第一部分：Spring框架概述

Part I. Overview of Spring Framework

The Spring Framework is a lightweight solution and a potential one-stop-shop for building your enterprise-ready applications. However, Spring is modular, allowing you to use only those parts that you need, without having to bring in the rest. You can use the IoC container, with any web framework on top, but you can also use only the Hibernate integration code or the JDBC abstraction layer. The Spring Framework supports declarative transaction management, remote access to your logic through RMI or web services, and various options for persisting your data. It offers a full-featured MVC framework, and enables you to integrate AOP transparently into your software.
Spring is designed to be non-intrusive, meaning that your domain logic code generally has no dependencies on the framework itself. In your integration layer (such as the data access layer), some dependencies on the data access technology and the Spring libraries will exist. However, it should be easy to isolate these dependencies from the rest of your code base.
This document is a reference guide to Spring Framework features. If you have any requests, comments, or questions on this document, please post them on the user mailing list. Questions on the Framework itself should be asked on StackOverflow (see https://spring.io/questions).

翻译：Spring框架是一个轻量级的解决方案，是一个一站式服务解决框架，用于构建企业服务的应用程序。但是，Spring是模块化的，允许您仅使用所需的那些部分，而不必引入其余部分。您可以将IoC容器与顶部的任何Web框架一起使用，但也可以仅使用Hibernate集成代码或JDBC抽象层。 Spring框架支持声明式事务管理，通过RMI或Web服务对逻辑的远程访问以及用于持久化数据的各种选项。它提供了功能全面的MVC框架，并使您能够将AOP透明地集成到软件中。

Spring被设计为非侵入式的，这意味着您的域逻辑代码通常不依赖于框架本身。在您的集成层（例如数据访问层）中，将存在对数据访问技术和Spring库的某些依赖关系。但是，将这些依赖项与其余代码库隔离起来应该很容易。

本文档是Spring Framework功能的参考指南。如果您对此文档有任何要求，评论或问题，请将其张贴在用户邮件列表中。有关框架本身的问题应在StackOverflow上提出（请参阅https://spring.io/questions）。

1. Getting Started with Spring

This reference guide provides detailed information about the Spring Framework. It provides comprehensive documentation for all features, as well as some background about the underlying concepts (such as *"Dependency Injection"*) that Spring has embraced.
If you are just getting started with Spring, you may want to begin using the Spring Framework by creating a Spring Boot based application. Spring Boot provides a quick (and opinionated) way to create a production-ready Spring based application. It is based on the Spring Framework, favors convention over configuration, and is designed to get you up and running as quickly as possible.
You can use start.spring.io to generate a basic project or follow one of the "Getting Started" guides like the Getting Started Building a RESTful Web Service one. As well as being easier to digest, these guides are very *task focused*, and most of them are based on Spring Boot. They also cover other projects from the Spring portfolio that you might want to consider when solving a particular problem.
翻译：1. Spring入门
该参考指南提供了有关Spring框架的详细信息。它提供了所有功能的全面文档，以及Spring所接受的有关基础概念（例如“依赖注入”）的一些背景知识。

如果您刚刚开始使用Spring，则可能需要通过创建基于Spring Boot的应用程序来开始使用Spring Framework。 Spring Boot提供了一种快速（且自以为是）的方法来创建可用于生产的基于Spring的应用程序。它基于Spring框架，更倾向于约定而不是配置，并且旨在使您尽快启动并运行。

您可以使用start.spring.io生成一个基本项目，也可以遵循“入门”指南之一，例如“构建RESTful Web服务入门”。这些指南不仅易于理解，而且非常注重任务，并且大多数基于Spring Boot。它们还涵盖了Spring产品组合中的其他项目，您在解决特定问题时可能要考虑这些项目。

2. Introduction to the Spring Framework

The Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from "plain old Java objects" (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.
Examples of how you, as an application developer, can benefit from the Spring platform:

- Make a Java method execute in a database transaction without having to deal with transaction APIs.
- Make a local Java method an HTTP endpoint without having to deal with the Servlet API.
- Make a local Java method a message handler without having to deal with the JMS API.
- Make a local Java method a management operation without having to deal with the JMX API.

2. Spring框架简介
Spring框架是一个Java平台，为开发Java应用程序提供全面的基础架构支持。 Spring处理基础结构，因此您可以专注于应用程序。

Spring使您能够从“普通的Java对象”（POJO）构建应用程序，并将企业服务非侵入性地应用于POJO。 此功能适用于Java SE编程模型以及全部和部分Java EE。

作为应用程序开发人员，如何从Spring平台中受益的示例：

使Java方法在数据库事务中执行，而不必处理事务API。
使本地Java方法成为HTTP端点，而不必处理Servlet API。
使本地Java方法成为消息处理程序，而不必处理JMS API。
使本地Java方法成为管理操作，而不必处理JMX API。

2.1 Dependency Injection and Inversion of Control

A Java application — a loose term that runs the gamut from constrained, embedded applications to n-tier, server-side enterprise applications — typically consists of objects that collaborate to form the application proper. Thus the objects in an application have *dependencies* on each other.
Although the Java platform provides a wealth of application development functionality, it lacks the means to organize the basic building blocks into a coherent whole, leaving that task to architects and developers. Although you can use design patterns such as *Factory*, *Abstract Factory*, *Builder*, *Decorator*, and *Service Locator* to compose the various classes and object instances that make up an application, these patterns are simply that: best practices given a name, with a description of what the pattern does, where to apply it, the problems it addresses, and so forth. Patterns are formalized best practices that *you must implement yourself* in your application.
The Spring Framework *Inversion of Control* (IoC) component addresses this concern by providing a formalized means of composing disparate components into a fully working application ready for use. The Spring Framework codifies formalized design patterns as first-class objects that you can integrate into your own application(s). Numerous organizations and institutions use the Spring Framework in this manner to engineer robust, *maintainable* applications.

Background
"*The question is, what aspect of control are [they] inverting?*" Martin Fowler posed this question about Inversion of Control (IoC) on his site in 2004. Fowler suggested renaming the principle to make it more self-explanatory and came up with *Dependency Injection*.

翻译：2.1依赖注入和控制反转
Java应用程序（一种宽松的术语，其范围从受约束的嵌入式应用程序到n层服务器端企业应用程序），通常由协作形成适当应用程序的对象组成。因此，应用程序中的对象相互依赖。

尽管Java平台提供了丰富的应用程序开发功能，但是它缺乏将基本构建模块组织成一个连贯的整体的方法，而将任务留给了架构师和开发人员。尽管您可以使用诸如Factory，Abstract Factory，Builder，Decorator和Service Locator之类的设计模式来组成组成应用程序的各种类和对象实例，但是这些模式只是：给定最佳实践的名称，并描述模式的作用，在哪里应用，解决的问题等等。模式是形式化的最佳实践，您必须在应用程序中实现自己。

Spring框架控制反转（IoC）组件通过提供一种形式化的方法来将不同的组件组成一个可以正常使用的应用程序，从而解决了这一问题。 Spring框架将形式化的设计模式编码为一流的对象，您可以将其集成到自己的应用程序中。许多组织和机构都以这种方式使用Spring Framework来设计健壮，可维护的应用程序。

背景

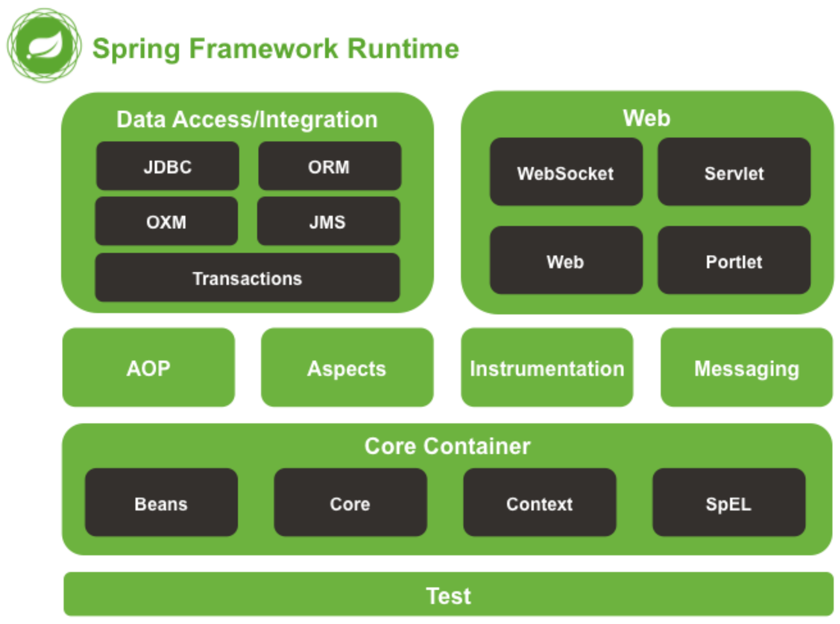
“问题是，（他们）控制的哪个方面是反向的？”马丁·福勒（Martin Fowler）于2004年在他的网站上提出了有关控制反转（IoC）的

问题。福勒建议重命名该原理，使其更加不言自明，并提出了依赖注入。

2.2 Framework Modules

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test, as shown in the following diagram.

Figure 2.1. Overview of the Spring Framework

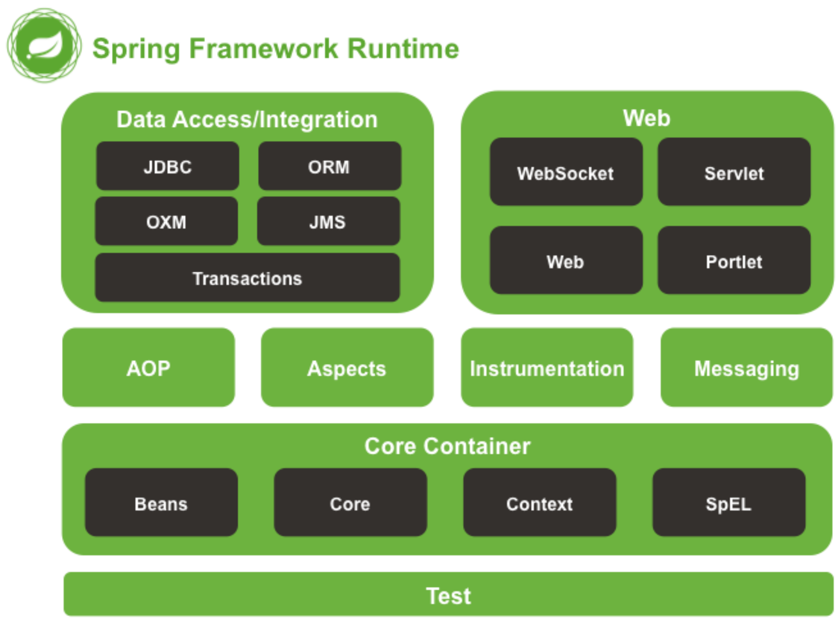


The following sections list the available modules for each feature along with their artifact names and the topics they cover. Artifact names correlate to *artifact IDs* used in Dependency Management tools.

2.2框架模块

Spring框架包含组织为约20个模块的功能。 这些模块分为核心容器，数据访问/集成，Web，AOP（面向方面的编程），检测，消息传递和测试，如下图所示。

图2.1。 Spring框架概述



以下各节列出了每个功能的可用模块，以及其工件名称和它们涵盖的主题。 工件名称与“依赖关系管理”工具中使用的工件ID相关。

2.2.1 Core Container

The *Core Container* consists of the `spring-core`, `spring-beans`, `spring-context`, `spring-context-support`, and `spring-expression` (Spring Expression Language) modules.
The `spring-core` and `spring-beans` modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The `BeanFactory` is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.
The *Context* (`spring-context`) module builds on the solid base provided by the *Core and Beans* modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry. The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event propagation, resource loading, and the transparent creation of contexts by, for example, a Servlet container. The Context module also supports Java EE features such as EJB, JMX, and basic remoting. The `ApplicationContext` interface is the focal point of the Context module. `spring-context-support` provides support for integrating common third-party libraries into a Spring application context for caching (EhCache, Guava, JCache), mailing (JavaMail), scheduling (CommonJ, Quartz) and template engines (FreeMarker, JasperReports, Velocity).
The `spring-expression` module provides a powerful *Expression Language* for querying and manipulating an object graph at runtime. It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification. The language supports setting and getting property values, property assignment, method invocation, accessing the content of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from Spring's IoC container. It also supports list projection and selection as well as common list aggregations.

翻译：2.2.1核心容器
核心容器由spring-core，spring-beans，spring-context，spring-context-support和spring-expression（Spring表达式语言）模块组成。

spring-core和spring-beans模块提供了框架的基本部分，包括IoC和依赖注入功能。 BeanFactory是工厂模式的复杂实现。它消除了对编程单例的需要，并允许您将依赖项的配置和规范与实际程序逻辑脱钩。

上下文（spring-context）模块建立在Core和Beans模块提供的坚实基础上：它是一种以类似于JNDI注册中心的框架样式方式访问对象的方法。 Context模块从Beans模块继承其功能，并增加了对国际化（例如，使用资源束），事件传播，资源加载以及通过Servlet容器透明创建上下文的支持。上下文模块还支持Java EE功能，例如EJB，JMX和基本远程处理。 ApplicationContext接口是Context模块的焦点。 spring-context-support提供了将常见第三方库集成到Spring应用程序上下文中以支持缓存（EhCache，Guava，JCache），邮件（JavaMail），调度（CommonJ，Quartz）和模板引擎（FreeMarker，JasperReports，Velocity）的支持。

spring-expression模块提供了一种功能强大的表达式语言，用于在运行时查询和操作对象图。它是对JSP 2.1规范中指定的统一表达语言（统一EL）的扩展。该语言支持设置和获取属性值，属性分配，方法调用，访问数组，集合和索引器，逻辑和算术运算符，命名变量以及按名称从Spring的IoC容器中检索对象的内容。它还支持列表投影和选择以及常见的列表聚合。

2.2.2 AOP and Instrumentation

The `spring-aop` module provides an *AOP* Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated. Using source-level metadata functionality, you can also incorporate behavioral information into your code, in a manner similar to that of .NET attributes.
The separate `spring-aspects` module provides integration with AspectJ.
The `spring-instrument` module provides class instrumentation support and classloader implementations to be used in certain application servers. The `spring-instrument-tomcat` module contains Spring's instrumentation agent for Tomcat.

翻译：2.2.2 AOP和仪器
spring-aop模块提供了一个符合AOP Alliance要求的面向方面的编程实现，例如，您可以定义方法拦截器和切入点，以干净地解耦实现应分离功能的代码。 使用源级元数据功能，您还可以以类似于.NET属性的方式将行为信息合并到代码中。

单独的spring-aspects模块提供了与AspectJ的集成。

spring-instrument模块提供类仪表支持和类加载器实现，以在某些应用服务器中使用。 spring-instrument-tomcat模块包含用于Tomcat的Spring工具代理。

2.2.3 Messaging

Spring Framework 4 includes a `spring-messaging` module with key abstractions from the *Spring Integration* project such as `Message`, `MessageChannel`, `MessageHandler`, and others to serve as a foundation for messaging-based applications. The module also includes a set of annotations for mapping messages to methods, similar to the Spring MVC annotation based programming model.

翻译：2.2.3消息传递

Spring Framework 4包含一个Spring-messaging模块，该模块具有来自Spring Integration项目的关键抽象，例如Message，MessageChannel，MessageHandler等，它们充当基于消息的应用程序的基础。 该模块还包括一组注释，用于将消息映射到方法，类似于基于Spring MVC注释的编程模型。

2.2.4 Data Access/Integration

The *Data Access/Integration* layer consists of the JDBC, ORM, OXM, JMS, and Transaction modules.
The `spring-jdbc` module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.
The `spring-tx` module supports programmatic and declarative transaction management for classes that implement special interfaces and for *all your POJOs (Plain Old Java Objects)*.
The `spring-orm` module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, and Hibernate. Using the `spring-orm` module you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.
The `spring-oxm` module provides an abstraction layer that supports Object/XML mapping implementations such as JAXB, Castor, XMLBeans, JiBX and XStream.
The `spring-jms` module (Java Messaging Service) contains features for producing and consuming messages. Since Spring Framework 4.1, it provides integration with the `spring-messaging` module.

翻译：2.2.4数据访问/集成

数据访问/集成层由JDBC，ORM，OXM，JMS和事务模块组成。

spring-jdbc模块提供了JDBC抽象层，从而无需进行繁琐的JDBC编码和解析数据库供应商特定的错误代码。

spring-tx模块支持对实现特殊接口的类以及所有POJO（普通Java对象）进行编程和声明式事务管理。

spring-orm模块为流行的对象关系映射API（包括JPA，JDO和Hibernate）提供了集成层。使用spring-orm模块，您可以将所有这些O / R映射框架与Spring提供的所有其他功能结合使用，例如前面提到的简单的声明式事务管理功能。

spring-oxm模块提供了一个抽象层，该抽象层支持对象/ XML映射实现，例如JAXB，Castor，XMLBeans，JiBX和XStream。

spring-jms模块（Java Messaging Service）包含用于生成和使用消息的功能。从Spring Framework 4.1开始，它提供了与spring-messaging模块的集成。

2.2.5 Web

The *Web* layer consists of the `spring-web`, `spring-webmvc`, `spring-websocket`, and `spring-webmvc-portlet` modules.
The `spring-web` module provides basic web-oriented integration features such as multipart file upload functionality and the initialization of the IoC container using Servlet listeners and a web-oriented application context. It also contains an HTTP client and the web-related parts of Spring’s remoting support.
The `spring-webmvc` module (also known as the *Web-Servlet* module) contains Spring’s model-view-controller (*MVC*) and

REST Web Services implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms and integrates with all of the other features of the Spring Framework.
The `spring-webmvc-portlet` module (also known as the *Web-Portlet* module) provides the MVC implementation to be used in a Portlet environment and mirrors the functionality of the Servlet-based `spring-webmvc` module.

翻译：2.2.5网络
Web层由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的集成功能，例如多部分文件上传功能以及使用Servlet侦听器和面向Web的应用程序上下文对IoC容器的初始化。 它还包含HTTP客户端和Spring远程支持的Web相关部分。

spring-webmvc模块（也称为Web-Servlet模块）包含Spring的模型视图控制器（MVC）和针对Web应用程序的REST Web服务实现。 Spring的MVC框架在域模型代码和Web表单之间提供了清晰的分隔，并与Spring框架的所有其他功能集成在一起。

spring-webmvc-portlet模块（也称为Web-Portlet模块）提供要在Portlet环境中使用的MVC实现，并镜像基于Servlet的spring-webmvc模块的功能。

2.2.6 Test

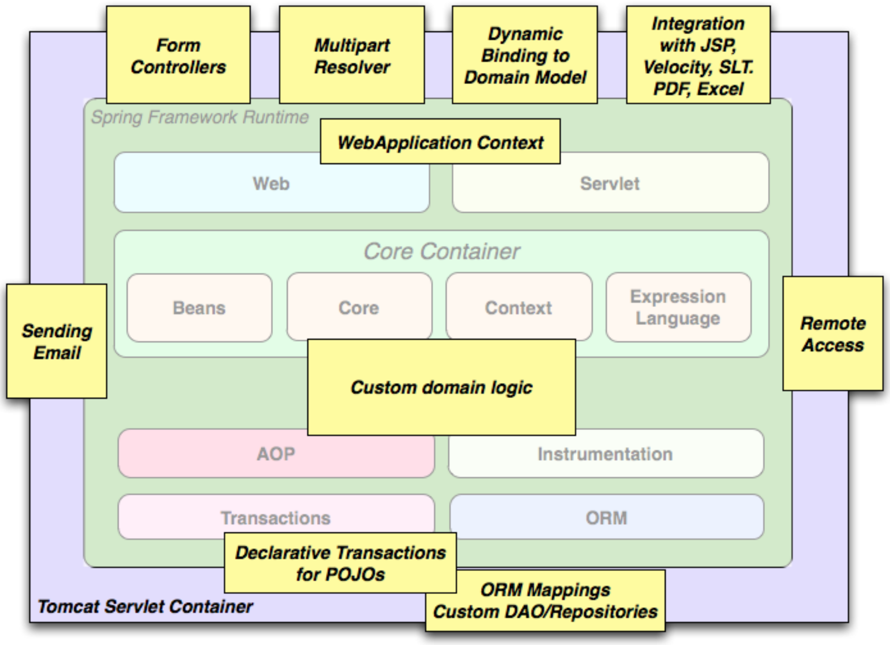
The `spring-test` module supports the unit testing and integration testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring `ApplicationContext`s and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.

翻译：2.2.6测试
spring-test模块支持使用JUnit或TestNG对Spring组件进行单元测试和集成测试。 它提供了Spring ApplicationContexts的一致加载以及这些上下文的缓存。 它还提供了可用于隔离测试代码的模拟对象。

2.3 Usage scenarios

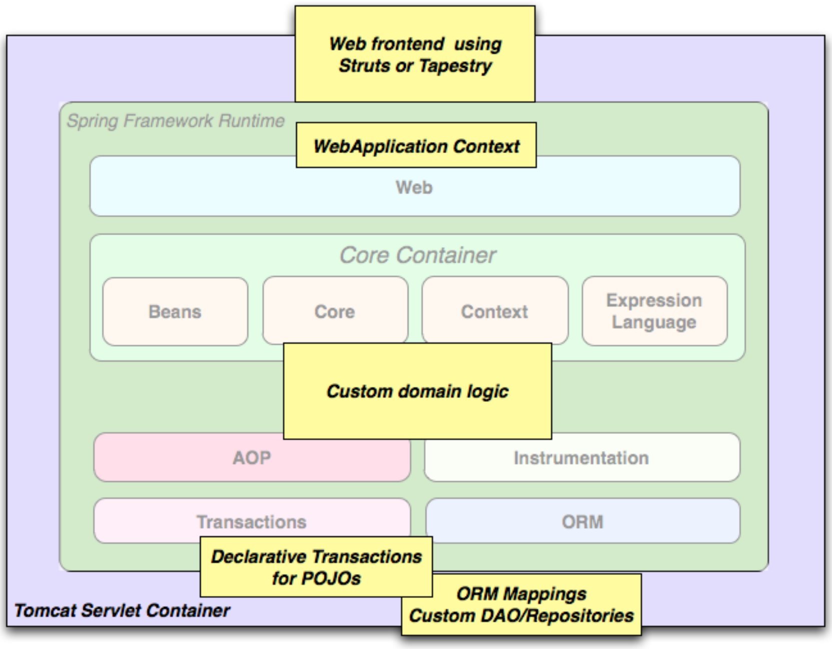
The building blocks described previously make Spring a logical choice in many scenarios, from embedded applications that run on resource-constrained devices to full-fledged enterprise applications that use Spring's transaction management functionality and web framework integration.

Figure 2.2. Typical full-fledged Spring web application



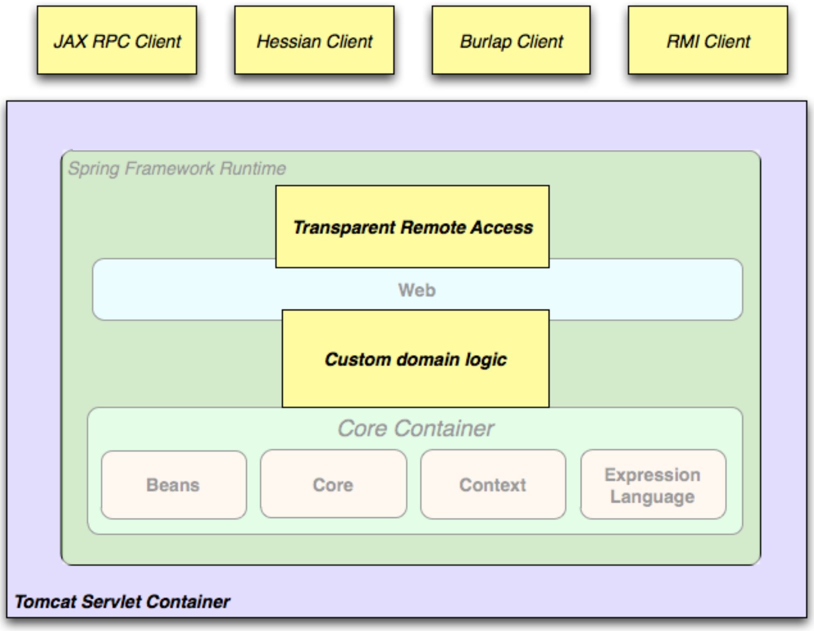
Spring's declarative transaction management features make the web application fully transactional, just as it would be if you used EJB container-managed transactions. All your custom business logic can be implemented with simple POJOs and managed by Spring's IoC container. Additional services include support for sending email and validation that is independent of the web layer, which lets you choose where to execute validation rules. Spring's ORM support is integrated with JPA, Hibernate and JDO; for example, when using Hibernate, you can continue to use your existing mapping files and standard Hibernate `SessionFactory` configuration. Form controllers seamlessly integrate the web-layer with the domain model, removing the need for `ActionForms` or other classes that transform HTTP parameters to values for your domain model.

Figure 2.3. Spring middle-tier using a third-party web framework



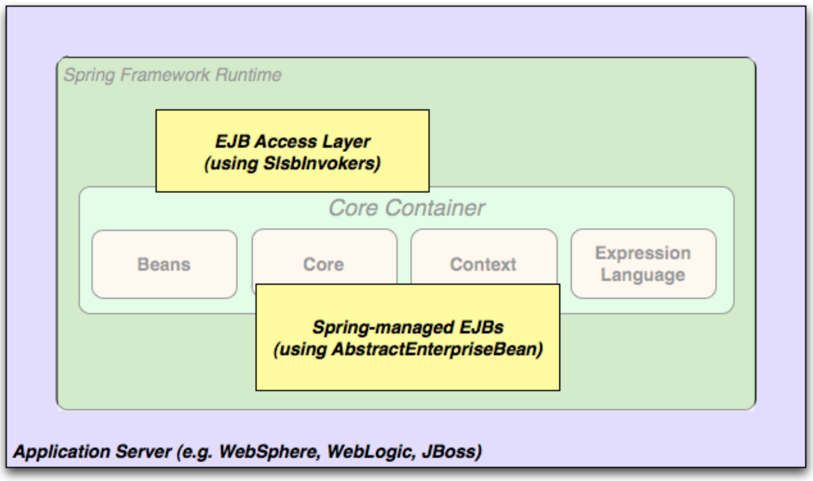
Sometimes circumstances do not allow you to completely switch to a different framework. The Spring Framework does *not* force you to use everything within it; it is not an *all-or-nothing* solution. Existing front-ends built with Struts, Tapestry, JSF or other UI frameworks can be integrated with a Spring-based middle-tier, which allows you to use Spring transaction features. You simply need to wire up your business logic using an `ApplicationContext` and use a `WebApplicationContext` to integrate your web layer.

Figure 2.4. Remoting usage scenario



When you need to access existing code through web services, you can use Spring's `Hessian-`, `Burlap-`, `Rmi-` or `JaxRpcProxyFactory` classes. Enabling remote access to existing applications is not difficult.

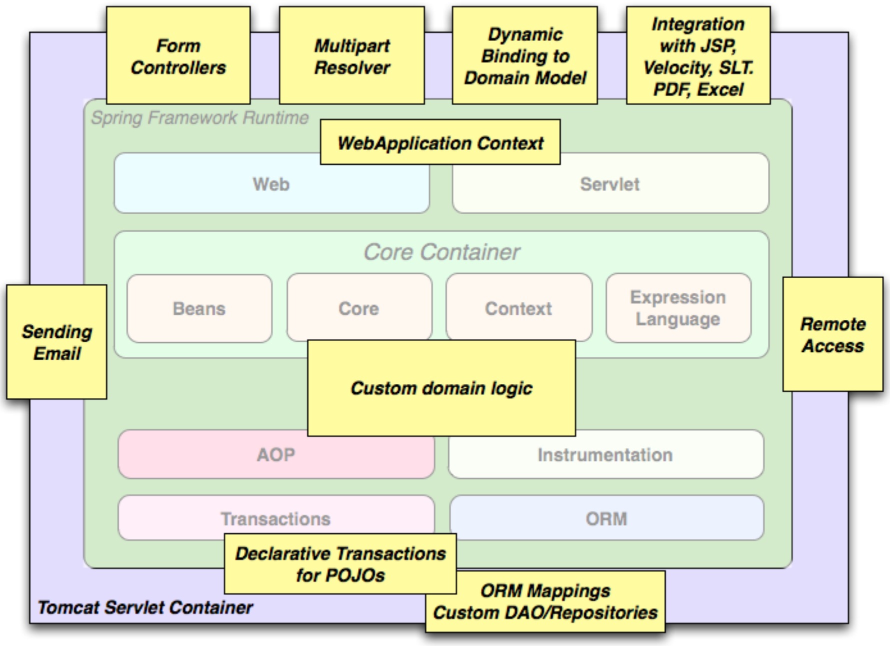
Figure 2.5. EJBs - Wrapping existing POJOs


Spring Framework Runtime
EJB Access Layer (using SlsbInvokers)
Core Container
Beans
Core
Context
Expression Language
Spring-managed EJBs (using AbstractEnterpriseBean)
Application Server (e.g. WebSphere, WebLogic, JBoss)

The Spring Framework also provides an access and abstraction layer for Enterprise JavaBeans, enabling you to reuse your existing POJOs and wrap them in stateless session beans for use in scalable, fail-safe web applications that might need declarative security.

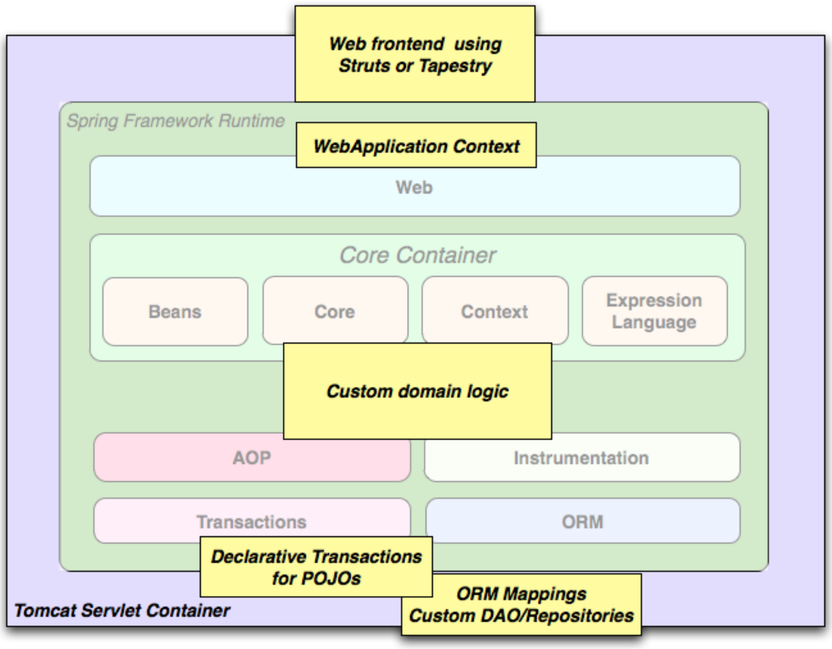
翻译：2.3使用场景
从在资源受限的设备上运行的嵌入式应用程序到使用Spring的事务管理功能和Web框架集成的成熟的企业级应用程序，Spring在许多情况下都使Spring成为合理的选择。

图2.2。 典型的成熟Spring Web应用程序



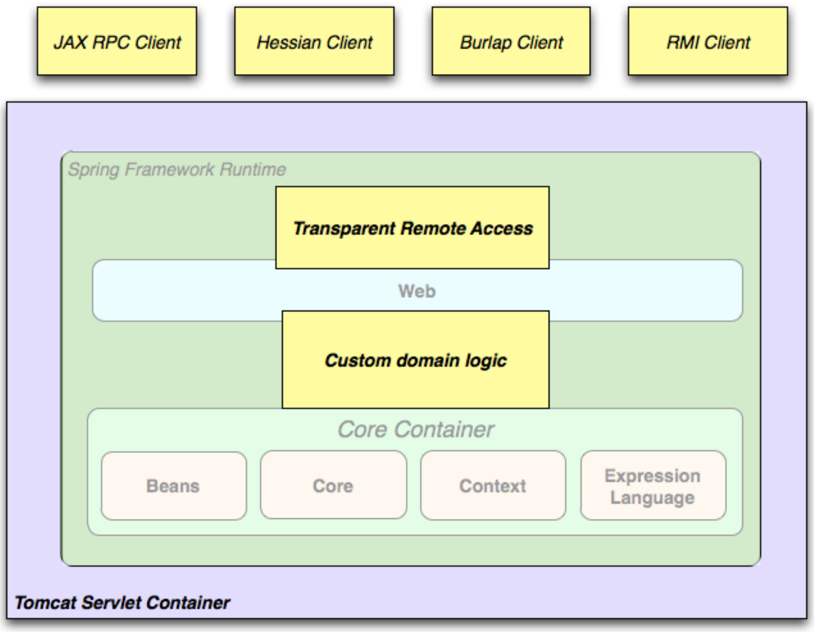
Spring的声明式事务管理功能使Web应用程序具有完全事务性，就像使用EJB容器管理的事务一样。 您所有的自定义业务逻辑都可以通过简单的POJO来实现，并由Spring的IoC容器进行管理。 其他服务包括独立于Web层的对发送电子邮件和验证的支持，使您可以选择在何处执行验证规则。 Spring的ORM支持与JPA，Hibernate和JDO集成在一起； 例如，当使用Hibernate时，您可以继续使用现有的映射文件和标准的Hibernate SessionFactory配置。 表单控制器将Web层与域模型无缝集成，从而无需使用ActionForm或其他将HTTP参数转换为域模型值的类。

图2.3。 使用第三方Web框架的Spring中间层



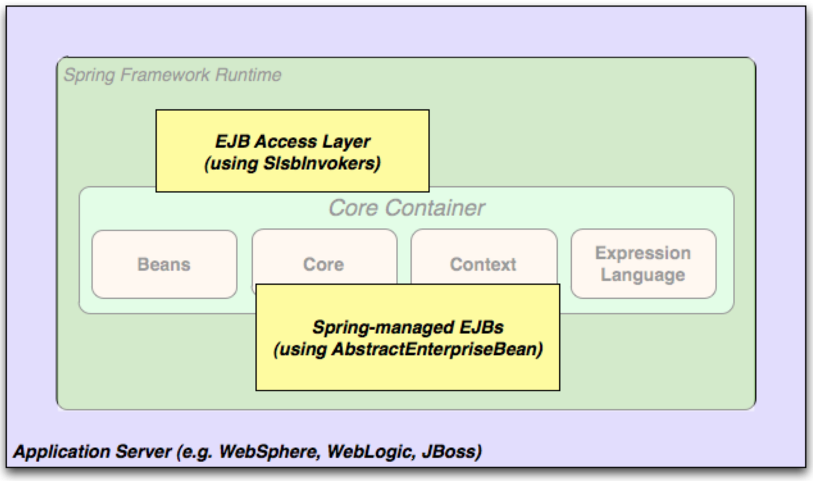
有时情况不允许您完全切换到其他框架。 Spring框架不会强迫您使用其中的所有内容； 这不是一个全有或全无的解决方案。 使用Struts，Tapestry，JSF或其他UI框架构建的现有前端可以与基于Spring的中间层集成在一起，从而可以使用Spring事务功能。 您只需要使用ApplicationContext连接业务逻辑并使用WebApplicationContext集成您的Web层。

图2.4。 远程使用场景



当您需要通过网络服务访问现有代码时，可以使用Spring的Hessian，Burlap，Rmi或JaxRpcProxyFactory类。 启用对现有应用程序的远程访问并不困难。

图2.5。 EJB-包装现有的POJO


Spring Framework Runtime
EJB Access Layer (using SlsbInvokers)
Core Container
Beans
Core
Context
Expression Language
Spring-managed EJBs (using AbstractEnterpriseBean)
Application Server (e.g. WebSphere, WebLogic, JBoss)

Spring框架还为Enterprise JavaBean提供了访问和抽象层，使您可以重用现有的POJO，并将它们包装在无状态会话Bean中，以用于可能需要声明性安全性的可伸缩，故障安全的Web应用程序中。

2.3.1 Dependency Management and Naming Conventions

Dependency management and dependency injection are different things. To get those nice features of Spring into your application (like dependency injection) you need to assemble all the libraries needed (jar files) and get them onto your classpath at runtime, and possibly at compile time. These dependencies are not virtual components that are injected, but physical resources in a file system (typically). The process of dependency management involves locating those resources, storing them and adding them to classpaths. Dependencies can be direct (e.g. my application depends on Spring at runtime), or indirect (e.g. my application depends on `commons-dbcp` which depends on `commons-pool`). The indirect dependencies are also known as "transitive" and it is those dependencies that are hardest to identify and manage.

If you are going to use Spring you need to get a copy of the jar libraries that comprise the pieces of Spring that you need. To make this easier Spring is packaged as a set of modules that separate the dependencies as much as possible, so for example if you don't want to write a web application you don't need the spring-web modules. To refer to Spring library modules in this guide we use a shorthand naming convention `spring-*` or `spring-*.jar,` where `*` represents the short name for the module (e.g. `spring-core`, `spring-webmvc`, `spring-jms`, etc.). The actual jar file name that you use is normally the module name concatenated with the version number (e.g. *spring-core-4.3.25.RELEASE.jar*).

Each release of the Spring Framework will publish artifacts to the following places:

- Maven Central, which is the default repository that Maven queries, and does not require any special configuration to use. Many of the common libraries that Spring depends on also are available from Maven Central and a large section of the Spring community uses Maven for dependency management, so this is convenient for them. The names of the jars here are in the form `spring-*-<version>.jar` and the Maven groupId is `org.springframework`.
- In a public Maven repository hosted specifically for Spring. In addition to the final GA releases, this repository also hosts development snapshots and milestones. The jar file names are in the same form as Maven Central, so this is a useful place to get development versions of Spring to use with other libraries deployed in Maven Central. This repository also contains a bundle distribution zip file that contains all Spring jars bundled together for easy download.

So the first thing you need to decide is how to manage your dependencies: we generally recommend the use of an automated system like Maven, Gradle or Ivy, but you can also do it manually by downloading all the jars yourself.

Below you will find the list of Spring artifacts. For a more complete description of each module, see Section 2.2, "Framework Modules".

Table 2.1. Spring Framework Artifacts

GroupId	ArtifactId	Description
org.springframework	spring-aop	Proxy-based AOP support
org.springframework	spring-aspects	AspectJ based aspects
org.springframework	spring-beans	Beans support, including Groovy
org.springframework	spring-context	Application context runtime, including sch oting abstractions
org.springframework	spring-context-support	Support classes for integrating common es into a Spring application context
org.springframework	spring-core	Core utilities, used by many other Spring
org.springframework	spring-expression	Spring Expression Language (SpEL)
org.springframework	spring-instrument	Instrumentation agent for JVM bootstrap
org.springframework	spring-instrument-tomcat	Instrumentation agent for Tomcat
org.springframework	spring-jdbc	JDBC support package, including DataSo JDBC access support
org.springframework	spring-jms	JMS support package, including helper c ceive JMS messages
org.springframework	spring-messaging	Support for messaging architectures and
org.springframework	spring-orm	Object/Relational Mapping, including JPA upport
org.springframework	spring-oxm	Object/XML Mapping
org.springframework	spring-test	Support for unit testing and integration te mponents
org.springframework	spring-tx	Transaction infrastructure, including DAO A integration
org.springframework	spring-web	Foundational web support, including web based remoting
org.springframework	spring-webmvc	HTTP-based Model-View-Controller and for Servlet stacks
org.springframework	spring-webmvc-portlet	MVC implementation to be used in a Port
org.springframework	spring-websocket	WebSocket and SockJS infrastructure, in messaging support

翻译：2.3.1依赖性管理和命名约定
依赖管理和依赖注入是不同的东西。为了将Spring的这些不错的功能（例如依赖注入）引入到您的应用程序中，您需要组装所需的所有库（jar文件），并在运行时（可能在编译时）将它们放入类路径。这些依赖项不是注入的虚拟组件，而是文件系统中的物理资源（通常）。依赖项管理的过程包括查找这些资源，存储它们并将它们添加到类路径中。依赖关系可以是直接的（例如，我的应用程序在运行时依赖于Spring），也可以是间接的（例如，我的应用程序依赖于取决于commons-pool的commons-dbcp）。间接依赖关系也称为“传递性”，是那些最难识别和管理的依赖关系。

如果要使用Spring，则需要获取包含所需Spring片段的jar库的副本。为了简化操作，Spring被打包为一组模块，这些模块尽可能地分隔了依赖关系，例如，如果您不想编写Web应用程序，则不需要spring-web模块。要在本指南中引用Spring库模块，我们使用简写命名约定spring- *或spring-*。jar，其中*表示模块的简称（例如spring-core，spring-webmvc，spring-jms等。）。您使用的实际jar文件名称通常是与版本号串联的模块名称（例如spring-core-4.3.25.RELEASE.jar）。

Spring Framework的每个发行版都会将工件发布到以下位置：

Maven Central是Maven查询的默认存储库，不需要任何特殊配置即可使用。 Spring依赖的许多常见库也可以从Maven Central中获得，并且Spring社区的很大一部分都使用Maven进行依赖项管理，因此这对他们来说很方便。这里的jar的形式为spring-*-<version>.jar，Maven groupId为org.springframework。
在专门为Spring托管的公共Maven存储库中。除了最终的GA版本外，该存储库还托管开发快照和里程碑。 jar文件的名称与Maven Central的格式相同，因此这是使Spring开发版本与部署在Maven Central中的其他库一起使用的有用位置。该存储库还包含捆绑分发zip文件，该文件包含捆绑在一起以方便下载的所有Spring jar。
因此，您需要决定的第一件事是如何管理依赖项：我们通常建议使用自动化系统，例如Maven，Gradle或Ivy，但是您也可以通过自己下载所有jar来手动进行操作。

在下面，您将找到Spring构件的列表。有关每个模块的更完整说明，请参见第2.2节“框架模块”。

GroupId	ArtifactId	Description
org.springframework	spring-aop	Proxy-based AOP support
org.springframework	spring-aspects	AspectJ based aspects
org.springframework	spring-beans	Beans support, including Groovy
org.springframework	spring-context	Application context runtime, including scheduling and remoting abstractions
org.springframework	spring-context-support	Support classes for integrating common third-party libraries into a Spring application context
org.springframework	spring-core	Core utilities, used by many other Spring modules
org.springframework	spring-expression	Spring Expression Language (SpEL)
org.springframework	spring-instrument	Instrumentation agent for JVM bootstrapping
org.springframework	spring-instrument-tomcat	Instrumentation agent for Tomcat
org.springframework	spring-jdbc	JDBC support package, including DataSource setup and JDBC access support
org.springframework	spring-jms	JMS support package, including helper classes to send/receive JMS messages
org.springframework	spring-messaging	Support for messaging architectures and protocols
org.springframework	spring-orm	Object/Relational Mapping, including JPA and Hibernate support
org.springframework	spring-oxm	Object/XML Mapping
org.springframework	spring-test	Support for unit testing and integration testing Spring components
org.springframework	spring-tx	Transaction infrastructure, including DAO support and JCA integration
org.springframework	spring-web	Foundational web support, including web client and web-based remoting
org.springframework	spring-webmvc	HTTP-based Model-View-Controller and REST endpoints for Servlet stacks
org.springframework	spring-webmvc-portlet	MVC implementation to be used in a Portlet environment
org.springframework	spring-websocket	WebSocket and SockJS infrastructure, including STOMP messaging support

Spring Dependencies and Depending on Spring

Although Spring provides integration and support for a huge range of enterprise and other external tools, it intentionally keeps its mandatory dependencies to an absolute minimum: you shouldn’t have to locate and download (even automatically) a large number of jar libraries in order to use Spring for simple use cases. For basic dependency injection there is only one mandatory external dependency, and that is for logging (see below for a more detailed description of logging options).

Next we outline the basic steps needed to configure an application that depends on Spring, first with Maven and then with Gradle and finally using Ivy. In all cases, if anything is unclear, refer to the documentation of your dependency management system, or look at some sample code - Spring itself uses Gradle to manage dependencies when it is building, and our samples mostly use Gradle or Maven.

Maven Dependency Management

If you are using Maven for dependency management you don’t even need to supply the logging dependency explicitly. For example, to create an application context and use dependency injection to configure an application, your Maven dependencies will look like this:

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

That's it. Note the scope can be declared as runtime if you don't need to compile against Spring APIs, which is typically the case for basic dependency injection use cases.
The example above works with the Maven Central repository. To use the Spring Maven repository (e.g. for milestones or developer snapshots), you need to specify the repository location in your Maven configuration. For full releases:

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

For milestones:

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

And for snapshots:

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

翻译：Spring依赖关系和Spring依赖
尽管Spring为大量企业和其他外部工具提供了集成和支持，但它有意地将其强制性依赖项保持在绝对最低限度：您不必为了自动地找到和下载（甚至自动）大量的jar库而能够 将Spring用于简单的用例。 对于基本的依赖注入，只有一个强制性的外部依赖，即用于日志记录（有关日志记录选项的详细说明，请参见下文）。

接下来，我们概述配置依赖于Spring的应用程序所需的基本步骤，首先使用Maven，然后使用Gradle，最后使用Ivy。 在所有情况下，如果不清楚，请参阅您的依赖项管理系统的文档，或查看一些示例代码-Spring本身在构建时使用Gradle来管理依赖项，而我们的示例大多使用Gradle或Maven。

Maven依赖管理
如果您使用Maven进行依赖性管理，则甚至不需要显式提供日志记录依赖性。 例如，要创建应用程序上下文并使用依赖项注入来配置应用程序，您的Maven依赖项将如下所示：

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

而已。 请注意，如果您不需要针对Spring API进行编译，则可以将范围声明为运行时，这通常是基本依赖项注入用例的情况。

上面的示例适用于Maven Central存储库。 要使用Spring Maven存储库（例如用于里程碑或开发人员快照），您需要在Maven配置中指定存储库位置。 完整版本：

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

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

对于里程碑：

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

对于快照版：

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

Maven "Bill Of Materials" Dependency

It is possible to accidentally mix different versions of Spring JARs when using Maven. For example, you may find that a third-party library, or another Spring project, pulls in a transitive dependency to an older release. If you forget to explicitly declare a direct dependency yourself, all sorts of unexpected issues can arise.

To overcome such problems Maven supports the concept of a "bill of materials" (BOM) dependency. You can import the `spring-framework-bom` in your `dependencyManagement` section to ensure that all spring dependencies (both direct and transitive) are at the same version.

```
<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>
```

An added benefit of using the BOM is that you no longer need to specify the `<version>` attribute when depending on Spring Framework artifacts:

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

Gradle Dependency Management

To use the Spring repository with the Gradle build system, include the appropriate URL in the `repositories` section:

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

You can change the `repositories` URL from `/release` to `/milestone` or `/snapshot` as appropriate. Once a repository has been configured, you can declare dependencies in the usual Gradle way:

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

Ivy Dependency Management

If you prefer to use Ivy to manage dependencies then there are similar configuration options.
To configure Ivy to point to the Spring repository add the following resolver to your `ivysettings.xml`:

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

You can change the `root` URL from `/release/` to `/milestone/` or `/snapshot/` as appropriate.
Once configured, you can add dependencies in the usual way. For example (in `ivy.xml`):

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

翻译：Maven“物料清单”依赖性
使用Maven时，可能会意外地混合使用不同版本的Spring JAR。 例如，您可能会发现第三方库或另一个Spring项目将传递性依赖项引入了较旧的发行版。 如果您忘记自己明确声明直接依赖项，则可能会出现各种意外问题。

为了克服这些问题，Maven支持“物料清单”（BOM）依赖项的概念。 您可以在dependencyManagement部分中导入spring-framework-bom，以确保所有spring依赖项（直接和传递）都处于同一版本。

```
<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的另一个好处是，根据Spring Framework工件，您不再需要指定<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>
```

Gradle依赖管理
要将Spring存储库与Gradle构建系统一起使用，请在存储库部分中包含相应的URL：

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

您可以根据需要将存储库URL从/ release更改为/ milestone或/ snapshot。 一旦配置了存储库，您就可以按照通常的Gradle方式声明依赖项：

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

常春藤依赖管理
如果您更喜欢使用Ivy来管理依赖项，则可以使用类似的配置选项。

要将Ivy配置为指向Spring存储库，请在您的ivysettings.xml中添加以下解析器：

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

您可以将根URL从/ release /更改为/ milestone /或/ snapshot /。

配置完成后，您可以按常规方式添加依赖项。 例如（在ivy.xml中）：

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

Distribution Zip Files

Although using a build system that supports dependency management is the recommended way to obtain the Spring Framework, it is still possible to download a distribution zip file.
Distribution zips are published to the Spring Maven Repository (this is just for our convenience, you don’t need Maven or any other build system in order to download them).
To download a distribution zip open a web browser to https://repo.spring.io/release/org/springframework/spring and select the appropriate subfolder for the version that you want. Distribution files end `-dist.zip`, for example spring-framework-{spring-version}-RELEASE-dist.zip. Distributions are also published for milestones and snapshots.

翻译：分发压缩文件
尽管建议使用支持依赖关系管理的构建系统来获取Spring Framework，但是仍然可以下载发行版zip文件。

发行版本zip发布到Spring Maven存储库（这只是为了我们的方便，您不需要Maven或任何其他构建系统即可下载它们）。

要下载发行版zip，请打开Web浏览器到https://repo.spring.io/release/org/springframework/spring，然后为所需版本选择适当的子文件夹。 分发文件以-dist.zip结尾，例如spring-framework- {spring-version} -RELEASE-dist.zip。 还发布了有关里程碑和快照的发行

版。

2.3.2 Logging

Logging is a very important dependency for Spring because *a)* it is the only mandatory external dependency, *b)* everyone likes to see some output from the tools they are using, and *c)* Spring integrates with lots of other tools all of which have also made a choice of logging dependency. One of the goals of an application developer is often to have unified logging configured in a central place for the whole application, including all external components. This is more difficult than it might have been since there are so many choices of logging framework.

The mandatory logging dependency in Spring is the Jakarta Commons Logging API (JCL). We compile against JCL and we also make JCL `Log` objects visible for classes that extend the Spring Framework. It's important to users that all versions of Spring use the same logging library: migration is easy because backwards compatibility is preserved even with applications that extend Spring. The way we do this is to make one of the modules in Spring depend explicitly on `commons-logging` (the canonical implementation of JCL), and then make all the other modules depend on that at compile time. If you are using Maven for example, and wondering where you picked up the dependency on `commons-logging`, then it is from Spring and specifically from the central module called `spring-core`.

The nice thing about `commons-logging` is that you don't need anything else to make your application work. It has a runtime discovery algorithm that looks for other logging frameworks in well known places on the classpath and uses one that it thinks is appropriate (or you can tell it which one if you need to). If nothing else is available you get pretty nice looking logs just from the JDK (java.util.logging or JUL for short). You should find that your Spring application works and logs happily to the console out of the box in most situations, and that's important.

Using Log4j 1.2 or 2.x

Log4j 1.2 is EOL in the meantime. Also, Log4j 2.3 is the last Java 6 compatible release, with newer Log4j 2.x releases requiring Java 7+.

Many people use Log4j as a logging framework for configuration and management purposes. It is efficient and well-established, and in fact it is what we use at runtime when we build Spring. Spring also provides some utilities for configuring and initializing Log4j, so it has an optional compile-time dependency on Log4j in some modules.

To make Log4j 1.2 work with the default JCL dependency (`commons-logging`) all you need to do is put Log4j on the classpath, and provide it with a configuration file (`log4j.properties` or `log4j.xml` in the root of the classpath). So for Maven users this is your dependency declaration:

Many people use Log4j as a logging framework for configuration and management purposes. It is efficient and well-established, and in fact it is what we use at runtime when we build Spring. Spring also provides some utilities for configuring and initializing Log4j, so it has an optional compile-time dependency on Log4j in some modules.

To make Log4j 1.2 work with the default JCL dependency (`commons-logging`) all you need to do is put Log4j on the classpath, and provide it with a configuration file (`log4j.properties` or `log4j.xml` in the root of the classpath). So for Maven users this is your dependency declaration:

```
<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>
```

And here's a sample log4j.properties for logging to the console:

```
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
```

To use Log4j 2.x with JCL, all you need to do is put Log4j on the classpath and provide it with a configuration file (`log4j2.xml`, `log4j2.properties`, or other supported configuration formats). For Maven users, the minimal dependencies needed are:

```
<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>
```

If you also wish to enable SLF4J to delegate to Log4j, e.g. for other libraries which use SLF4J by default, the following dependency is also needed:

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

Here is an example `log4j2.xml` for logging to the console:

```
<?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>
```

翻译：2.3.2日志

日志记录对于Spring是非常重要的依赖项，因为a）它是唯一的强制性外部依赖项，b）每个人都希望从他们使用的工具中看到一些输出，并且c）Spring与许多其他工具集成在一起，所有这些工具也使日志依赖项的选择。应用程序开发人员的目标之一通常是在整个应用程序（包括所有外部组件）的中央位置配置统一日志记录。由于日志记录框架有很多选择，所以这比以前要困难的多。

Spring中的强制日志记录依赖项是Jakarta Commons Logging API（JCL）。我们根据JCL进行编译，并使扩展Spring框架的类对

JCL日志对象可见。对用户来说重要的是，所有版本的Spring都使用相同的日志库：迁移非常容易，因为即使使用扩展Spring的应用程序也可以保留向后兼容性。我们这样做的方法是使Spring中的模块之一显式依赖于commons-logging（JCL的规范实现），然后使所有其他模块在编译时依赖于该模块。例如，如果您正在使用Maven，并且想知道在哪里获取了对commons-logging的依赖，那么它来自Spring，特别是来自名为spring-core的中央模块。

Commons Logging的好处是您不需要其他任何东西就能使您的应用程序正常工作。它具有一种运行时发现算法，该算法在类路径上众所周知的位置寻找其他日志记录框架，并使用它认为合适的框架（或者您可以告诉它是否需要）。如果没有其他可用的东西，那么您仅从JDK（java.util.logging或简称JUL）就可以获得漂亮的日志。在大多数情况下，您应该会发现自己的Spring应用程序可以正常工作并愉快地登录到控制台，这一点很重要。
使用Log4j 1.2或2.x
[注意]
Log4j 1.2同时处于停产状态。 此外，Log4j 2.3是最新的Java 6兼容版本，而较新的Log4j 2.x发行版则需要Java 7+。

许多人将Log4j用作用于配置和管理目的的日志记录框架。 它是高效且完善的，实际上，这是我们在构建Spring时在运行时使用的。 Spring还提供了一些实用程序来配置和初始化Log4j，因此在某些模块中，它对Log4j具有可选的编译时依赖性。

要使Log4j 1.2与默认JCL依赖项（公共记录）一起使用，您需要做的就是将Log4j放在类路径上，并为其提供配置文件（类路径根目录中的log4j.properties或log4j.xml）。 因此，对于Maven用户，这是您的依赖项声明：

```xml
<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>
```

这是用于登录到控制台的示例log4j.properties：

```
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
```

要将Log4j 2.x与JCL一起使用，您所需要做的就是将Log4j放在类路径上并为其提供配置文件（log4j2.xml，log4j2.properties或其他受支持的配置格式）。 对于Maven用户，所需的最小依赖性为：

```xml
<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来委派给Log4j，例如 对于默认情况下使用SLF4J的其他库，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>
```

这是用于登录到控制台的示例log4j2.xml：

```
<?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>
```

Avoiding Commons Logging

Unfortunately, the runtime discovery algorithm in the standard `commons-logging` API, while convenient for the end-user, can be problematic. If you'd like to avoid JCL's standard lookup, there are basically two ways to switch it off:

1. Exclude the dependency from the `spring-core` module (as it is the only module that explicitly depends on `commons-logging`)
2. Depend on a special `commons-logging` dependency that replaces the library with an empty jar (more details can be found in the SLF4J FAQ)

To exclude commons-logging, add the following to your `dependencyManagement` section:

```
<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>
```

Now this application is currently broken because there is no implementation of the JCL API on the classpath, so to fix it a new one has to be provided. In the next section we show you how to provide an alternative implementation of JCL using SLF4J.

Using SLF4J with Log4j or Logback

The Simple Logging Facade for Java (SLF4J) is a popular API used by other libraries commonly used with Spring. It is typically used with Logback which is a native implementation of the SLF4J API.

SLF4J provides bindings to many common logging frameworks, including Log4j, and it also does the reverse: bridges between other logging frameworks and itself. So to use SLF4J with Spring you need to replace the `commons-logging` dependency with the SLF4J-JCL bridge. Once you have done that then logging calls from within Spring will be translated into logging calls to the

SLF4J API, so if other libraries in your application use that API, then you have a single place to configure and manage logging. A common choice might be to bridge Spring to SLF4J, and then provide explicit binding from SLF4J to Log4j. You need to supply several dependencies (and exclude the existing `commons-logging`): the JCL bridge, the SLF4j binding to Log4j, and the Log4j provider itself. In Maven you would do that like this

```xml
<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>
    <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>
```

A more common choice amongst SLF4J users, which uses fewer steps and generates fewer dependencies, is to bind directly to Logback. This removes the extra binding step because Logback implements SLF4J directly, so you only need to depend on just two libraries, namely `jcl-over-slf4j` and `logback`):

```xml
<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>
```

翻译：避免commons logging

不幸的是，尽管标准的公共记录API中的运行时发现算法虽然方便了最终用户，但却可能会出现问题。 如果您想避免JCL的标准查找，基本上可以通过两种方法将其关闭：

从spring-core模块中排除依赖项（因为它是唯一明确依赖commons-logging的模块）

依赖于特殊的commons-logging依赖关系，该依赖关系将库替换为空的jar（更多详细信息可以在SLF4J FAQ中找到）

要排除commons-logging，请将以下内容添加到您的dependencyManagement部分：

```xml
<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，由于在类路径上没有JCL API的实现，该应用程序当前已被破坏，因此要修复该应用程序，必须提供一个新的应用程序。在下一节中，我们将向您展示如何使用SLF4J提供JCL的替代实现。

将SLF4J与Log4j或Logback一起使用

Java的简单日志外观（SLF4J）是Spring常用的其他库所使用的流行API。它通常与Logback一起使用，Logback是SLF4J API的本机实现。

SLF4J提供了对许多常见日志记录框架（包括Log4j）的绑定，并且它的作用相反：在其他日志记录框架与其自身之间的桥梁。因此，要将SLF4J与Spring一起使用，您需要用SLF4J-JCL桥替换commons-logging依赖项。完成此操作后，Spring中的日志记录调用将转换为对SLF4J API的日志记录调用，因此，如果应用程序中的其他库使用该API，那么您就可以在一个地方配置和管理日志记录。

常见的选择可能是将Spring桥接到SLF4J，然后提供从SLF4J到Log4j的显式绑定。您需要提供几个依赖项（并排除现有的commons-logging）：JCL桥，绑定到Log4j的SLF4j以及Log4j提供程序本身。在Maven中，您将像这样

```xml
<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>
    <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>
```

在SLF4J用户中，更常见的选择是直接绑定到Logback，该步骤使用较少的步骤并生成较少的依赖项。 因为Logback直接实现SLF4J，所以省去了额外的绑定步骤，因此您只需要依赖两个库，即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>
```

Using JUL (java.util.logging)

Commons Logging will delegate to `java.util.logging` by default, provided that no Log4j is detected on the classpath. So there is no special dependency to set up: just use Spring with no external dependency for log output to `java.util.logging`, either in a standalone application (with a custom or default JUL setup at the JDK level) or with an application server's log system (and its system-wide JUL setup).

Commons Logging on WebSphere

Spring applications may run on a container that itself provides an implementation of JCL, e.g. IBM's WebSphere Application Server (WAS). This does not cause issues per se but leads to two different scenarios that need to be understood:
In a "parent first" ClassLoader delegation model (the default on WAS), applications will always pick up the server-provided version of Commons Logging, delegating to the WAS logging subsystem (which is actually based on JUL). An application-provided variant of JCL, whether standard Commons Logging or the JCL-over-SLF4J bridge, will effectively be ignored, along with any locally included log provider.
With a "parent last" delegation model (the default in a regular Servlet container but an explicit configuration option on WAS), an application-provided Commons Logging variant will be picked up, enabling you to set up a locally included log provider, e.g. Log4j or Logback, within your application. In case of no local log provider, regular Commons Logging will delegate to JUL by default, effectively logging to WebSphere's logging subsystem like in the "parent first" scenario.
All in all, we recommend deploying Spring applications in the "parent last" model since it naturally allows for local providers as well as the server's log subsystem.

翻译：使用JUL（java.util.logging）
如果没有在类路径上检测到Log4j，默认情况下Commons Logging将委托给java.util.logging。因此，不需要设置任何特殊的依赖项：只需在独立应用程序（具有JDK级别的自定义或默认JUL设置）或应用程序服务器的外部应用程序中，使用不带外部依赖项的Spring即可将日志输出到java.util.logging。日志系统（及其系统范围内的JUL设置）。

在WebSphere上的Commons Logging
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 Application Server（WAS）。这本身不会引起问题，但是会导致需要理解两种不同的情况：

在“父先” ClassLoader委托模型（WAS的默认值）中，应用程序将始终使用服务器提供的Commons Logging版本，委派给WAS日志子系统（实际上基于JUL）。应用程序提供的JCL变体，无论是标准Commons Logging还是JCL-over-SLF4J桥，都会与任何本地包含的日志提供程序一起被有效忽略。

使用“父辈”委托模型（常规Servlet容器中的默认值，但WAS上有显式配置选项），将选择应用程序提供的Commons Logging变体，从而使您能够设置本地包含的日志提供程序，例如您的应用程序中的Log4j或Logback。在没有本地日志提供程序的情况下，默认情况下，常规的Commons Logging将委派给JUL，从而有效地登录到WebSphere的日志记录子系统，如“父先”方案。

总而言之，我们建议在“父后”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。