Spring 框架参考文档-4.3.25-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：4.3.25.RELEASE

文档官网：https://docs.spring.io/spring/docs/4.3.25.RELEASE/spring-framework-reference/
现在官网的稳定版是 4.3.25 版本，翻译文档是完整版，前后历时 8 个月，数十次的修改和校对【本文档从 4.3.10 版本升级而来，忽略中间的版本，即从 2017.10.14-2019.10.15 的过渡升级】。另外 5.x.x 版本也是同步翻译的【5.0.0 的翻译早已完成】，会在适当的时间公布。

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，由于未润色，过程翻译可能有错有漏，可自行改正【毕竟本人非英语非计算机，双非专业】；
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 300 多；
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；

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；
3.如果有任何问题或者建议，欢迎来邮件交流：sekift@163.com。

陆印章
4.3.10 版本　2017 年 10 月 14 日　于广州
4.3.25 版本　2019 年 10 月 14 日　于广州

目 录

第一部分. Spring 框架的概述

Spring 框架是一个轻量级的解决方案，可以一站式地构建企业级应用。同时，Spring 是模块化的，这意味着你可以只引入所需的部分，而不是全部。IoC 容器可以用在任意的 Web 框架上，但也可以只使用 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 服务远程访问你的逻辑代码，并支持多种数据持久化方案。Spring 提供全功能的 MVC 框架（第 22.1 节），这样你能将 AOP（第 11.1 节）透明地集成到你的软件中。

Spring 的设计是非侵入性的，也就是说你的领域逻辑代码通常对框架本身无依赖性。在集成层中（如数据访问层），需要依赖数据访问技术和 Spring 某些库。但是，很容易就能将这些依赖从你的代码中分离开来。

本文档是 Spring 框架功能的参考指南。如果你有关于这个文档的任何要求、意见或问题，请发送到用户邮件列表[1]。对框架本身的问题应该到 StackOverflow 发问（见 https://spring.io/questions[2]）。

1. 开启 Spring 之旅

本参考指南提供关于 Spring 框架的详细信息。文档会介绍 Spring 包含的全部功能，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）”）。

如果你刚开始接触 Spring，你可能会开始使用 Spring 框架创建基于 Spring Boot[3]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 应用程序。它是基于 Spring 框架的，仅需少量的配置即可快速搭建可运行程序。

你可以使用 start.spring.io[4]创建基本项目或遵循类似于“动手创建 RESTful Web 服务”[5]的“入门”[6]指南。这些指南都非常专注于具体的任务，其中大部分基于 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某个特定问题时需要的其他 Spring 项目。

2. Spring 框架简介

Spring 框架提供完善的基础设施，用于支持 Java 开发应用程序。Spring 负责基础设施功能，而你可以专注于编写你的应用。

使用 Spring，你可以用“简单的 Java 对象”（POJO）构建应用程序，并且将 POJO 以非侵入性的形式应用到企业服务。此功能适用于 Java SE 编程模型以及完全或者部分的 Java EE。

举个例子，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)使用 Java 方法执行数据库事务而不用去处理事务 API。

~~b)使用本地 Java 方法执行远程过程而不用去处理远程 API。【4.3.25 版本已经删除】~~

b)使用本地 Java 方法作为 HTTP 端点而不用去处理 Servlet API。【4.3.25 版本增加】

c)使用本地 Java 方法执行管理操作而不用去处理 JMX API。

d)使用本地 Java 方法执行消息处理而不用去处理 JMS API。

---

[1] https://groups.google.com/forum/#!forum/spring-framework-contrib
[2] https://spring.io/questions
[3] http://projects.spring.io/spring-boot/
[4] http://start.spring.io/
[5] https://spring.io/guides/gs/rest-service/
[6] https://spring.io/guides

2.1 依赖注入和控制反转

Java 应用程序--运行在各种各样的设备上，从受限的嵌入式应用程序，到多层架构的服务端企业级应用程序--通常会将应用的对象组合在一起工作。因此，对象在应用程序中是彼此依赖的。

尽管 Java 平台提供了丰富的应用程序开发功能，但它缺乏用来组织基本构件的完整方法。这个任务留给了架构师和开发人员。虽然你可以使用设计模式，例如 Factory、Abstract Factory、Builder、Decorator 和 Service Locator 来组合各种类和对象实例组成应用程序，这些模式关注的是：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，但开发者必须在应用程序中自行实现。

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，使其构成完整的、可用的应用。Spring 框架将规范化的设计模式作为一等对象，你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

“人们常常会问：[他们]哪些方面的控制被反转了？”这个问题由 Martin Fowler 在他的 Inversion of Control(IoC)网站于 2004 年提出[7]。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。这些模块分别是 Core Container ,Data Access/Integration,Web,AOP(Aspect Oriented Programming),Instrumentation,Messaging, 和 Test，如下图：

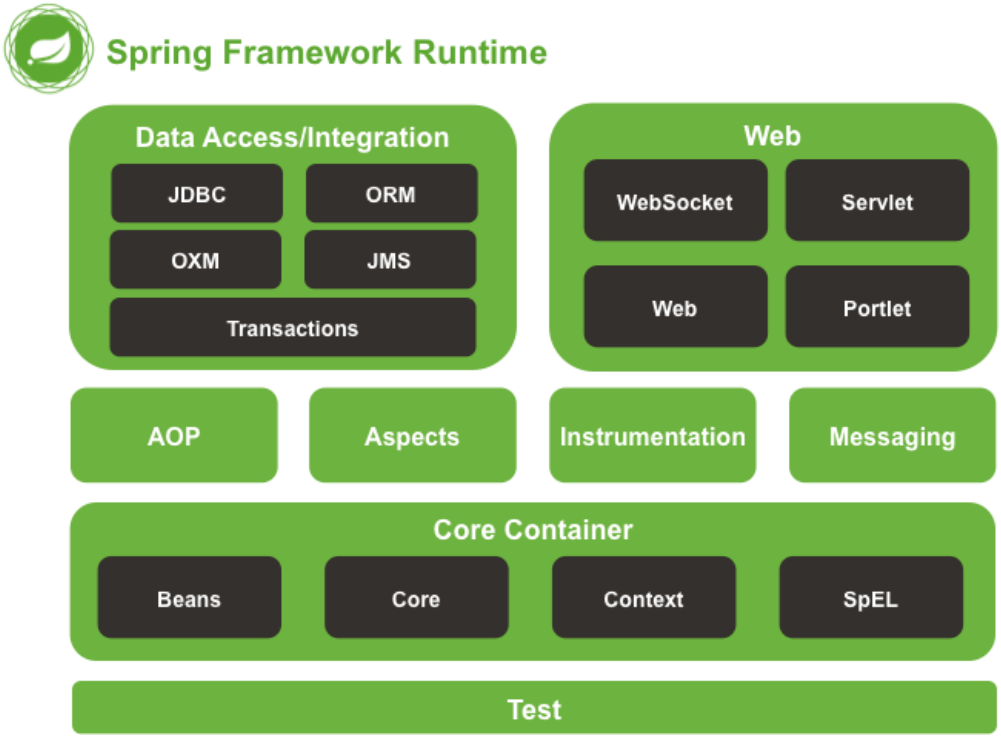

图 2.1 Spring 框架概览

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

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（SpEL 语言）模块组成。

spring-core 和 spring-beans 提供框架的基础功能（第 7.1 节），包括 IoC 和 Dependency Injection 功能。BeanFactory 是一个复杂的工厂模式实现，它舍弃了单例模式的侵入，允许你将配置和特定的依赖从实际程序逻辑中解耦。

Context(spring-context)（第 7.15 节）模块构建在 Core 和 Beans（第 7.1 节）模块的基础上，它提供了在框架范围内操作对象的方法，相当于 JNDI 注册功能。Context 继承了 Beans 模块的特性，并且增加了对国际化（例如用于 resource 包中）、事件传播、资源加载和创建上下文（例如 Servlet 容器）等功能。Context 模块也支持如 EJB，JMX 和基础远程【basic remoting】这样的 Java EE 特性。ApplicationContext 接口是 Context 模块的重点。spring-context-support 提供对常见第三方库的支持，使其集成到 Spring 应用上下文，例如缓存(EhCache,Guava,JCache)、通信(JavaMail)、调度(CommonJ,Quartz)以及模板引擎(FreeMarker,JasperReports,Velocity)。

spring-expression 模块提供了强大的 Expression Language（表达式语言，第 10 章）用来在运行时查询和操作对象。SpEL 是 JSP 2.1 规范指定的统一表达式语言（unified EL）的延伸。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的操作，并支持通过名称从 Spring 的控制反转容器中取回对象。表达式语言模块也支持 List 映射和选取操作，这与常用 List 是相同的。

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，并允许自定义注解。例如，方法拦截器和切入点能与程序代码完全解耦。使用源码级别元数据的功能，也可以在行为信息并入到逻辑代码中，这在某种程度上与.NET 的属性相似。

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

spring-instrument 模块提供了类插装【instrumentation】的支持以及在某些应用程序服务器中的类加载器实现。spring-instrument-tomcat 用于 Tomcat 类插装代理。

2.2.3 消息组件

Spring 4 框架包含 spring-messaging 模块，它从 Spring 集成项目中抽象出来，包含 Message,MessageChannel,MessageHandler 及其他用来提供基于消息的基础服务。该模块还包含一组消息映射方法的注解，类似于基于编程模型的 Spring MVC 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 模块隐藏了冗余繁杂的 JDBC 代码（第 19.1 节），并提供了为数据库厂商特有的错误代码而编写的 JDBC 抽象层。

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

spring-orm 模块提供了流行的对象/关系映射（object-relational mapping，第 20.1 节）API 集成层，包括 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 映射框架集成所有 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对象/XML 映射（Object/XML mapping，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 4.1 框架开始提供集成的 spring-messaging 模块。

2.2.5 Web

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

spring-web 模块提供了基本的面向 Web 开发的集成功能，例如多文件上传、Servlet listeners 和 Web 开发应用程序上下文初始化 IoC 容器。也包含 HTTP 客户端以及 Spring 有关 Web 部分的远程访问支持。

spring-webmvc 模块（也称为 Web Servlet 模块）包含 Spring 的模型-视图-控制器（model-view-controller 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 框架所有其他功能。

spring-webmvc-portlet 模块（也称为 Web-Portlet 模块）提供了用于 Portlet 环境的 MVC 模式实现以及 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）。它提供了持续加载(第 15.5.4 节)的 Spring ApplicationContext，并且会缓存(第 15.5.4.10 节）这些上下文。它还提供了模仿对象（mock object，第 14.1 节）功能，用于测试代码隔离。

2.3 应用场景

构建模块的模式，使 Spring 成为大众的、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 集成框架。

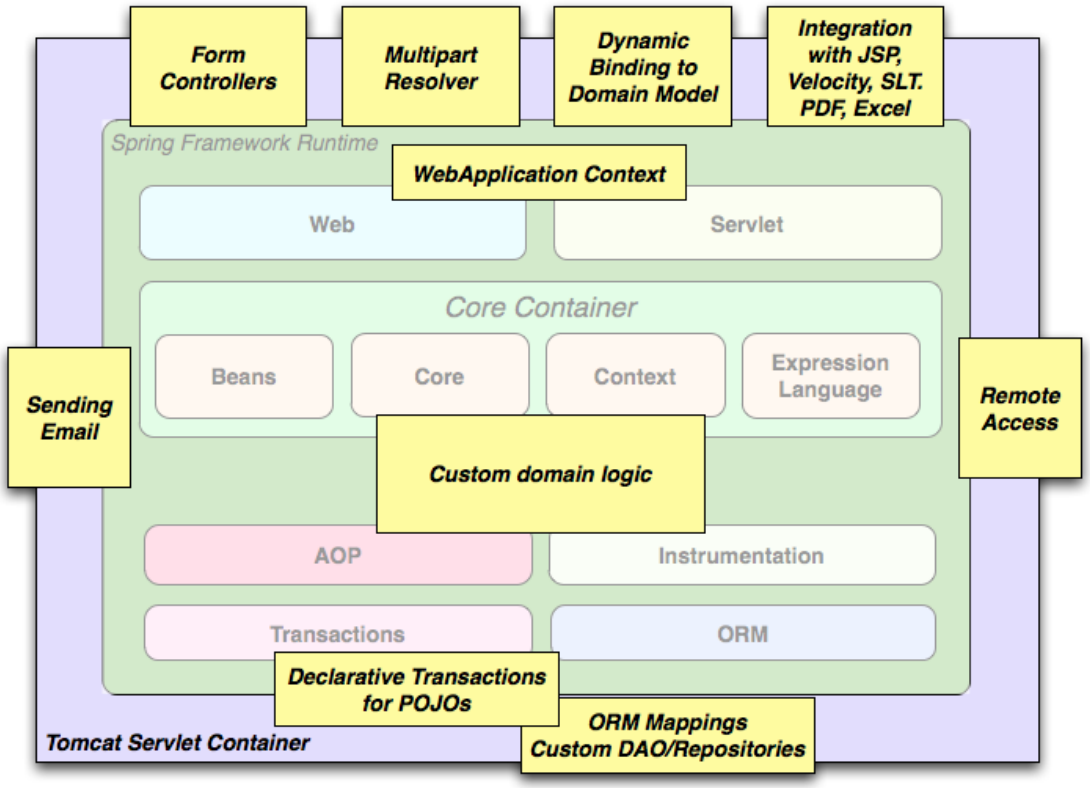

图 2.2 典型的、成熟的 Spring Web 应用

Spring 声明式事务管理特性（第 17.5 节）使 Web 应用程序拥有完整的事务管理功能，如同使用 EJB 容器管理事务那样。所有的自定义业务逻辑可以使用简单的 POJO 实现，并且使用 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，则由独立的网络层支持，这样可以让开发者自行选择在何处、何时执行规则验证。Spring ORM 模块适配 JPA，Hibernate，JDO 持久层；例如，使用 Hibernate，开发者可以继续使用现有的映射文件和标准的 Hibernate SessionFactory 配置。表单控制器将 Web 层和领域模型无缝集成，消除 ActionForms 或其他需要将 HTTP 参数转换为领域模型值的繁杂过程。

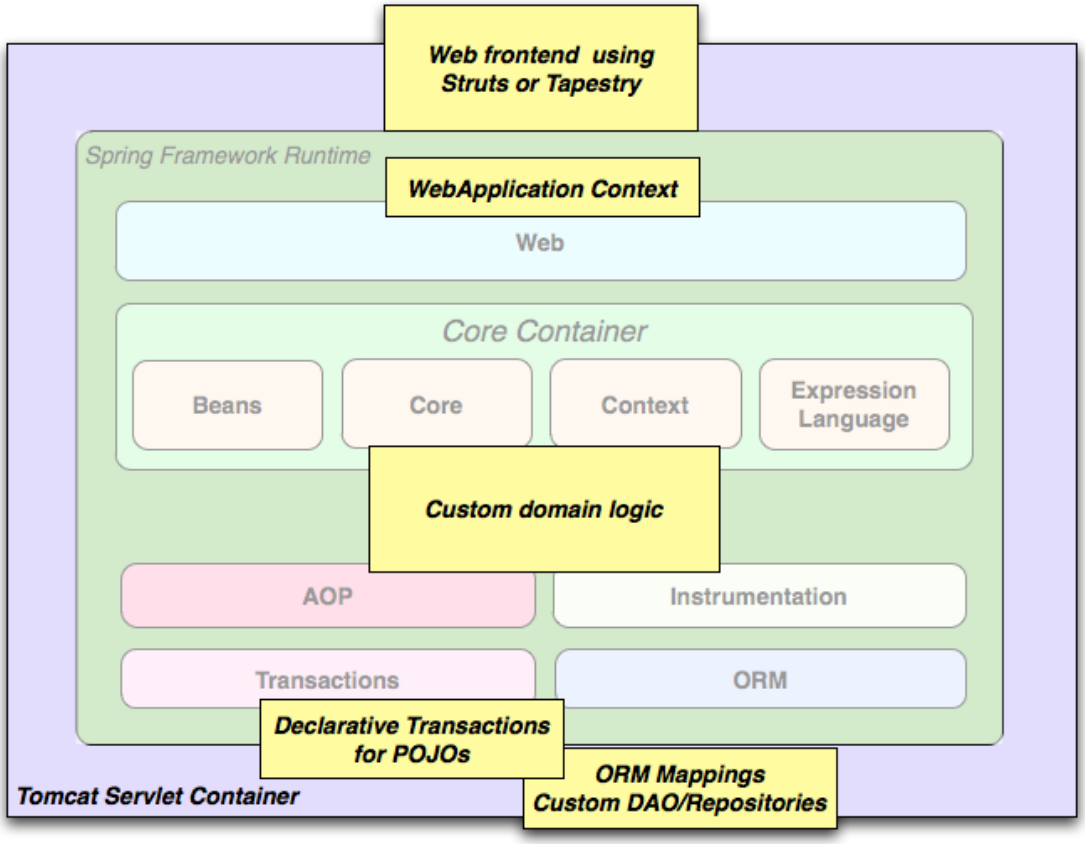

图 2.3 Spring 使用第三方 Web 框架工具的中间层

某些情况下不允许开发者完全地切换到不同的框架。Spring 框架不会强制使用它，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 的事务功能。你只需要在业务逻辑中使用 ApplicationContext 以及将 WebApplicationContext 集成到 Web 层即可。

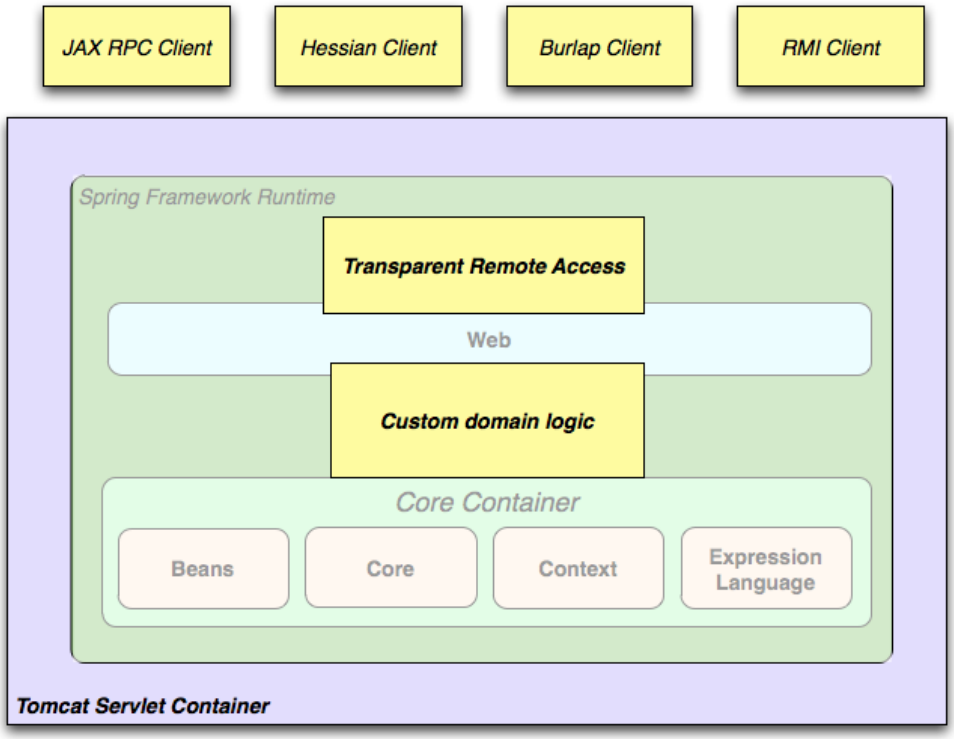

图 2.4 远程使用场景

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非常容易。

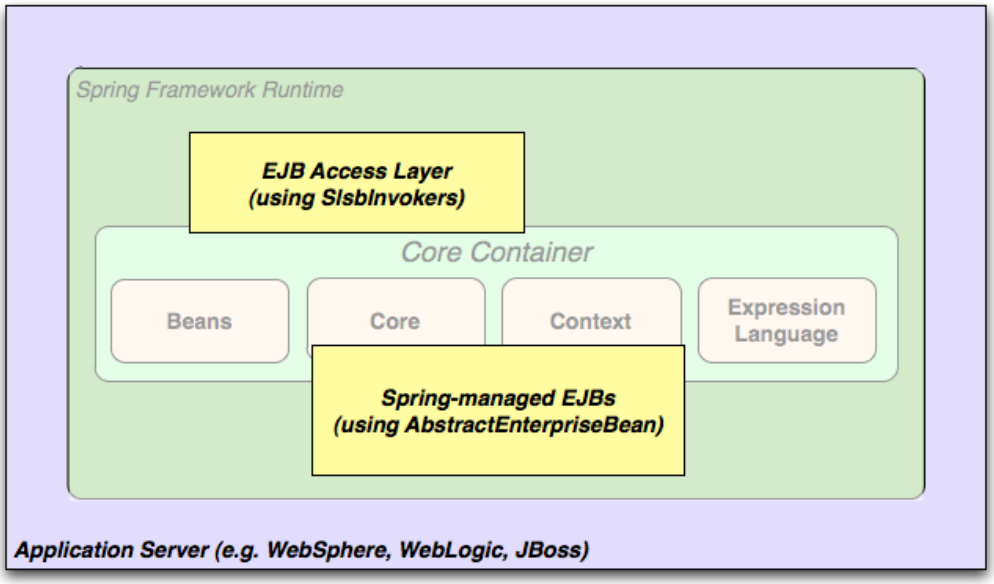

图 2.5 EJBs—包装已存在的 POJO

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

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 强大的功能应用到运行程序中（比如依赖注

入），需要引入功能对应的库（JAR 文件），并且在编译、运行的时候将它们引入到应用的类路径中。这些依赖不是虚拟组件的注入，而通常是文件系统的物理资源（大多数情况如此）。依赖管理过程包括定位资源、存储资源并添加到类路径。依赖可以是直接的（应用程序运行时依赖 Spring），或者间接的（应用程序依赖 commons-dbcp，而 commons-dbcp 又依赖 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。

如果确定使用 Spring，如何判断需要使用哪些 Spring jar 包呢。为了简化过程，Spring 已经被打包成一组组的模块，这些模块尽可能解耦，消除互相依赖。例如，例如，如果项目无需用到 Web 功能，就不必引入 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（如 spring-core,spring-webmvc,spring-jms 等）。实际的 jar 文件的名字，通常是使用模块名字和版本号区分的（例如 spring-core-4.3.0.RELEASE.jar）。

每个 Spring 框架发行版本将会放到下面仓库：

a)Maven Central（Maven 中央库），Maven 默认仓库，无需任何特殊的配置即可使用。许多常用的 Spring 依赖库也存放在 Maven Central 中，Spring 社区多数项目默认使用 Maven 进行项目管理，所以这是最方便的。jar 包的命名格式是 spring-*-<version>.jar，Maven groupId 是 org.springframework。

b)此外，Spring 也有自己的公共 Maven 库。除了最终的 GA 版本，这个库还保存开发的快照和里程碑文件。JAR 包的命名格式与 Maven Central 相同，这样部署 Spring 开发版本无需做过多的改动。该库还包含一个用于发布的 zip 文件，它含了 Spring 所有 jar 包，方便开发者下载。

首先，你要决定用什么方式管理你的依赖，推荐使用自动构建功能，例如 Maven,Gradle 或 Ivy，当然也可以自行下载 jar 后手动引入。

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 框架组件

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 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 模块使用的核心应用程序
org.springframework	spring-expression	Spring 表达式语言(SpEL)
org.springframework	spring-instrument	用于 JVM 引导的插装代理
org.springframework	spring-instrument-tomcat	用于 Tomcat 的插装代理
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 Spring 组件的单元测试和集成测试
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	应用于 Portlet 环境的 MVC 实现
org.springframework	spring-websocket	WebSocket 和 SockJS 实现，包括 STOMP 支持

2.3.1.1 Spring 依赖和依赖 Spring

Spring 集成了大量企业和其他外部工具的支持，它的模块设计将项目间的强依赖性降到了最低：在简单的用例中，开发者无需查找和下载（甚至是自动的）一大批 jar 包来使用 Spring。基本的依赖只有一个用来做日志的强制性外部依赖（下节将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，首先是 Maven，然后是 Gradle 和最后介绍 Ivy。在整个过程中，如果有不清楚的地方，可以查看的依赖管理工具的文档，或参考一些示例代码。Spring 本身是使用 Gradle 来管理依赖的【奇怪上面提到 Spring 很多项目是用 Maven 的】，因此这里也使用 Gradle 或 Maven 工具作为示例。

2.3.1.2 Maven 依赖管理

如果你是 Maven[8]来进行项目管理，你甚至无需明确配置日志依赖。例如，要创建应用程序上下文和使用依赖注入配置应用程序，你的 Maven 依赖看起来如下：

````
```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.25.RELEASE</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```
````

就是这么简单。注意 scope（作用域，范围）可声明为 runtime（运行时），如果你不需要编译 Spring API，这是使用依赖注入的最佳实践。

上面是使用 Maven Central 库的项目实例。如果使用 Spring Maven 存储库（如里程碑或开发者快照）的版本，需要配置指定的 Maven 存储位置。如下：

````
```
<repositories>
```
````

[8] 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 依赖清单

在使用 Maven 时，有可能下载了不同版本的 Spring JARs。例如，你可能发现第三方的库，或另一个 Spring 的项目下载了依赖前发布的包。如果开发者忘记配置直接依赖，可能会出现各种意想不到的问题。

为了克服这些问题，Maven 支持"bill of materials"(BOM)的依赖概念。开发者可以在 dependencyManagement 中引入 spring-framework-bom 来确保所有 Spring 依赖（包括直接和间接依赖）是同一版本。

````
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>4.3.25.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
````

使用 BOM 后，当依赖 Spring 框架组件时，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

使用 Gradle[9]来引入 Spring，只需在 repositories 中填入适当的 URL：

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

只需修改 URL 即可从/release 到切换/milestone 或/snapshot 版本。库一旦配置 Gradle 依赖就会生效：

````
dependencies {
    compile("org.springframework:spring-context:4.3.25.RELEASE")
    testCompile("org.springframework:spring-test:4.3.25.RELEASE")
}
````

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

2.3.1.5 Ivy 的依赖管理

如果你更喜欢用 Ivy[10]管理依赖，其配置选项也是类似的。

使用 Ivy，首先在 ivysettings.xml 文件的 resolver 中需要配置如下的 Spring 库：

```
<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 选择不同的版本，例如/release、/milestone 或/snapshot。接下来配置具体版本的依赖，例如（在 ivy.xml 中）：

```
<dependency org="org.springframework"
    name="spring-core" rev="4.3.25.RELEASE" conf="compile->runtime"/>
```

2.3.1.6 使用发布的 zip 文件

虽然推荐使用项目管理工具来管理 Spring 依赖，但是开发者仍然可以下载发布的 zip 文件来引入 Spring 框架。

zip 文件一般会发布到 Spring Maven Repository 中（使用 zip 文件主要是方便，在下载这些文件的时候无需配置 Maven 或者其他的项目管理工具）。

zip 包的访问网址为 http://repo.spring.io/release/org/springframework/spring[11]，然后选择需要版本的文件夹。下载文件结尾是 -dist.zip 的 zip 包即可使用，例如，spring-framework-{springversion}-RELEASE-dist.zip。分发的 zip 也包含里程碑[12]和快照[13]的版本。

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，强制性的外部依赖；

b）每个人都喜欢从他们使用的工具中看到输出信息；

c）Spring 集成的很多其他工具都依赖日志功能。

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。这时可能很困难，因为现在有太多日志框架可供选择。

Spring 强制性的日志依赖是 Jakarta Commons Logging API（JCL）。编译 JCL 时，开发者需使 JCL Log 对象对 Spring 框架的扩展类可见。所有版本的 Spring 使用相同的日志库，这对于用户来说很重要：Spring 或应用程序升级时能够向后兼容。这样做是为了使 Spring 的一个模块明确依赖 commons-logging(JCL 的典型实现)，然后其他的所有模块在编译的时候都依赖它。以 Maven 为例，如果你想知道何处依赖了 commons-logging，答案无疑是 Spring 的中心模块：spring-core。

使用 commons-logging 的好处是：无需再添加其他 jar 包就能让应用程序跑起来。它提供了运行

[10] https://ant.apache.org/ivy
[11] http://repo.spring.io/release/org/springframework/spring
[12] http://repo.spring.io/milestone/org/springframework/spring
[13] http://repo.spring.io/snapshot/org/springframework/spring

时发现算法，该算法会在 classpath 路径下寻找其他的日志框架并且使用它认为适合的框架（或者由开发者明确指定）。如果没有其他的日志框架存在，就会从 JDK 中（Java.util.logging 或者简称为 JUL）配置日志。在大多数情况下，能够在控制台查看 Spring 应用程序的工作和日志是很有必要的。

2.3.2.1 使用 Log4j 1.2 或 2.x

注意：目前 Log4j 1.2 版本已经停止更新。同时，Log4j 2.3 版本是兼容 Java 6 的最终版本，而最新发布的 2.x 都必需要配置 Java 7+的 JDK。

出于配置和管理的目的，许多人使用 Log4j[14]作为日志框架。它提供了有效的、完整的功能，事实上 Spring 框架在运行时也是使用 Log4j 框架。Spring 也提供一些配置和初始化 Log4j 的工具，因此某些模块在编译时默认依赖于 Log4j。

为了使 Log4j 工作在默认的 JCL（commons-logging）依赖下，所需要做的就是把 Log4j 放到类路径下，并为它提供配置文件（位于类路径的根目录下的 log4j.properties 或者 log4j.xml）。Maven 用户的依赖声明如下：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.25.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

下面是 log4j.properties 的配置实例，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
```

如果需配置 Log4j 2.x 与 JCL，先将 Log4j 的 jar 包添加到类路径中，并提供配置文件(log4j2.xml,log4j2.properties 或其他支持的配置格式[15])。对于 Maven 用户，所需的最小依赖包是:

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

[14] https://logging.apache.org/log4j
[15] https://logging.apache.org/log4j/2.x/manual/configuration.html

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

如果需要将 SLF4J 委派给 Log4j 作为默认库，还需要以下依赖项:

````
<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 放弃 commons-logging

不幸的是，虽然 commons-logging 提供了运行时日志框架发现算法，确实方便了用户，但也有很多无法解决的问题。如果我们能够时光倒流，现在从新为 Spring 项目配置不同的日志依赖。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 使用到应用程序中。

有两种方法可以关闭 commons-logging:

a) 通过 spring-core 模块排除依赖（因为它是唯一的直接依赖于 commons-logging 的模块）。

b)依赖特殊的 commons-logging，使用一个空的 jar（更多的细节可以在 SLF4J FAQ[16]中找到）替换掉实际库。

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

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.25.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，要修复它就必须提供一个新的日志框架。下一节我们将展示如何提供另一种 JCL 实现——使用 SLF4J 的实现。

2.3.2.3 将 Log4j 或 Logback 用作 SLF4J

The Simple Logging Facade for Java (SLF4J[17])常用作 Spring 的日志组件，使用其他库实现了常用的 API。通常选择 Logback[18]，它是 SLF4J API 的本地实现。

SLF4J 能够绑定很多常见的日志框架，包括 JCL，它还做了反向工作：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，需要使用 SLF4J-JCL 桥接替换掉 commons-logging 的依赖。一旦如此，Spring 调用日志就会调用 SLF4J API，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。

常见的选择就是桥接 Spring 和 SLF4J，使用显式的方式将 SLF4J 绑定到 Log4J。开发者需要添加几个依赖（排除现有的 commons-logging）：JCL 桥接，绑定到 Log4J 的 SLF4j 以及 Log4J 实现自身。使用 Maven 可以做如下配置：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.25.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
```

[16] http://slf4j.org/faq.html#excludingJCL
[17] http://www.slf4j.org/
[18] https://logback.qos.ch/

```
    </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>1.7.21</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：使用更少的步骤和更少的依赖，那就是直接绑定 Logback。因为 Logback 直接实现了 SLF4J，因此只需要配置依赖两个库（jcl-over-slf4j 和 logback）即可：

```
<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)

如果类路径上未检测到 Log4j，则日志框架会默认使用 java.util.logging。此时，无需设置特殊的依赖项：这适合与使用 Spring 作为独立的应用程序(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 上的 commons-loggin

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（WAS）。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，使你能够设置本地包含的 log 提供程序,例如 Log4j 或 Logback,在你的应用程序中。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，建议在"父级在后"模型中部署 Spring 应用程序，因为它默认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 4.x 中的新特性

本章介绍 Spring 框架 4.3 中引入的新功能和改进。如果你对更详细的信息感兴趣，请查看 4.3 开发过程的问题集：Issue Tracker tickets[19]。

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 增加了注解(annotation-driven)的配置支持;Spring 3.0 增加了对 Java 5+版本的支持和基于 Java 的@Configuration 模型。

Spring 4.0 是最新的主要版本【当然现在最新版本是 5.x.x】，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[20]上查看升级 Spring 4.0[21]的迁移指南。

3.1 更好的入门体验

新的 spring.io[22]网站提供了整个系列的"入门指南[23]"，帮助新手学习 Spring。你可以在本文档的第一章 “Getting Started with Spring” 阅读更多的入门指南。新网站还提供了 Spring 其他项目的全面概述。

如果你是 Maven 用户,你可能会对 BOM 这个有用的 POM 文件感兴趣[24],这个文件已经随每个 Spring 的发布版一起发布。

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都已经从 Spring 4.0 中移除。如果你从之前的发布版升级 Spring，你需要保证你的应用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[25]。

注意，所有可选的第三方依赖都已经升级到 2010/2011 年发布的最新版本(例如 Spring 4 通常只支持 2010 年最新或者现在的最新版本)：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）。

---

[19] 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

[20] https://github.com/spring-projects/spring-framework/wiki

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

[22] https://spring.io/

[23] https://spring.io/guides

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

[25] 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 的某些特性。你可以在 Spring 的回调接口中使用 lambda 表达式和引用方法。支持 java.time (JSR-310[26])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，强烈建议新创建的 Spring 4 项目使用 Java 7 或 Java 8 版本 JDK。

注意【4.3.25 新增内容】：

截至 2017 年底，JDK 6 正在被逐步淘汰，即使 Spring 仍然支持 JDK 6。Oracle 和 IBM 将在 2018 年终止对 JDK 6 的所有商业支持。虽然 Spring 将在整个 4.3.x 系列中保持对 JDK 6 的兼容性，但建议升级到 JDK 7 或更高版本，以便获得更多支持：特别是对 JDK 6 特定的错误修复或其他问题，升级到 JDK 7 可以解决该问题。

3.4 Java EE 6 和 7

Java EE 6+版本是 Spring 4 要求的最低版本，这对 JPA 2.0 和 Servlet 3.0 规范有着重要的意义。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。但强烈建议在 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 服务器时规范同样适用。

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。这两者都不支持 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。在概念上，配置类似于使用 XML 的 bean 定义，但是支持更简洁的语法。使用 Groovy 还可以轻松地将 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) {
```
````

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

```
        dataSource = dataSource
    }
    myService(MyService) {
        nestedBean = { AnotherBean bean ->
            dataSource = dataSource
        }
    }
}
````

有关更多信息，请参阅 GroovyBeanDefinitionReader 的 javadocs[27].

3.6 核心容器的改进

核心容器的常规改进包括：

a)Spring 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，可以便捷地插入特定的实现：@Autowired Repository<Customer> customerRepository。

b)如果使用 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 lists 和 arrays 时，可以对当中的 Beans 进行排序（第 7.9.2 节）。支持@Order 注解和 Ordered 接口两种方式。 @Lazy 注解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[28]提供的。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器可言了。

f)框架现在支持管理时区。例如 LocaleContext。

3.7 常规 Web 改进

现在的 Spring 仍然可以部署到 Servlet 2.5 服务器，但是 Spring 4.0 主要用在 Servlet 3.0+ 环境。如果使用 Spring MVC 测试框架（第 15.6 节），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 Web 模块中：

a)可以在 Spring MVC 应用中使用新的@RestController 注解（第 22.3.3.6 节），不再需要给 @RequestMapping 的方法都添加@ResponseBody 注解。

b)添加了 AsyncRestTemplate 支持类，当开发 REST 客户端时，提供了非阻塞异步的支持（第 28.10.3 节）。

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

---

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

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

3.8 WebSocket，SockJS 和 STOMP 消息

新的 spring-websocket 模块提供了全面基于 WebSocket，能够在 Web 应用的客户端和服务器之间双向通信的支持。它与 Java WebSocket API JSR-356[29]兼容，此外还提供了当浏览器不支持 WebSocket 协议时(如 Internet Explorer<10)，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程模型路由和处理，给 WebSocket 客户端发送 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[30]，例如 Message、MessageChannel、MessageHandler 等等，这些都是消息传递的应用基础。

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 测试的改进

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），用于创建自定义的组合注解【composed annotations】，同时减少了测试套件配置。

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

c)spring-core 模块提供了新的 SocketUtils 类。这个类可以扫描本地主机的空闲的 TCP 和 UDP 服务端口。这个功能并不是专门用于测试的，但是在使用 Socket 编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 的。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“测试的改进”

4.1 JMS 改进

Spring 4.1 引入更简单的基础架构,使用@JmsListener[31]注解 bean 方法来注册 JMS 监听端点(第

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

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

[31] https://docs.spring.io/spring-framework/docs/4.3.25.RELEASE/javadoc-api/org/springframework/jms/annotation/JmsListener.html

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[32]， JmsListenerContainerFactory)来配置架构。也可以使用 JmsListenerConfigurer[33]注解来注册监听端点。

Spring 4.1 还调整了 JMS 的支持，优化了 Spring 4.0 中引入的 spring-messaging 抽象，包括：

a)消息监听端点提供了更灵活的名字，更直观的标准消息注解，例如@Payload、@Header、@Headers 和@SendTo 注解。另外，方法参数也可以使用标准的消息代替 javax.jms.Message。

b)新的可用 JmsMessageOperations[34]接口，允许使用 JmsTemplate 操作 Message 抽象。

另外，Spring 4.1 还做了其他各种各样的改进：

a)JmsTemplate 中的同步请求-回答操作支持。

b)每个<jms:listener/>元素都可指定监听器的优先权。

c)消息监听器容器恢复选项可以使用 BackOff[35]实现进行配置。

d)JMS 2.0 提供共享消费者支持。

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抽象；使用标准注解无需作任何更改。

Spring 4.1 也大大改进了自行实现的缓存抽象：

a)缓存可以在运行时使用 CacheResolver 解析（第 36.3.1.4 节）。因此使用 value 参数定义的缓存名称不再是强制性的。

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)新型的类级别注解：@CacheConfig（第 36.3.5 节）可以在类级别上共享常用配置，而无需开启任何缓存操作。

d)使用 CacheErrorHandler 更好的处理缓存方法异常。

Spring 4.1 在 Cache 接口添加新的 putIfAbsent 方法，并做了重大改进。

4.3 Web 改进

a)现有的基于 ResourceHttpRequestHandler 的资源处理已经扩展成新的抽象 ResourceResolver，ResourceTransformer 和 ResourceUrlProvider。一些内置的实现提供了版本控制资源的 URL（提供更高效的 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，即底层服务（或者调用

---

32 https://docs.spring.io/spring-framework/docs/4.3.25.RELEASE/javadoc-api/org/springframework/jms/annotation/EnableJms.html

33 https://docs.spring.io/spring-framework/docs/4.3.25.RELEASE/javadoc-api/org/springframework/jms/annotation/JmsListenerConfigurer.html

34 https://docs.spring.io/spring-framework/docs/4.3.25.RELEASE/javadoc-api/org/springframework/jms/core/JmsMessageOperations.html

35 https://docs.spring.io/spring-framework/docs/4.3.25.RELEASE/javadoc-api/org/springframework/util/backoff/BackOff.html

AsyncRestTemplate）返回的 ListenableFuture。

d)@ModelAttribute 方法现在按照相互依存关系的顺序调用。见 SPR-6299[36]。

e)Jackson 的@JsonView 直接用于@ResponseBody 和 ResponseEntity 控制器方法，能够对相同的 POJO 的不同细节（如摘要与细节页）进行序列化。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”。

g)在响应被写入之前拦截@ResponseBody 和 ResponseEntity 方法，能够控制方法返回的生命周期。可以使用@ControllerAdvicebean 来实现 ResponseBodyAdvice 接口，为@JsonView 和 JSONP 提供内置支持。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—一个比 Jackson 更轻量级的封装；已经用于 Spring Android 模块中【据反馈 gson 的 bug 不是一般的多】。

b)Google Protocol Buffers—高效和优秀的企业级内部跨业务数据通信协议，但也可以用于浏览器 JSON 和 XML 的扩展。

c)基于 XML 的 Jackson 序列化，现在通过 jackson-dataformat-xml 扩展得到支持。如果 classpath 下包含 jackson-dataformat-xml[37]文件，则默认情况下会使用@EnableWebMvc 或 <mvc:annotation-driven/>进行注解，而不是 JAXB2。

g)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。默认名称会分配给每一个 @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@RequestMapping 命名。在 Spring JSP 标签库中新的 mvcUrl 功能找到使这个简单的 JSP 页面并使用。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 为控制器向服务器端展示响应，例如，ResponseEntity.ok()。

i)RequestEntity 是一种新型的，提供了 builder 风格的 API 为客户端的 REST 响应展示 HTTP 请求。

MVC 的 Java 配置和 XML 命名空间：

a)视图解析器现在可以支持内容协商配置，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。更多用例参看这里[38]。

c)现在可以自定义内置路径匹配。参见第 22.16.11 节 “路径匹配”。

j)Groovy 的标记模板[39]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 以及包下的其他类。

b)新的应用上下文事件类：SessionSubscribeEvent 和 SessionUnsubscribeEvent，分别用于 STOMP 客户端的订阅和取消订阅。

c)新的"websocket"作用域。查看第 26.4.15 节 “WebSocket 的作用域”。

d)@SendToUser 注解可以只用在会话上，而不一定需要用户授权。

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

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

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

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

e)@MessageMapping 方法可以使用“.”号来代替 “/”作为目录分隔符。查看 SPR-11660[40]。

f)STOMP/WebSocket 监听消息的收集和日志记录。更多查看“运行时监控”（ 第 26.4.17 节）。

g)优化和改善了日志在 DEBUG 等级下的可读性和简洁性。

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。更多查看 MessageHeaderAccessor 的 Java 文档。

i)WebSocket 会话建立之后的 60s 没有任何活动的话将会关闭 STOMP/WebSocket 连接。更多查看 SPR-11884[41]。

4.5 测试的改进

可以在 ApplicationContext 中配置 Groovy 脚本，并使用 TestContext 框架中加载后进行集成测试。

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，可以编程式地开始和结束测试管理事务。

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 注解声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。

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

现在默认的 TestExecutionListeners 可以自动被发现。

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

现在自定义 TestExecutionListeners 可以通过默认的监听器自动进行合并。

详见第 15.5.3.2 节“合并 TestExecutionListeners”。

TestContext 框架中测试事务的文档已经通过更多解释和案例进行了完善。

详见第 15.5.7 节“事务管理”。

a)优化了 MockServletContext、MockHttpServletRequest 以及其他 Servlet 接口。

b)AssertThrows 重构后，支持使用 Throwable 代替 Exception。

c)Spring MVC 测试中，使用 JSONPath 选项返回的 JSON 格式可以使用 JSONAssert[42]进行断言，这与之前的 XML 和 XMLUnit 是类似的。

d)MockMvcBuilder 方法【原文为 recipes】现在可以在 MockMvcConfigure 的帮组下进行创建。这个方法的加入让 SpringSecurity 设置变得简单，适用于任何第三方的普通封装或仅用在本项目。

e)MockRestServiceServer 现在支持 AsyncRestTemplate 用作客户端测试。

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 “测试的改进”

[40] https://jira.spring.io/browse/SPR-11660
[41] https://jira.spring.io/browse/SPR-11884
[42] https://github.com/skyscreamer/JSONassert

5.1 核心容器改进

a)诸如@bean 的注释，用起来与 Java 8 默认方法一样，可以在方法上与组合配置类共用。

b)配置类可以声明@import 作为常规组件类，可共用引入的配置类和组件类。

c)配置类可以声明@Order 值，用来处理相应的处理顺序(例如重写 bean 的名字)，通过类路径扫描的检测也可以使用此注解。

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 的延迟初始化代理。

e)现在的应用程序事件基础架构提供了基于注解的模型[43]以及发布任意事件的能力。

a)任何注解为 bean 的公共方法都可以使用@EventListener 注解用于事件消费。

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

f)Spring 4.2 引入了类优先【first-class】概念支持声明和查找注释属性的别名。新的@AliasFor 注解可在单个注解上声明一对带别名的属性，还可以将自定义的组合注解的属性值声明为元注解的属性值。

a)下面的注解已添加@AliasFor 的支持，可提供更有意义的 value 值别名:@Cacheable, @CacheEvict, @CachePut, @ComponentScan, @ComponentScan.Filter, @ImportResource, @Scope, @ManagedResource, @Header, @Payload, @SendToUser, @ActiveProfiles, @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】的元注解覆盖属性，现在可以使用@AliasFor 进行细粒度的控制，可准确将属性值用于具体的层次结构上。事实上，现在可以给元注解的 value 属性声明别名。

e)例如，开发组合注解用于自定义的属性的覆盖。

```
@ContextConfiguration
public @interface MyTestConfig {
```

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

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

        // ...
    }
````

f)见 Spring 注解编程模式[44]。

g)例如，开发组合注解用于自定义的属性的覆盖，改进了 Spring 用于寻找元注解的搜索算法。例如，局部声明组合注解现在流行继承注解。

h)从元注解覆盖属性的组合注解，可以被接口发现，就像用在类、标准方法、构造函数和字段上的 abstract、bridge 一样。

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。

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

l)DefaultFormattingConversionService 提供开箱即用的,支持 JSR-354 的 Money&Currency 类型(前提是'javax.money'API 已经添加到类路径上):它们分别命名为 MonetaryAmount 和 CurrencyUnit。支持使用@NumberFormat 注解。

m)@NumberFormat 现在作为元注解使用。

n)JavaMailSenderImpl 中新的 testConnection()方法可用于检查服务器连接。

o)ScheduledTaskRegistrar 可用于公开调度的任务。

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

q)引入 StandardScriptFactory 作为脚本化的 bean（符合 JSR-223 规范），通过 XML 中的 lang:std 元素公开。同时支持 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 JPA 供应商(自动适配)如同原生的 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“测试的改进”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 可以在每个监听器容器中配置。

c)@SendTo 的值可以使用 SpEL 表达式。

d)响应目的地可以通过 JmsResponse 在运行时计算[45]。

---

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

e)@JmsListener 可以使用重复的注解，可以在多个 JMS 容器上声明多个相同的方法(如果你还没有用上 Java 8，请使用新引入的@JmsListeners 注解)。

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(MVC Java 配置和 XML 命名空间)和局部(如@CrossOrigin)配置。见第 27 章：CORS 支持。

c)HTTP 缓存改进：

a) 新的 CacheControl 构建器；添加了 ResponseEntity,WebContentGenerator,ResourceHttpRequestHandler 等方法。

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

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 方法可用于自定义 DispatcherServlet 实例。

g)HandlerMethod 可作为@ExceptionHandler 方法的参数，特别是@ControllerAdvice 组件。

h)java.util.concurrent.CompletableFuture 可作为@Controller 方法的返回值类型。

i)在 HttpHeaders 中，字节范围（Byte-range）可以用于对静态资源的请求。

j)@ResponseStatus 能够发现嵌套异常。

k)在 RestTemplate 中进行 UriTemplateHandler 扩展。

a)DefaultUriTemplateHandler 公开 baseUrl 属性和路径段的编码选项。

b) 扩展端点能够使用到任何的 URI 模板库中。

l)OkHTTP[46]已经集成到 RestTemplate 中。

m)MvcUriComponentsBuilder 可自定义 baseUrl 选择方法。

n)序列化/反序列化异常消息调整为 WARN 级别。

o)默认的 JSON 前缀从"{} && "改为更安全的")]}', "。

p)新的 RequestBodyAdvice 扩展和内置的实现支持 Jackson 序列号，并且用到@RequestBody 的@JsonView 中。

q)使用 GSON 或 Jackson 2.6+，用于提高返回类型参数化的序列化，例如 List<Foo>。

r)引入 ScriptTemplateView 作为 JSR-223 脚本化 Web 视图的基础机制，引入 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)在服务器集群中共享展示信息(见代理中继配置选项)。

b)在服务器集群中解决用户目的地的操作(见代理中继配置选项)。

c)StompSubProtocolErrorHandler 扩展端点用于自定义和控制 STOMP ERROR 帧发送给用户。

d)通过@ControllerAdvice 组件引入了全局的@MessageExceptionHandler 方法。

e)用于 SimpleBrokerMessageHandler 订阅的心跳和 SpEL 表达式（“selector”头）。

---

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

46 https://square.github.io/okhttp/

f)STOMP 客户端使用 TCP 和 WebSocket；见第 26.4.14 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 支持从@MessageMapping 和@SubscribeMapping 方法返回类型的值。

i)用于 XML 负载的 MarshallingMessageConverter 方法。

5.6 测试的改进

a)基于 JUnit 的集成测试现在可以用于执行 JUnit 规则，而不仅仅用于 SpringJUnit4ClassRunner。这允许 Spring 集成 JUnit 的 Parameterized 或第三方库来进行测试。

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 的 WebDriver，允许在未部署到 Servlet 容器的情况下进行基于页面的 Web 应用测试。

参看第 15.6.2 节“HtmlUnit 集成”。

c)AopTestUtils 是新的测试工具，它允许开发者在一个或多个 Spring 代理中获取潜在的目标对象引用。

参看第 14.2.1 节“ 通用测试功能”。

d)ReflectionTestUtils 现在支持 setting 和 getting static 字段，包括常量。

e)bean 定义支持文件的原始顺序，通过@ActiveProfiles 进行声明，例如 Spring 的 ConfigFileApplicationListener 可以基于活动文件的名称加载配置文件。

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

g)ApplicationContext--例如，如果一些繁杂的(即，有待确定的)测试在大型测试套件的 ApplicationContext 原始配置中已经被去掉。

h)新注解@Commit 现在可以直接取代@Rollback(false)注解。

i)@Rollback 用来配置类级别的默认回滚语义。

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

j)@Sql 现在可以通过新的 statements 属性来支持内联 SQL 语句的执行。

k)以前 ContextCache 用于缓存测试中的 ApplicationContext，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 框架中，MvcResult 详情可以被日志记录（需设置成 DEBUG 级别）或者写入自定义的 OutputStream 或 Writer 中。详见 log()、print(OutputStream)和 MockMvcResultHandlers 的 print(Writer)方法。

o)JDBC XML 名称空间支持新的<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——通过 SpEL 表达式或活动的 bean 定义配置文件中的占位符。

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 “测试的改进”

i)第 6.1 节 “新库和访问的支持”

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用于改进编程。

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,则无需要指定@Autowired 注解。

d)@Configuration 类支持构造函数注入。

e)@EventListener 可以使用 SpEL 表达式引入符合条件的 bean（例如@beanName.method()）。

f)组合注解可以使用包含元注解数组属性的组件元素来进行覆盖。例如，@RequestMapping 的 String[] path 可以在组合注解中使用 String path 进行覆盖。

g)@Scheduled 和@Schedules 现在可以用作元注解，可以通过属性覆盖来创建自定义组合注解。

h)@Scheduled 注解支持任意作用域的 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，并能用于脚本。

6.3 缓存优化

Spring 4.3 允许在给定的 key 并发调用时实现需要的同步，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 的新型 sync 属性来启用此功能。此功能的引入修改了 Cache 接口，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)@EventListener 可以使用 SpEL 表达式引入符合条件的 bean（例如@beanName.method()）。

b)ConcurrentMapCacheManager 和 ConcurrentMapCache 现在通过新的 storeByValue 属性来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 现在可以用作元注解，能够通过属性覆盖来创建自定义组合注解。

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 现在可以用作元注解，能够通过属性覆盖来创建自定义组合注解。

6.5 Web 优化

a)内部提供了 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 用于替代已有的@RequestMapping 注解。

详见第 22.3.2.1 节 “组合@RequestMapping 注解”。

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 语义的组合。

e)@ResponseStatus 现在支持用于类级别，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 可以声明为可选的注解(用于@ModelAttribute 方法)。

k)错误和自定义的异常都将被统一到 MVC 异常处理器中处理。

l)对 HTTP 消息转换编码进行了统一处理，包括对多部分文本内容默认使用 UTF-8 编码。

m)配置 ContentNegotiationManager 用于媒体类型的计算，并处理静态资源。

n)RestTemplate 和 AsyncRestTemplate 支持通过 DefaultUriTemplateHandler 来实现严格的 URI 请求内容编码。

o)AsyncRestTemplate 支持拦截请求。

6.6 WebSocket 消息优化

@SendTo 和@SendToUser 现在可以在类级指定共同的目的地。

6.7 测试的改进

a)为了支持 Spring TestContext 框架，现在需要 JUnit 4.12+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，在基于 Java 8 的接口上使用测试接口。

d)声明为 null 的@ContextConfiguration 在检测到默认的 XML 文件、Groovy 脚本或 @Configuration 类型将会被完全忽略。

e)@Transactional 测试方法不再需要 public(例如在 TestNG 和 JUnit 5 中)关键字。

f)@BeforeTransaction 和@AfterTransaction 不再需要 public，并且能在基于 Java 8 接口的默认方法上声明。

g)Spring TestContext 框架的 ApplicationContext 的缓存默最大值为 32，这符合最小空间实现最大性能的策略。最大的大小可以通过设置 JVM 系统属性 --spring.test.context.cache.maxSize--或 Spring 配置。

h)ContextCustomizer API 用于自定义测试 ApplicationContext，测试点是在 bean 定义加载到上下文之后但在上下文被刷新前。定制工具可以在全局范围内由第三方进行注册，而无需要对 ContextLoader 进行自定义现实。

i)@Sql 和@SqlGroup 现在可以用作元注解，能够通过属性覆盖来创建自定义组合注解。

j)ReflectionTestUtils 在 set 或 get 一个字段时，会自动拆卸代理。

k)服务器端的 Spring MVC 测试支持持有多个值的响应头。

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 的 spring-core.jar 包中集成了 ASM 5.1、CGLIB 3.2.4 和 Objenesis 2.4 功能。

第三部分. 核心技术

这部分的参考文档将完整介绍 Spring 框架中所有最最最重要的技术。

其中最重要的部分就是 Spring 框架中的控制反转（IoC)容器。Spring 框架的 IoC 功能是实现在 Spring 面向切面编程（AOP）技术之上的。Spring 框架自行实现 一套 AOP 框架，这套框架从概念上非常容易理解，而且成功解决了 Java 企业级应用中使用 AOP 解决的 80%核心需求。

Spring 也提供与 AspectJ 的集成（从功能上来看，它是最丰富的，而且是本领域最成熟的 AOP 实现）。

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

7. IoC 容器

由于与合作平台版权的原因，后面的章节将在年底合同到期后全部公布。