapshots>
    </repository>
</repositories>
```

2.3.1.3 Maven 的依赖清单

在使用 Maven 时，有可能下了不同版本的 Spring JARs。例如，你可能发现第三方的库，或另一个 Spring 的项目，拉取了依赖前发布的包。如果你自己忘记了显式声明直接依赖，各种意想不到的问题就会出现。

为了克服这些问题，Maven 支持"bill of materials"(BOM)的依赖的概念。你可以在你的 dependency Management 部分引入 spring-framework-bom 来确保所有 spring 依赖（包括直接和传递的）是同一版本。

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>4.3.10.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

使用 BOM 后，当依赖 Spring 框架组件后，无需指定 <version>属性

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>
<dependencies>
```

2.3.1.4 Gradle 的依赖管理

用 Gradle[26]来使用 Spring,在 repositories 中填入适当的 URL:

```
repositories {
    mavenCentral()
    // and optionally...
    maven { url "http://repo.spring.io/release" }
}
```

可以适当修改 URL 从/release 到/milestone 或/snapshot。库一旦配置，就能声明 Gradle 依赖:

```
dependencies {
    compile("org.springframework:spring-context:4.3.4.RELEASE")
    testCompile("org.springframework:spring-test:4.3.4.RELEASE")
}
```

[26] http://www.gradle.org/

2.3.1.5 Ivy 的依赖管理

如果你更喜欢用 Ivy[27]管理依赖，其配置选项也是类似的。

如果配置 Ivy，可以将指向 Spring 的库添加下面的 resolver（解析器）到你的 ivysettings.xml 文件中：

```
<resolvers>
    <ibiblio name="io.spring.repo.maven.release"
            m2compatible="true"
            root="http://repo.spring.io/release/"/>
</resolvers>
```

可以适当修改 root URL，可以是/release、/milestone 或/snapshot。

一旦配置好了，就可以添加依赖了，例如（在 ivy.xml 文件中）：

```
<dependency org="org.springframework"
    name="spring-core" rev="4.3.10.RELEASE" conf="compile->runtime"/>
```

2.3.1.6 发布的 zip 文件

虽然使用构建系统，支持依赖管理是推荐的方式获得了 Spring 框架，，它仍然是可下载分发的 zip 文件。

分发的 zip 文件是发布到 Spring Maven Repository（这是为了我们的便利，在下载这些文件的时候你不需要 Maven 或者其他的构建系统）。

下载一个 Zip，在 Web 浏览器打开 http://repo.spring.io/release/org/springframework/spring[28]，选择适当的文件夹的版本。下载完毕文件结尾是-dist.zip，例如，spring-framework-{springversion}-RELEASE-dist.zip。分发也支持发布里程碑[29]和快照[30]。

2.3.2 日志

对于 Spring 日志是非常重要的依赖，因为：

a）它是唯一的外部强制性的依赖；

b）每个人都喜欢从他们使用的工具看到一些输出；

c）Spring 结合很多其他工具都选择了日志依赖。

应用开发者的目标就是在整个应用程序中使用统一的日志配置，包括所有的外部组件。这时显得更加困难，因为现在已经有太多选择的日志框架供选择。

在 Spring 强制性的日志依赖是 Jakarta Commons Logging API（JCL）。编译 JCL 时，开发者也使得 JCL Log 对象对 Spring 框架的扩展类可见。所有版本的 Spring 使用同样的日志库，这对于用户来说是很重要的：迁移就会变得容易向后兼容性，即使扩展了 Spring 的应用程序。我们这样做是为了使 Spring 的模块之一明确依赖 commons-logging(JCL 的典型实现)，然后使得其他的所有模块在编译的时候都依赖它。使用 Maven 为例，如果你想知道何处依赖了 commons-logging，那么就是来自

[27] https://ant.apache.org/ivy
[28] http://repo.spring.io/release/org/springframework/spring
[29] http://repo.spring.io/milestone/org/springframework/spring
[30] http://repo.spring.io/snapshot/org/springframework/spring

Spring 的并且明确来自中心模块：spring-core。

关于 commons-logging 的好处是你不需要任何东西就能让你的应用程序程序跑起来。它有运行时发现算法，该算法在众所周知的 classpath 路径下寻找其他的日志框架并且使用它认为适合的（或者你可以告诉它你需要的是哪一个）。如果没有其他的日志框架存在，你可以从 JDK（Java.util.logging 或者简称为 JUL）获得日志。在大多数情况下，你可以在控制台查看你的 Spring 应用程序工作和日志，并且这是很重要的。

2.3.2.1 使用 Log4j 1.2 或 2.x

注意：现在 Log4j 1.2 版本已经停止更新。同时，Log4j 2.3 版本是最后一版兼容 Java 6 的，而最新发布的 2.x 版本都必需要 Java 7 及以上的 Java 版本。

许多人使用 Log4j[31]作为日志框架，用于配置和管理的目的。它是有效的和完善的，事实上这也是我们在运行时使用的，当我们构架和测试 Spring 时。Spring 也提供一些配置和初始化 Log4j 的工具，因此在某些模块上它有可选的编译时依赖在 Log4j。

许多人使用 Log4j 1.2 作为日志框架，用于配置和管理的为了使 Log4j 工作在默认的 JCL 依赖下（commons-logging），你所需要做的就是把 Log4j 放到类路径下，为它提供配置文件(log4j.properties 或者 log4j.xml 在类路径的根目录下)。因此对于 Maven 用户这就是你的依赖声明：目的。它是有效的和完善的，事实上这也是我们在运行时使用的，当我们构架和测试 Spring 时。Spring 也提供一些配置和初始化 Log4j 的工具,因此在某些模块上它有可选的编译时依赖需要 Log4j。

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.10.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

下面是 log4j.properties 的实例，用于将日志打印到控制台：

```
log4j.rootCategory=INFO, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %t %c{2}:%L - %m%n
log4j.category.org.springframework.beans.factory=DEBUG
```

要使用 Log4j 2.x 与 JCL,所有您需要做的是把 Log4j jar 包加入到类路径中,并提供配置文件(log4j2.xml,log4j2.properties,或其他支持的配置格式[32])。对于 Maven 用户,所需的最小依赖包是:

[31] https://logging.apache.org/log4j
[32] https://logging.apache.org/log4j/2.x/manual/configuration.html

```
<dependencies>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.6.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-jcl</artifactId>
        <version>2.6.2</version>
    </dependency>
</dependencies>
```

如果您还希望启用 SLF4J 委派到 Log4j 工作例如，对于默认情况下使用 SLF4J 的其他库，也需要以下依赖项:

```
<dependencies>
    dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j-impl</artifactId>
        <version>2.6.2</version>
    </dependency>
</dependencies>
```

下面是用于记录到控制台的 log4j2.xml 示例:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout  pattern="%d{HH:mm:ss.SSS}  [%t]  %-5level  %logger{36}
        - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
    <Logger name="org.springframework.beans.factory" level="DEBUG"/>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

2.3.2.2 不使用通用的日志

不幸的是，commons-logging 的运行时日志框架发现算法，确实方便了最终用户,但却是有问题的。如果我们能够时光倒流，现在从新开始 Spring 项目并且他使用了不同的日志依赖。第一个选择很可能是 Simple Logging Façade for Java(SLF4J)，过去也曾被许多其他工具通过 Spring 使用到他们的应用程序。

有两种方式可以关闭 commons-logging:

a) 通过 spring-core 模块排除依赖（因为它是唯一的显示依赖于 commons-logging 的模块）。

b)依赖特殊的 commons-logging 依赖，用空的 jar（更多的细节可以在 SLF4J FAQ[33]中找到）替换掉库。

排除 commons-logging，添加以下的内容到 dependencyManagement 部分：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.10.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

现在，这个应用程序可能运行不了，因为在类路径上没有实现 JCL API，因此要修复它就必须提供一个新的(日志框架)。在下一节我们将向你展示如何提供另一种实现 JCL，使用 SLF4J 的另一种实现。

2.3.2.3 将 Log4j 或 Logback 用作 SLF4J

The Simple Logging Facade for Java (SLF4J[34])是常用于 Spring 的日志组件，使用其他库实现了常用的 API。它通常用于 Logback[35]，它是 SLF4J API 的本地实现。

SLF4J 提供了绑定很多的常见日志框架，包括 JCL，它也做了反向工作:是其他日志框架和它自己之间的桥梁。因此在 Spring 中使用 SLF4J 时，你需要使用 SLF4J-JCL 桥接替换掉 commons-logging 的依赖。一旦你这么做了，Spring 调用日志就会调用 SLF4JAPI，因此如果在你的应用程序中的其他库使用这个 API，那么你就需要有个地方配置和管理日志。

一个常见的选择就是桥接 Spring 和 SLF4J，提供显示的绑定 SLF4J 到 Log4J 上。你需要支持 4 个的依赖（排除现有的 commons-logging）：桥接，SLF4JAPI，绑定 Log4J 和 Log4J 实现自身。在 Maven 中你可以这样做：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
```

[33] http://slf4j.org/faq.html#excludingJCL
[34] http://www.slf4j.org/
[35] https://logback.qos.ch/

```
        <version>4.3.10.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.821</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

对于 SLF4J 用户来说，更常见的选择是：使用更少的步骤和产生更少的依赖，那就是直接绑定 Logback。这消除了多余的绑定步骤，因为 Logback 直接实现了 SLF4J，因此你只需要依赖两个库而不是 4 个（jcl-over-slf4j 和 logback）。

```
<dependencies>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.1.7</version>
    </dependency>
</dependencies>
```

2.3.2.4 使用 JUL(java.util.logging)

如果在类路径上未检测到 Log4j，则共享日志记录默认委派给 java.util.logging。因此,没有

特殊的依赖关系可以设置：只需在独立应用程序(使用 JDK 级别的自定义或默认 JUL 安装程序)或应用服务器的日志系统(系统范围的 JUL 设置)。

2.3.2.5 WebSphere 的公共日志

Spring 应用程序可以在本身提供 JCL 实现的容器上运行，例如 IBM 的 WebSphere 应用程序服务器。这本身不会导致问题，但会导致需要了解的两种不同情况：

在"父级优先"的类加载器委派模型(默认值为 on)中，应用程序总是会接收到服务器提供的 Commons Logging，将其委派给被记录子系统(实际上是基于 JUL)。应用程序提供 JC 的变体，无论是标准 Commons Logging 还是 JCL-over-SLF4J 的桥接，都将与任何本地包含的日志提供程序一起被忽略。

使用"父级在后"的委派模型(常规 Servlet 容器中的默认设置,但在 WAS 上有一个显式配置选项)，将获取应用程序提供的 Commons Logging 变体，使您能够设置本地包含的 log 提供程序,例如 Log4j 或 Logback,在您的应用程序中。如果没有本地日志提供程序，则在默认情况下，Commons Logging 将委派给 JUL，有效地记录到 WebSphere 的日志子系统,如"父级优先"的方案。

最后,我们建议在"父级在后"模型中部署 Spring 应用程序,因为它自然允许本地提供程序以及服务器的日志子系统。

第二部分. Spring 框架 5.x 中的新特性[5.0.0 版本]

地 址 ：https://github.com/spring-projects/spring-framework/wiki/What%27s-New-in-the-Spring-Framework#whats-new-in-spring-framework-5x

3. Spring 框架 5.0 中的新特性和新功能

3.1 JDK 8+和 Java EE 7+基础

a）现在是基于 Java 8 源代码级别的整个框架代码库。
    a)通过可推断的泛型、lambda 等提供可读性。
b)现在在代码上有对 Java 8 功能的条件性支持。
    b）与 JDK 9 的开发和部署完全兼容。
a)兼容的类路径以及模块路径(具有稳定的自动模块名称)。
    b)JDK 9 上的框架构建和测试套件的兼容(默认情况下是在 JDK 8 上运行)。
c)Spring 有些功能现在需要 Java EE 7 的 API。
    a)Servlet 3.1、Bean 验证 1.1、JPA 2.1，JMS 2.0 等
    b)最新的服务器，如 Tomcat 8.5+、Jetty 9.4+、WildFly 10+
d) 在运行时与 Java EE 8 API 级别的兼容性。
    a)Servlet 4.0、Bean 验证 2.0、JPA 2.2、JSON 绑定 API 1。0
b) Tomcat 测试 9.0、Hibernate 验证 6.0、Apache Johnzon 1.1.

3.2 移除的包、类和方法

a)包 beans.factory.access（BeanFactoryLocator 机制）。
b)包 jdbc.support.nativejdbc（NativeJdbcExtractor 机制）。
c)包 mock.staticmock 从 spring-aspects 模块中移除了。
    a)不再支持 AnnotationDrivenStaticEntityMockingControl。
d)包 web.view.tiles2 和 orm.hibernate3/hibernate4 已被移除。
    a)现在最低的版本要求是：Tiles 3 和 Hibernate 5。
e)不再支持：Portlet, Velocity, JasperReports, XMLBeans, JDO, Guava.
    a)推荐：如果还需要它们，那么请继续使用 4.3.x 框架。
f)在代码库中删除了很多已经否决的类和方法。
    a)对生态系统中常用的方法作出了一些妥协。

3.3 主要的 core 包修改

a)JDK 8+的增强功能：
    a)基于 Java 8 反射增强的高效方法参数访问。
    b)在核心 Spring 接口中有选择地声明 Java 8 的默认方法。
    c)一致使用 JDK 7 字符集和 StandardCharsets 增强功能。
b)JDK 9 兼容性：
    a)尽可能避免 JDK 9 中弃用的 JDK API。
    b)通过构造函数进行一致的实例化(修改后的异常处理)。

c)对核心 JDK 类的反射属性进行有预防性的使用。

c)在包级别上使用 Non-null 的 API 声明：

a)使用@Nullable 显式注解的可为 null 的参数、字段和返回值。

b)主要用于 IntelliJ IDEA 和 Kotlin，同时也可以用于 Eclipse 和 Findbugs。

c)一些 Spring API 不再容忍 null 值（例如在 SpringUtils 中）。

d)资源抽象提供了用于预防 getFile 访问的 isFile 指示器。

a)在资源接口中提供了基于 NIO readableChannel 的访问者。

b)通过 NIO 2 流（不再使用 FileInput/OutputStream）进行文件系统访问。

e)Spring 框架 5.0 提供现成的与 Commons Logging 结合的桥梁：

a)spring-jcl 代替了标准的 Commons Logging; excludable/overridable 保持。

b)在没有任何额外的桥梁时，能自动检测 Log4j 2.x, SLF4J, JUL (java.util.logging)系统。

f)spring-core 将基于 ASM 6.0【就是修改字节码的 ASM】，下一步是打算基于 CGLIB 3.2.5 和 Objenesis 2.6）。

3.4 核心容器

a)支持任何@Nullable 注解作为可选的注入点。

b)GenericApplicationContext/AnnotationConfigApplicationContext 的函数式风格、

a)带有 bean 定义的自定义回调的基于 Supplier bean 注册的 API。

c)在接口方法上有一致的检测事务、缓存和异步注解。

a)在 CGLIB 代理的情况下。

d)将 XML 配置命名空间简化为无版本模式。

a)始终解决最新的 xsd 文件；不支持已弃用的功能。

b)仍然支持版本特定的声明，但针对最新的模式进行了验证。

e)支持候选组件索引（作为类路径扫描的替代方法）.

3.5 Spring Web MVC

a)在 Spring 提供的过滤器实现中，完全支持 Servlet 3.1 签名。

b)在 Spring MVC 控制器方法中支持 Servlet 4.0 PushBuilder 参数。

c)MaxUploadSizeExceededException 用于公共服务上的 Servlet 3.0 多部分上传的解析。

d)通过 MediaTypeFactory 委托对公共媒体类型的统一支持。

a)用于取代 Java 的激活框架。

e)与不可变对象的数据绑定（Kotlin/Lombok/@ConstructorProperties）。

f)支持 JSON Binding API（作为 Jackson 和 GSON 的替代品）。

g)支持 Jackson 2.9。

h)支持 Protobuf 3.

i)支持 Reactor 3.1 Flux 和 Mono 并且支持 RxJava 1.3 和 2.1 作为 Spring MVC 控制方法的返回值,这也是 Spring MVC 控制器中新型的 reactive WebClient(看下节)或 Spring Data Reactive 仓库。

j)新的 ParsingPathMatcher 替代 AntPathMatcher 提供了更有效的解析和扩展语法。

k)@ExceptionHandler 方法允许 RedirectAttributes 参数(因此是 flash 属性)。

l)支持 ResponseStatusException 作为@ResponseStatus 语法上的替代方案。

m)支持不实现 Invocable 的脚本引擎,方法是通过直接使用 ScriptEngine#eval(String,

Bindings)提供的脚本，并通过新的 RenderingContext 参数在 ScriptTemplateView 中 i18n 和嵌套模板上使用。

n)Spring 的 FreeMarker 宏(spring.ftl)现在使用 HTML 输出格式(要求 FreeMarker 2.3.24+)。

3.6 Spring WebFlux

a)全新的 Spring 模块：spring-webflux，一个用来代替 spring-webmvc 的模块，它建立在响应式[36]的基础上 -- 完全异步和非阻塞的，用于事件循环执行模型对比传统的那种使用单个线程请求模型的大型线程池。

b)响应式架构在 spring-core 中的作用就像 Encoder 和 Decoder 用于对象流进行编码和解码一样；DataBuffer 抽象，如使用 Java ByteBuffer 或 Netty ByteBuf；在控制器方法的签名中，包括响应式库中的 ReactiveAdapterRegistry，它提供这种转换的支持。

c)spring-web 中的响应式架构包含 HttpMessageReader 和 HttpMessageWriter，这些都是建立和委托在 Encoder 和 Decoder 的基础上；带有适配器的 HttpHandler（非阻塞的）用在运行时的服务器上，例如 Servlet 3.1+容器、Netty 和 Undertow；WebFilter、WebHandler 和其他与 Servlet API 等价的非阻塞功能都是替代品。

d)@Controller 风格，基于注解，编程模型，类似 Spring MVC，但被 WebFlux 支持，运行在响应式栈中，例如支持使用响应式的类型作为控制器方法参数，永远不会阻塞在 I/O 上，遵循所有使用在 HTTP 套接字上的方式，并且运行额外的非 Servlet 容器，例如 Netty 和 Undertow。

e)新型的函数式编程模型（“WebFlux.fn”）可以作为@Controller 的替代方案，基于注解，编程模型 -- 其终端路由 API 小而透明，运行在相同的响应式栈和 WebFlux 架构上。

f)为 HTTP 调用而建的具有函数式的和响应式 API 的 WebClient，能够与 RestTemplate 相比美，但是它是建立在 WebFlux 架构上的流式 API，具有非阻塞以及处理流式场景的出色功能；在 Spring 5.0 中，AsyncRestTemplate 已经被弃用并且被 WebClient 取代。

3.7 Kotlin 的支持

a)Kotlin 1.1.50+使用到了 null 安全 API

b)支持具有可选参数和默认值的 Kotlin 不可变类

c)使用函数式的 bean 定义 Kotlin DSL

d)使用函数式路由的 Kotlin DSL 用于 WebFlux

e)利用 Kotlin 具体化类型参数，避免显示指定要用于 RestTemplate 或 WebFlux API 等各种 API 的序列化/反序列化的类

f)Kotlin 的 null 安全支持@Autowired/@Inject 和@RequestParam/@RequestHeader/等注解，以确定是否需要 parameter/bean。

g)Kotlin 脚本支持在 ScriptTemplateView 上，用在 Spring MVC 和 Spring WebFlux。

h)Model、ModelMap 和 Environment 中添加了类数组的 setter

i)支持具有可选参数的 Kotlin 自动构造函数

j)Kotlin 反射用于确定接口方法参数

3.8 测试的改进

a)Spring TestContext 框架完全支持 JUnit 5[37]的 Jupiter 编程和扩展模型。

[36] https://github.com/reactive-streams/reactive-streams-jvm
[37] http://junit.org/junit5/

a)SpringExtension:来自 JUnit Jupiter 的多个扩展 API 实现，它为 Spring TestContext 框架的现有功能集成提供了完全的支持。此支持是通过@ExtendWith(SpringExtension.class)启用的.

b)@SpringJUnitConfig：一个组合的注解，它将@ExtendWith（SpringExtension.class）与来自 JUnit Jupiter 的@ContextConfiguration 在 Spring TestContext 框架中集成。

c) @SpringJUnitWebConfig：一个组合的注解，它将@ExtendWith（SpringExtension.class）与来自 JUnit Jupiter 的@ContextConfiguration 和@WebAppConfiguration 在 Spring TestContext 框架中集成。

d)@EnabledIf:如果 SpEL 表达式或属性占位符的计算结果为 true，则表示启用注解的测试类或测试方法的信号。

e)@DisabledIf：如果 SpEL 表达式或属性占位符的计算结果为 true，则表示禁用注解的测试类或测试方法的信号。

b)支持在 SpringTestContext 框架中执行并行测试。

c)在 SpringTestContext 框架中,通过 SpringRunner(但不是通过 JUnit 4 规则)支持 TestNG、JUnit 5 和 JUnit 4,在测试执行回调之前和之后进行新的操作。

a)TestExecutionListener API 和 TestContextManager 中添加了新 beforeTestExecution()和 afterTestExecution()回调方法。

d）MockHttpServletRequest 现在有 getContentAsByteArray()和 getContentAsString()方法来访问内容(即请求正文)。

e)Spring MVC Test 中的 print()和 log()方法如果在模拟请求中设置了字符编码，则表示是打印请求正文。

f)Spring MVC Test 中的 redirected()和 forwardedUrl()方法现在支持可变扩展的 URI 模板。

g)XMLUnit 支持升级到 2.3 版本。

第二部分. Spring 框架 4.x 中的新特性

本章概述了 Spring 框架 4.3 中引入的新功能和改进。如果您对更详细的信息感兴趣，请查看作为 4.3 开发过程的一部分而解决的 Issue Tracker tickets[38]。

3. Spring 框架 4.0 中的新特性和新功能

Spring 框架第一个版本发布于 2004 年，自发布以来已历经三个主要版本更新:Spring 2.0 提供了 XML 命名空间和 AspectJ 支持；Spring 2.5 增加了注释驱动（annotation-driven）的配置支持；Spring 3.0 增加了对 Java 5+版本的支持和 Java-based @Configuration 模型。

Spring 4.0 是最新的主要版本【当然现在最新版本是 5.0.0】，并且首次完全支持 Java8 的特性。你仍然可以使用老版本的 Java，但是最低版本的要求已经提高到 Java SE 6。我们也借主要版本更新的机会删除了许多过时的类和方法。

你可以在 Spring Framework GitHub Wiki[39]上查看升级 Spring 4.0[40]的迁移指南。

3.1 改进的入门体验

新的 spring.io[41]网站提供了整个系列的"入门指南[42]"帮助你学习 Spring。你可以本文档的第一章 “Getting Started with Spring” 阅读更多的入门指南。新网站还提供了 Spring 之下其他额外项目的一个全面的概述。

如果你是一个 Maven 用户，你可能会对 BOM 这个有用的 POM 文件感兴趣[43]，这个文件已经与每个 Spring 的发布版发布。

3.2 移除过时的包和方法

所有过时的包和许多过时的类和方法已经从 Spring 4.0 中移除。如果你从之前的发布版升级 Spring，你需要保证已经修复了所有使用过时的 API 方法。

查看完整的变化：API 差异报告[44]。

请注意，所有可选的第三方依赖都已经升级到了最低版本是 2010/2011 年的(例如 Spring 4 通常只支持 2010 年的最新或者现在的最新发布版本):尤其是 Hibernate 3.6+、EhCache 2.1+、Quartz 1.8+、Groovy 1.8+、Joda-Time 2.0+。但是有一个例外，Spring 4 依赖最近的 Hibernate Validator 4.3+，现在对 Jackson 的支持集中在 Spring 2.0+版本(Spring 3.2 支持的 Jackson 1.8/1.9，现在已经过时）。

---

38 https://jira.spring.io/issues/?jql=project%20%3D%20SPR%20AND%20fixVersion%20in%20(%224.3%20RC1%22%2C%20%224.3%20RC2%22%2C%20%224.3%20GA%22)%20ORDER%20BY%20issuetype%20DESC&startIndex=50

39 https://github.com/spring-projects/spring-framework/wiki

40 https://github.com/spring-projects/spring-framework/wiki/Migrating-from-earlier-versions-of-the-spring-framework

41 https://spring.io/

42 https://spring.io/guides

43 https://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/htmlsingle/#overview-maven-bom

44 http://docs.spring.io/spring-framework/docs/3.2.4.RELEASE_to_4.0.0.RELEASE/

3.3 Java 8(以及 6 和 7)

Spring 4 支持 Java 8 的一些特性。你可以在 Spring 的回调接口中使用 lambda 表达式和方法引用。支持 java.time (JSR-310[45])的值类型和一些改进过的注解，例如@Repeatable。你还可以使用 Java 8 的参数名称发现机制（基于 -parameters 编译器标志）。

Spring 仍然兼容老版本的 Java 和 JDK：Java SE 6（具体来说，支持 JDK 6 update18）以上版本，我们建议新的基于 Spring 4 的项目使用 Java 7 或 Java 8。

3.4 Java EE 6 和 7

Java EE 6 或以上版本是 Spring 4 要求的最低版本,与 JPA 2.0 和 Servlet 3.0 规范有着特殊的意义。为了保持与 Google App Engine 和旧的应用程序服务器兼容,Spring 4 可以部署在 Servlet 2.5 运行环境。但是我们强烈的建议您在 Spring 测试和模拟测试的开发环境中使用 Servlet 3.0+。

注意：如果你是 WebSphere 7 的用户，一定要安装 JPA 2.0 功能包。在 WebLogic 10.3.4 或更高版本，安装附带的 JP A2.0 补丁。这样就可以将这两种服务器变成 Spring 4 兼容的部署环境。

从长远的观点来看，Spring 4.0 现在支持 Java EE 7 级别的适用性规范：尤其是 JMS 2.0,JTA 1.2,JPA 2.1,Bean Validation 1.1 和 JSR-236 并发工具类。像往常一样，支持的重点是独立的使用这些规范。例如在 Tomcat 或者独立环境中。但是，当把 Spring 应用部署到 Java EE 7 服务器时它同样适用。

注意，Hibernate 4.3 是 JPA 2.1 的提供者，因此它只支持 Spring 4。同样适用用于作为 Bean Validation 1.1 提供者的 Hibernate Validator 5.0。这两个都不支持 Spring 3.2。

3.5 Groovy Bean 定义 DSL

Spring 4.0 支持使用 Groovy DSL 来进行外部的 bean 定义配置。这在概念上类似于使用 XML 的 bean 定义,但是支持更简洁的语法。使用 Groovy 还允许您轻松地将 bean 定义直接嵌入到引导代码中。例如：

```
def reader = new GroovyBeanDefinitionReader(myApplicationContext)
reader.beans {
    dataSource(BasicDataSource) {
        driverClassName = "org.hsqldb.jdbcDriver"
        url = "jdbc:hsqldb:mem:grailsDB"
        username = "sa"
        password = ""
        settings = [mynew:"setting"]
    }
    sessionFactory(SessionFactory) {
        dataSource = dataSource
    }
    myService(MyService) {
        nestedBean = { AnotherBean bean ->
            dataSource = dataSource
        }
```

[45] https://jcp.org/en/jsr/detail?id=310

```
    }
}
```

有关更多信息，请参阅 GroovyBeanDefinitionReader 的 javadocs[46].

3.6 核心容器的改进

有几处对核心容器的常规改进：

a)Spring 现在注入 Bean 的时候把泛型类型当成一种形式的限定符（第 7.9.5 节）。例如：如果你使用 Spring Data Repository，你可以方便的插入特定的实现：@Autowired Repository<Customer> customerRepository。

b)如果你使用 Spring 的元注解支持，你现在可以开发自定义注解来公开源注解的特定属性（第 7.10.2 节）。

c)当自动装配到 lists 和 arrays 时，Beans 现在可以进行排序了（第 7.9.2 节）。支持@Order 注解和 Ordered 接口两种方式。 @Lazy 注解现在可以用在注入点以及@Bean 定义上。

d)引入@Description 注解,开发人员可以使用基于 Java 方式的配置（第 7.12.3.7 节）。

e)根据条件筛选 Beans 的广义模型(第 7.12.5.2 节)通过@Conditional 注解加入。这和@Profile 支持的类似，但是允许以编程式开发用户定义的策略。

e)基于 CGLIB 的代理类（第 12.5.3 节）不再需要默认的构造方法。这个支持是由 objenesis 库[47]提供。这个库重新打包到 Spring 框架中，作为 Spring 框架的一部分发布。通过这个策略，针对代理实例被调用没有构造可言了。

f)框架现在支持管理时区。例如 LocaleContext。

3.7 常规 Web 改进

现在仍然可以部署到 Servlet 2.5 服务器，但是 Spring 4.0 现在主要集中在 Servlet 3.0+环境。如果你使用 Spring MVC 测试框架（第 15.6 节），你需要将 Servlet 3.0 兼容的 JAR 包放到测试的 classpath 下。

除了稍后会提到的 WebSocket 支持外，下面的常规改进已经加入到 Spring 的 Web 模块：

a)你可以在 Spring MVC 应用中使用新的@RestController 注解（第 22.3.3.6 节），不在需要给@RequestMapping 的方法添加 @ResponseBody 注解。

b)AsyncRestTemplate 类已被添加进来,当开发 REST 客户端时,允许非阻塞异步支持(第 28.10.3 节）。

c)当开发 Spring MVC 应用时，Spring 现在提供了全面的时区支持（第 22.8.1 节）。

3.8 WebSocket，SockJS 和 STOMP 信息

新的 spring-websocket 模块提供了全面的基于 WebSocket 和在 Web 应用的客户端和服务器之间双向通信的支持。它和 Java WebSocket API JSR-356[48]兼容,此外还提供了当浏览器不支持 WebSocket 协议时(如 Internet Explorer<10)的基于 SockJS 的备用选项(如 WebSocket emulation)。

新的 spring-messaging 模块添加了支持 STOMP 作为 WebSocket 子协议用于在应用中使用注解编

[46] http://docs.spring.io/spring-framework/docs/4.3.10.RELEASE/javadoc-api/org/springframework/beans/factory/groovy/GroovyBeanDefinitionReader.html

[47] https://code.google.com/p/objenesis/

[48] https://jcp.org/en/jsr/detail?id=356

程模型路由和处理从 WebSocket 客户端发送的 STOMP 消息。因此@Controller 现在可以同时包含@RequestMapping 和@MessageMapping 方法用于处理 HTTP 请求和来自 WebSocket 连接客户端发送的消息。新的 spring-messaging 模块还包含了来自以前 Spring 集成项目的关键抽象[49]，例如 Message、MessageChannel、MessageHandler 等等,以此作为基于消息传递的应用程序的基础。

关于更加详细的细节，包括更加详细的介绍，请参考 26 章节：WebSocket 支持。

3.9 测试改进

除了精简 spring-test 模块中过时的代码外，Spring 4 还引入了几个用于单元测试和集成测试的新功能。

a) 几乎 spring-test 模块中所有的注解（例如： @ContextConfiguration 、@WebAppConfiguration、@ContextHierarchy、@ActiveProfiles 等等)现在可以用作元注解（第 15.4.4 节）来创建自定义的 composed annotations 并且可以减少测试套件的配置。

b) 现在可以以编程方式解决 Bean 定义配置文件的激活。只需要实现自定义的 ActiveProfilesResolver，并且通过@ActiveProfiles 的 resolver 属性注册。

c)新的 SocketUtils 类被引入到了 spring-core 模块。这个类可以使你能够扫描本地主机的空闲的 TCP 和 UDP 服务端口。这个功能不是专门用在测试的，但是可以证明在你使用 Socket 写集成测试的时候非常有用。例如测试内存中启动的 SMTP 服务器，FTP 服务器，Servlet 容器等。

d)从 Spring 4.0 开始,org.springframework.mock.web 包中的一套 mock 是基于 Servlet 3.0 API。此外，一些 Servlet API mocks（例如：MockHttpServletRequest、MockServletContext 等等）已经有一些小的改进更新，提高了可配置性。

4. Spring 4.1 中的新特性和功能改进

Spring 4.1 版本有很多的改进，具体如下：

a)第 4.1 节 “JMS 改进”

b)第 4.2 节 “缓存改进”

c)第 4.3 节 “Web 改进”

d)第 4.4 节 “WebSocket 消息改进”

e)第 4.5 节 “测试改进”

4.1 JMS 改进

Spring 4.1 引入更简单的基础架构，使用@JmsListener 注解 bean 方法来注册 JMS 监听端点（第 30.6 节）。XML 命名空间已经通过增强来支持这种新的方式（jms:annotation-driven），它也可以完全通过 Java 配置(@EnableJms, JmsListenerContainerFactory)来配置架构。也可以使用 JmsListenerConfigurer 注解来注册监听端点。

Spring 4.1 还调整了 JMS 的支持，使得你可以从 spring-messaging 在 Spring 4.0 引入的抽象获益，即：

a)消息监听端点可以有更为灵活的签名，并且可以从标准的消息注解获益，例如@Payload、@Header、@Headers 和@SendTo 注解。另外，也可以使用标准的消息，以代替 javax.jms.Message 作为方法参数。

b)一个新的可用 JmsMessageOperations 接口和允许操作使用 Message 抽象的 JmsTemplate。

最后，Spring 4.1 提供了其他各种各样的改进：

[49] http://projects.spring.io/spring-integration/

a)JmsT emplate 中的同步请求-答复操作支持

b)监听器的优先权可以指定每个<jms:listener/>元素

c)消息侦听器容器恢复选项可以通过使用 BackOff 实现进行配置

d)JMS 2.0 消费者支持共享

4.2 缓存改进

Spring 4.1 支持 JCache (JSR-107)注解（第 36.4 节）使用 Spring 的现有缓存配置和基础结构的抽象；使用标准注解不需要任何更改。

Spring 4.1 也大大提高了自己的缓存抽象：

a)缓存可以在运行时使用 CacheResolver 解决。因此使用 value 参数定义的缓存名称不在是强制性的。

b)更多的操作级自定义项：缓存解析器，缓存管理器，键值生成器。

c)一个新的@CacheConfig 类级别注解（第 36.3.5 节）允许在类级别上共享常用配置，不需要启用任何缓存操作。

d)使用 CacheErrorHandler 更好的处理缓存方法的异常。

Spring 4.1 在 CacheInterface 添加新的 putIfAbsent 方法，并做了重大的更改。

4.3 Web 改进

a) 现有的基于 ResourceHttpRequestHandler 的资源处理已经扩展了新的抽象 ResourceResolver，ResourceTransformer 和 ResourceUrlProvider。一些内置的实现提供了版本控制资源的 URL（有效的 HTTP 缓存），定位 gzip 压缩的资源，产生 HTML 5 AppCache 清单，以及更多的支持。参见第 22.16.9 节“资源服务”。

b)JDK 1.8 的 java.util.Optional 现在支持 @RequestParam， @RequestHeader 和 @MatrixVariable 控制器方法的参数。

c)ListenableFuture 支持作为返回值替代 DeferredResult 所在的底层服务（或者调用 AsyncRestTemplate）已经返回 ListenableFuture。

d)@ModelAttribute 方法现在依照相互依存关系的顺序调用。见 SPR-6299[50]。

e)Jackson 的@JsonView 被直接支撑在@ResponseBody 和 ResponseEntity 控制器方法用于序列化不同的细节对于相同的 POJO（如摘要与细节页）。同时通过添加序列化视图类型作为模型属性的特殊键来支持基于视图的渲染。见第 22.3.3.19 节 “Jackson 序列化视图支持”)

f)Jackson 现在支持 JSONP，见第 22.3.3.20 “Jackson JSONP 支持”，新的生命周期选项可用于在控制方法返回后，响应被写入之前拦截@ResponseBody 和 ResponseEntity 方法。要充分利用声明@ControllerAdvicebean 实现 ResponseBodyAdvice。为@JsonView 和 JSONP 的内置支持利用这一优势。参见第 22.4.1 节“使用 HandlerInterceptor 拦截请求”章节。

下面是 3 个新的 HttpMessageConverter 选项：

a)GSON-比 Jackson 更轻量级的封装;已经被使用在 Spring Android【据反馈 gson 的 bug 不是一般的多】。

b)Google Protocol Buffers-高效和有效的企业内部跨业务的数据通信协议，但也可以用于浏览器的 JSON 和 XML 的扩展

c)Jackson 基于 XML 序列化，现在通过 jackson-dataformat-xml 扩展得到了支持。如果 jackson-dataformat-xml [51] 在 classpath， 默认情况下使用 @EnableWebMvc 或

[50] https://jira.spring.io/browse/SPR-6299

[51] https://github.com/FasterXML/jackson-dataformat-xml

<mvc:annotation-driven/>，这是，而不是 JAXB2。

g)如 JSP 等视图现在可以通过名称参照控制器映射建立链接控制器。默认名称分配给每一个@RequestMapping。例如 FooController 的方法与 handleFoo 被命名为“FC#handleFoo”。命名策略是可插拔的。另外，也可以通过其名称属性明确命名的@RequestMapping。在 Spring JSP 标签库的新 mvcUrl 功能使这个简单的 JSP 页面中使用。参见第 22.7.3 节 “在视图中为控制器和方法指定 URI” 。

h)ResponseEntity 提供了 builder 风格的 API 来指导控制器向服务器端的响应展示，例如，ResponseEntity.ok()。

i)RequestEntity 是一种新型的，提供了 builder 风格的 API 来引导客户端的 REST 响应 HTTP 请求的展示。

MVC 的 Java 配置和 XML 命名空间：

a)视图解析器现在可以配置包括内容协商的支持，请参见第 22.16.8 节 “视图解析器”。

b)视图控制器现在已经内置支持重定向和设置响应状态。应用程序可以使用它来配置重定向的 URL，404 视图的响应，发送“no content”的响应，等等。有些用例这里列出[52]。

c)现在内置路径匹配的自定义。参见第 22.16.11 节 “路径匹配”。

j) Groovy 的标记模板[53]支持（基于 Groovy 2.3）。见 GroovyMarkupConfigurer 和各自的 ViewResolver 和“视图”的实现。

4.4 WebSocket 消息的优化

a)SockJS(Java)客户端支持。查看 SockJsClient 和在相同包下的其他类。

b)发布新的应用上下文实践 SessionSubscribeEvent 和 SessionUnsubscribeEvent,用于 STOMP 客户端的订阅和取消订阅。

c)新的"websocket"级别。查看第 25.4.15 节 “WebSocket 的作用域”。

d)@SendToUser 仅仅靠会话就可以了,而不一定需要授权的用户。

e)@MessageMapping 方法使用。来代替/作为目录分隔符。查看 SPR-1 1660[54]。

f)STOMP/WebSocket 监听消息的收集和记录。查看第 25.4.17 节 “运行时监控”。

g)显著优化和改善在调试模式下依然保持日志的可读性和简洁性。

h)优化消息创建，包含对临时消息可变性的支持和避免自动消息 ID 和时间戳的创建。查看 MessageHeaderAccessor 的 Java 文档。

i)在 WebSocket 会话建立之后的一分钟没有任何活动,则关闭 STOMP/WebSocket 连接。查看 SPR-11884[55]。

4.5 测试改进

Groovy 脚本可以在 ApplicationContext 中配置，在 TestContext 框架中集成加载。

详见第 15.5.4.2 节 “使用 Groovy 脚本的上下文配置”。

通过新的 TestTransaction API，使用编程化来开始结束测试管理事务。

详见第 15.5.7.4 节“可编程的事务管理”。

现在 SQL 脚本的执行可以通过 Sql 和 SqlConfig 注解申明在每一个类和方法中。

详见第 15.5.8 节“执行 sql 脚本”。

测试属性值可以通过配置@TestPropertySource 注解来自动覆盖系统和应用的属性值。

[52] https://jira.spring.io/browse/SPR-11543?focusedCommentId=100308&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-100308

[53] http://groovy-lang.org/docs/groovy-2.3.6/html/documentation/markup-template-engine.html

[54] https://jira.spring.io/browse/SPR-11660

[55] https://jira.spring.io/browse/SPR-11884

详见第 15.5.4.8 节“测试 property sources 的上下文配”。

默认的 TestExecutionListeners 现在可以自动被发现。

详见第 15.5.3.2 节“默认 TestExecutionListeners 的自动发现”。

自定义 TestExecutionListeners 现在可以通过默认的监听器自动合并。

详见第 15.5.3.2 节“合并 TestExecutionListeners”。

在 TestContext 框架中的测试事务方面的文档已经通过更多解释和其他事例得到改善。

详见第 15.5.7 节“事务管理”。

a)MockServletContext、MockHttpServletRequest 和其他 Servlet 接口 mocks 等诸多改善。

b)AssertThrows 重构后，Throwable 代替 Exception。

c)Spring MVC 测试中，使用 JSONPath 选项返回 JSON 格式可以使用 JSONAssert[56]来断言,非常像之前的 XML 和 XMLUnit。

d)MockMvcBuilder 现在可以在 MockMvcConfigure 的帮助下创建。MockMvcConfigurer 的加入使得 SpringSecurity 的设置更加简单，同时使用于任何第三方的普通封装设置或则仅仅在本项目中。

e)MockRestServiceServer 现在支持 AsyncRestTemplate 作为客户端测试。

5. Spring 4.2 新特性和改进

Spring 4.2 包含了很多改进，具体如下:

a)第 5.1 节 “核心容器改进”

b)第 5.1 节 “数据访问改进”

c)第 5.1 节 “JMS 改进”

d)第 5.1 节 “Web 改进”

e)第 5.1 节 “WebSocket 消息改进”

f)第 5.1 节 “测试改进”

5.1 核心容器改进

a)如@bean 注释,就如同得到发现和处理 Java 8 默认方法一样,可以允许组合配置类与默认@bean 接口方法。

b)配置类现在可以声明@import 作为常规组件类,允许引入的配置类和组件类进行混合。

c)配置类可以声明@Order 值,用来得到相应的处理顺序(例如重写 bean 的名字),即使通过类路径扫描检测。

d)@Resource 注入点支持@Lazy 声明,类似于@autowired,用于接收用于请求目标 bean 的懒初始化代理。

e)现在的应用程序事件基础架构提供了一个基于注解的模型[57]以及发布任意事件的能力。

a)任何受管 bean 的公共方法使用 @EventListener 注解来消费事件。

b)@TransactionalEventListener 提供事务绑定事件支持。

f)Spring 框架 4.2 引入了一流的支持声明和查找注释属性的别名。新@AliasFor 注解可用于声明一双别名属性在一个注释中或从一个属性在一个声明一个别名定义注解在元注释一个属性组成。

a)下面的注解已加了@AliasFor 为了支持提供更有意义的 value 属性的别名:@Cacheable, @CacheEvict, @CachePut, @ComponentScan, @ComponentScan.Filter, @ImportResource, @Scope, @ManagedResource, @Header, @Payload, @SendToUser, @ActiveProfiles,

[56] https://github.com/skyscreamer/JSONassert

[57] https://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/htmlsingle/#context-functionality-events-annotation

@ContextConfiguration, @Sql, @TestExecutionListeners, @TestPropertySource, @Transactional, @ControllerAdvice, @CookieValue, @CrossOrigin, @MatrixVariable, @RequestHeader, @RequestMapping, @RequestParam, @RequestPart, @ResponseStatus, @SessionAttributes, @ActionMapping, @RenderMapping, @EventListener, @TransactionalEventListener。

b)例如，spring-test 的@ContextConfiguration 现在声明如下：

```
public @interface ContextConfiguration {

    @AliasFor("locations")
    String[] value() default {};

    @AliasFor("value")
    String[] locations() default {};

    // ...
}
```

c)同样,组合注解（composed annotations）从元注解覆盖的属性,现在可以使用@AliasFor 进行细粒度控制哪些属性是覆盖在注释的层次结构。事实上,现在可以声明别名给元注释的 value 属性。

e)例如，开发组合注解用于自定义的属性的覆盖。

```
@ContextConfiguration
public @interface MyTestConfig {

    @AliasFor(annotation = ContextConfiguration.class, attribute = "value")
    String[] xmlFiles();

    // ...
}
```

f)见 Spring 注解编程模式[58]。

g)例如，开发组合注解用于自定义的属性的覆盖许多改进 Spring 的搜索算法用于寻找元注解。例如,局部声明组合注解现在喜欢继承注解。

h)从元注解覆盖属性的组合注解，可以被发现在接口和 abstract,bridge,&interface 方法就像在类，标准方法，构造函数，和字段。

i)Map 表示的注解属性(和 AnnotationAttributes 实例)可以 synthesized(合成,即转换)成注解。

j)基于字段的数据绑定的特点(DirectFieldAccessor)与当前的基于属性的数据绑定关联(BeanWrapper)。特别是,基于字段的绑定现在支持集合,数组和 Map 的导航。

k)DefaultConversionService 现在提供开箱即用的转化器给 Stream,Charset,Currency,和 TimeZone.这些转换器可以独立的添加到任何 ConversionService。

58 https://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/htmlsingle/#annotation-programming-model

l)DefaultFormattingConversionService 提供开箱即用的支持 JSR-354 的 Money&Currency 类型(前提是'javax.money'API 出现在 classpath):这些被命名为 MonetaryAmount 和 CurrencyUnit。支持使用@NumberFormat

m)@NumberFormat 现在作为元注解使用

m)JavaMailSenderImpl 中新的 testConnection()方法用于检查与服务器的连接

o)ScheduledT askRegistrar 用于暴露调度的任务

p)Apachecommons-pool2 现在支持用于 AOP CommonsPool2T argetSource 的池化

q)引入 StandardScriptFactory 作为脚本化 bean 的 JSR-223 的基本机制，通过 XML 中的 lang:std 元素暴露。支持如 JavaScript 和 JRuby。（注意：JRubyScriptFactory 和 lang:jruby 现在不推荐使用了,推荐用 JSR-223。

5.2 数据访问改进

a)javax.transaction.Transactional 现在可以通过 AspectJ 支持

b)SimpleJdbcCallOperations 现在支持命名绑定

c)完全支持 HibernateORM5.0:作为 JP A 供应商(自动适配)和原生的 API 一样(在新的 org.springframework.orm.hibernate5 包中涵盖了该内容)

d)嵌入式数据库可以自动关联唯一名字，并且 <jdbc:embedded-database>支持新的 database-name 属性。见下面“测试改进”内容。

5.3 JMS 改进

a)autoStartup 属性可以通过 JmsListenerContainerFactory 进行控制

b)应答类型 Destination 可以配置在每个监听器容器

c)@SendTo 的值可以用 SpEL 表达式

d)响应目的地可以通过 JmsResponse 在运行时计算[59]

e)@JmsListener 是可以可以重复的注解用于声明多个 JMS 容器在相同的方法上(若你还没有用上 Java 8 请使用新引入的@JmsListeners)。

5.4 Web 改进

a)支持 HTTPStreaming 和 Server-SentEvents,见第 22.3.4.3 节 “HTTP 流”。

b)内建支持 CORS，包括全局(MVCJava 配置和 XML 命名空间)和本地(如@CrossOrigin)配置。见第 27 章：CORS 支持。

c)HTTP 缓存改进

a) 新的 CacheControl 构建器；插入 ResponseEntity,WebContentGenerator,ResourceHttpRequestHandler

b)改进的 ETag/Last-Modified 在 WebRequest 中的支持

d)自定义映射注解，使用@RequestMapping 作为元数据注解

e)AbstractHandlerMethodMapping 中的 public 方法用于运行时注册和注销请求映射

f)AbstractDispatcherServletInitializer 中的 ProtectedcreateDispatcherServlet 方法用来进一步自定义 DispatcherServlet 实例

g)HandlerMethod 作为@ExceptionHandler 方法的方法参数,特别是方便@ControllerAdvice

[59] https://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/htmlsingle/#jms-annotated-response

组件

h)java.util.concurrent.CompletableFuture 作为@Controller 方法返回值类型

i)字节范围（Byte-range）的请求支持在 HttpHeaders，用于静态资源

j)@ResponseStatus 发现嵌套异常。

k)在 RestTemplate 中的 UriTemplateHandler 扩展。

a)DefaultUriT emplateHandler 公开 baseUrl 属性和路径段的编码选项

b)扩展端点可以使用插入任何 URI 模板库

l)OkHTTP[60]与 RestTemplate 集成

m)自定义 baseUrl 在 MvcUriComponentsBuilder 选择方法。

n)序列化/反序列化异常消息现在记录为 WARN 级别。

o)默认的 JSON 前缀改变了从"{} && "改为更安全的")]}', "。

p)新的 RequestBodyAdvice 扩展点和内置的实现支持 Jackson 的在@RequestBody 的@JsonView

q)当使用 GSON 或 Jackson 2.6+,处理程序方法的返回类型是用于提高参数化类型的序列化，比如 List<Foo>

r)引入的 ScriptTemplateView 作为 JSR-223 的脚本化 Web 视图机制为基础,关注 JavaScript 视图模板 Nashorn(JDK 8)。

5.5 WebSocket 消息改进

a)公开已经连接的用户和订阅的信息。

a)新 SimpUserRegistry 公开为名为“userRegistry”的 bean。

b)共享在服务器集群的展示信息(见代理中继配置选项)

b)解决用户目的地在集群的服务器(见代理中继配置选项)。

c)StompSubProtocolErrorHandler 扩展端点用来自定义和控制 STOMP ERROR 帧给用户。

d)全局@MessageExceptionHandler 方法通过@ControllerAdvice 组件。

e)心跳和 SpEL 表达式'selector'头用 SimpleBrokerMessageHandler 订阅。

f)STOMP 客户端使用 TCP 和 WebSocket；见第 25.4.14 节 “STOMP 客户端”。

g)@SendTo 和@SendToUser 可以包含目标变量的占位符。Jackson 的@JsonView 支持@MessageMapping 和@SubscribeMapping 方法返回值。

h)ListenableFuture 和 CompletableFuture 是从@MessageMapping 和@SubscribeMapping 方法返回类型值

i)MarshallingMessageConverter 用于 XML 负载。

5.6 测试改进

a)基于 JUnit 集成测试现在可以执行 JUnit 规则而不是 SpringJUnit4ClassRunner。这允许基于 Spring 的集成测试与运行 JUnit 的 Parameterized 或第三方库。

详见第 15.5.9.2 节 “Spring JUnit 4 规则”。

b)Spring MVC Test 框架，现在支持第一类 HtmlUnit，包括集成 Selenium’s WebDriver,允许基于页面的 Web 应用测试而无需部署到 Servlet 容器。

参看第 15.6.2 节 “HtmlUnit 集成”。

c)AopTestUtils 是新的测试工具，允许开发者获得潜在的目标对象的引用隐藏在一个或多个 Spring 代理。

参看第 14.2.1 节 “ 通用测试功能章节”。

[60] https://square.github.io/okhttp/

d)ReflectionT estUtils 现在支持 setting 和 gettingstatic 字段,包括常量

e)bean 定义归档文件的原始顺序，通过@ActiveProfiles 声明，现在保留为了支持用例,如 Spring 的 ConfigFileApplicationListener 引导加载配置文件基于活动归档文件的名称。

f)@DirtiesContext 支持新 BEFORE_METHOD，BEFORE_CLASS，BEFORE_EACH_TEST_METHOD 模式，用于测试之前关闭。

g)ApplicationContext——例如,如果一些烦人的(即，有待确定)测试在大型测试套件的 ApplicationContext 的原始配置已经损坏。

h)@Commit 是新的注解直接可以用来代替@Rollback(false)

i) @Rollback 用来配置类级别的默认回滚语义

a)因此,现在的@TransactionConfiguration 被弃用,在后续版本将被删除。

j) @Sql 现在支持内联 SQL语句的执行通过一个新的 statements 属性

k)ContextCache 用于缓存测试之间的 ApplicationContext，而现在这是一个公开的 API，默认的实现可以替代自定义的缓存需求

l)DefaultTestContext,DefaultBootstrapContext, 和 DefaultCacheAwareContextLoaderDelegate 现在是公开的类，支持子包，允许自定义扩展

m)TestContextBootstrapper 现在负责构建 TestContext

n)在 Spring MVC Test 框架，MvcResult 详情可以被日志记录在 DEBUG 级别或者写入自定义的 OutputStream 或 Writer。详见 log(),print(OutputStream),和 MockMvcResultHandlers 的 print(Writer)方法

o)JDBCXML 名称空间支持一个新的 <jdbc:embedded-database>的 database-name 属性,允许开发人员为嵌入式数据库设置独特的名字–例子通过 SpEL 表达式或前活动 bean 定义配置文件所影响的占位符属性

p)嵌入式数据库现在可以自动分配唯一的名称,允许常用的测试数据库配置在不同的 ApplicationContext 的测试套件中。

见 19.8.6 节“为嵌入的数据库生成唯一的名字”。

q)MockHttpServletRequest 和 MockHttpServletResponse 现在通过 getDateHeader 和 setDateHeader 方法为日期标头格式提供更好的支持。

6. Spring 4.3 的新特性和改进

第 Spring 4.3 包含的新改进，具体如下：

a)第 6.1 节 “核心容器改进”

b)第 6.1 节 “数据访问改进”

c)第 6.1 节 “缓存改进”

d)第 6.1 节 “JMS 改进”

e)第 6.1 节 “Web 改进”

f)第 6.1 节 “WebSocket 消息改进”

h)第 6.1 节 “测试改进”

i)第 6.1 节 “新的库和访问的支持”

6.1 核心容器改进

a)核心容器额外提供了更丰富的元数据来改进编程。

b)默认 Java 8 的方法检测为 bean 属性的 getter/setter 方法。

c)如果目标 bean 只定义了构造函数,则它无需要指定@Autowired 注解

d)@Configuration 类支持构造函数注入。

e)任何 SpEL 表达式用于指定@EventListener 的 condition 引用到 bean（例如@beanName.method()）。

f)组成注解现在可以用一个包含元注解中的数组属性的数组组件类型的元素来覆盖。例如，@RequestMapping 的的 String[]path 可以在组成注解用 String path 覆盖。

g)@Scheduled 和@Schedules 现在是作为元注解用来通过属性覆盖来创建自定义的组成注解。

h)@Scheduled 适当支持任何范围内的 bean。

6.2 数据访问优化

jdbc:initialize-database 和 jdbc:embedded-database 支持可配置的分离器被应用到每个脚本。

6.3 缓存优化

Spring 4.3 允许在给定的 key 并发调用时实现要同步，使得相应的值只计算一次。这是一个可选的功能，通过设置@Cacheable 的新的 sync 属性来启用。此功能引入了 Cache 接口的重大更改，即添加了 get(Object key,Callable<T> valueLoader)方法。

Spring 4.3 也改进了抽象缓存，如下:

a)SpEL 表达式对于缓存相关的注解，现在可以引用 bean（即@beanName.method()）。

b)ConcurrentMapCacheManager 和 ConcurrentMapCache 现在通过一个新的 storeByValue 属性支持缓存实体的序列化。@Cacheable，@CacheEvict，@CachePut 和@Caching 现在是作为元注解用来通过属性覆盖来创建自定义的组成注解。

c)@Cacheable、@CacheEvict、@CachePut 和@Caching 现在可以用作 meta-annotations 来创建具有属性重写的自定义组合批注。

6.4 JMS 的优化

a)@SendTo 现在可以在类级别指定一个共同回复目标。

b)@JmsListener 和@JmsListeners 现在是作为元注解用来通过属性覆盖来创建自定义的组成注解。

6.5 Web 优化

a)内部对 HTTP HEAD 和 HTTP OPTIONS 的支持，见第 22.3.2.5 节。

b)新的组合注解 @GetMapping,@PostMapping,@PutMapping,@DeleteMapping, 和 @PatchMapping 用于@RequestMapping。

详见第 22.3.2.1 节 “组合@RequestMapping 变量”。

c)新的@RequestScope、@SessionScope 和@ApplicationScope 可以用于 Web 作用域的组合注解。

详见 Request 作用域、Session 作用域和 Application 作用域（见 7.5.4 等节）。

d)新的@RestControllerAdvice 注解是@ControllerAdvice 和@ResponseBody 的语义结合。

e)@ResponseStatus 现在在类级别被支持，并被所有方法继承。

f)新的@SessionAttribute 注解用于访问 session 属性(见例子，第 22.3.3.11 节) 。

h)新的@RequestAttribute 注解用于访问请求属性(见例子，第 22.3.3.12 节) 。

i)@ModelAttribute 允许通过 binding=false 来避免数据绑定(见引用，第 22.3.3.8 节) 。

j)@PathVariable 可以声明为可选(用于@ModelAttribute 方法)。

k)错误和自定义抛出，将被统一到 MVC 异常处理器中处理。

l)HTTP 消息转换编码一致处理，包括默认 UTF-8 用于多部分文本内容。

m)静态资源处理使用配置的 ContentNegotiationManager 用于媒体类型计算。

n)RestTemplate 和 AsyncRestTemplate 支持通过 DefaultUriTemplateHandler 来实现严格的 URI 变量编码。

o)AsyncRestTemplate 支持请求拦截。

6.6 WebSocket 消息优化

@SendTo 和@SendToUser 现在可以在类级被指定为共享共同的目的地。

6.7 测试优化

a)为了支持 Spring TestContext Framework，现在需要 JUnit 4.12 或者更高的版本

b)新的 SpringRunner 关联到 SpringJUnit4ClassRunner

c)测试相关的注解，现在可以在接口上声明了。例如，基于 Java 8 的接口上使用测试接口

d)空声明的@ContextConfiguration 现在将会完全忽略，如果检测到默认的 XML 文件,Groovy 脚本,或@Configuration 类型

e)@Transactional 测试方法不再需要 public(如,在 TestNG 和 JUnit 5)

f)@BeforeTransaction 和@AfterTransaction 不再需要 public，并且在基于 Java8 的接口的默认方法上声明

g)在 Spring TestContext Framework 的 ApplicationContext 的缓存现在有界为 32 默认最大规模和最近最少使用驱逐策略。最大的大小可以通过设置称为 spring.test.context.cache.maxSize JVM 系统属性或 Spring 配置。

h)ContextCustomizerAPI 用于自定义测试 ApplicationContext 在 bean 定义加载到上下文后但在上下文被刷新前。定制工具可以在全球范围由第三方进行注册，而无需要实现自定义的 ContextLoader。

i)@Sql 和@SqlGroup 现在作为元注解通过覆盖属性来创建自定义组合注解

j)ReflectionTestUtils 现在在 set 或 get 一个字段时，会自动解开代理。

k)服务器端的 Spring MVC 测试支持具有多个值的响应头。

l)服务器端的 Spring MVC 测试解析表单数据的请求内容和填充请求参数。

m)服务器端的 Spring MVC 测试支持 mock 式的断言来调用处理程序方法。

n)客户端 REST 测试支持允许指定多少次预期的请求以及期望的声明顺序是否应该被忽略（参见第 15.6.3 节“客户端 REST 测试”）。

o)客户端 REST 测试支持请求主体表单数据的预期。

6.8 新的库和服务器的支持

Hibernate ORM 5.2 (still supporting 4.2/4.3 and 5.0/5.1 as well, with 3.6 deprecated now)

Hibernate Validator 5.3 (minimum remains at 4.3)

Jackson 2.8 (minimum raised to Jackson 2.6+ as of Spring 4.3)

OkHttp 3.x (still supporting OkHttp 2.x side by side)

Tomcat 8.5 as well as 9.0 milestones
Netty 4.1
Undertow 1.4
WildFly 10.1
【这一段就白话文了，再翻译就丢智商了】

另外，Spring 4.3 中集成了 ASM 5.1、CGLIB 3.2.4 和 Objenesis 2.4 在 spring-core.jar 包中。

第三部分. 核心技术

这部分的参考文档涵盖了 Spring 框架中所有最最最重要的技术。

这其中最重要的部分就是 Spring 框架中的控制反转（IoC)容器。Spring 框架中 IoC 容器是紧随着 Spring 中面向切面编程（AOP)技术的全面应用的来完整实现的。Spring 框架有它自己的一套 AOP 框架，这套框架从概念上很容易理解，而且成功解决了 Java 企业级应用中 AOP 需求 80%的核心要素。

同样 Spring 与 AspectJ 的集成（目前从功能上来说是最丰富，而且也无疑是 Java 企业领域最成熟的 AOP 实现）也涵盖在内。

a)第 7 章 IoC 容器

b)第 8 章 资源

c)第 9 章 验证、数据绑定和类型转换

d)第 10 章 SpEL 表达式语言

e)第 11 章 使用 Spring 实现面向切面编程

f)第 12 章 Spring AOP APIs