Spring框架参考文档 4.0.0正式版

英文原版：spring-framework-reference
翻译：碎
QQ：413615763
Weibo：碎　　le！
Email：not-three@foxmail.com

欢迎反馈，请允许我向那些为改善翻译质量而提供宝贵建议的敬爱的人表示感谢。

翻译进度

2013.12.16 | 2.2 模块
2013.12.17 | 2.3 使用场景:依赖管理和命名约定之前
2013.12.18 | 2.3 使用场景:依赖管理和命名约定
2013.12.18 | 2.3 使用场景:日志-不使用Commons Logging

推荐阅读

暂无

Spring框架概览

当您构建企业级应用时，Spring框架为您提供了一种可选的轻量级解决方案和一站式服务。Spring是模块化的，您可以选择只引入那些您需要的部件，而不用引入余下的部件。比如，您可以只使用IoC容器并加载Struts到它的顶层。您也可以只使用与Hibernate集成相关的代码或者JDBC抽象层。Spring框架支持声明式事务管理，通过RMI或者web服务远程调用您的业务逻辑，还为数据持久化提供了多种可选方案。她具备MVC框架的所有特性，您可以在您的软件中通过AOP隐式集成这些特性。

Spring遵循非侵入式设计，也就是说您的领域逻辑代码通常不会对框架本身有所依赖。在您的集成面（比如数据访问层）会存在一些与数据访问技术相关的依赖包和Spring库。尽管如此，她还是能很从容的隔离开这些依赖和其余的代码库。

本文档是关于Spring框架特性的参考指南。对于本文档，如果您有任何需求，建议，或疑问，请在用户邮件列表或者用户支持论坛提交。

第一次亲密接触Spring

本参考指南涵盖了Spring框架的详尽信息，全面说明了框架具备的所有特性，同时还涉及到Spring基础概念（如依赖注入）的背景。

如果您真的是第一次接触Spring，建议选择更轻松的Getting Started指南作为起点，这些指南可以在官方网站找到。为了更容易被读者消化，这些指南严格遵循了以任务为中心的编写模式。这

些指南也涵盖了Spring portfolio的其他项目，当您解决某一特定领域问题时，可以作为参考。

Getting Started Building a RESTful Web Service是初次体验Spring的极佳选择。

Spring框架简介

Spring框架作为一个Java平台，对Java应用开发提供了全方位的基础支持。由于Spring接管了基础设施，使您只需集中精力关注应用本身。

Spring帮助您从“简单原始的Java对象”（POJO）开始构建应用，在POJO中准确无误的适配企业服务。这种能力不仅适用于“Java SE”编程模型，对完整“Java EE”和部分“Java EE”也同样适用。

作为一位应用开发者，您将会在Spring平台中获利，举例说明下：

- 在数据库事务里实现Java方法，不需要处理事务API。
- 使用本地Java方法实现远程过程，不需要处理远程调用API。
- 使用本地Java方法实现管理操作，不需要处理JMX的API。
- 使用本地Java方法实现消息处理器，不需要处理JMS的API。

依赖注入和控制反转

背景

> *“问题是，它们反向控制了哪一方面？”2004年，Martin Fowler在他的个人站点上提出了关于控制反转（IoC）的问题。Fowler建议重新制定原则使之更加自然，由此提出了依赖注入。*

想要深入了解Ioc和DI，请移步Fowler的文章。

不确切的说，Java应用囊括了从局限性的小应用到N层服务端的企业应用的整个范围，很经典的组合了许多互相配合的用于构建适当应用的一系列对象。由此，应用中的这些对象彼此依赖。

Java平台提供了丰富的应用开发工具，但是它依旧缺少能够将基础构件块组合成一致整体的方式，而仅仅把这些难题抛给架构师和开发者。是的，您可以通过各种设计模式（诸如工厂，抽象工厂，构建者，装饰者和服务定位器）把各种不同的类和对象实例组织起来用于构建应用。然而，这些模式仅仅是：最佳实践的命名，并且描述了这个模式是做什么的，什么情况下使用，解决哪些问题，诸如此类。模式是标准化的最佳实践，但您仍需要在应用中亲自实现。

Spring框架的控制反转（IoC）组件为您提供了标准化方案（一种可以把不同的部件组织成一个完整且可即刻运行的应用的方案）来解决这类问题。Spring框架将标准化的设计模式进一步编程实现了第一级对象，从而您可以在您自己的应用中直接集成它们。无数的组织和机构都是以这种方式来采用Spring框架来设计健壮的可维护的应用。

模块

Spring框架包含了很多特性，大约由20个模块组成。如下图所示，这些模块主要分成核心容器，数据访问/集成，Web，面对切面编程（AOP），基础组件和测试。

图 2.1 Spring框架概览

核心容器

核心容器包含Core，Beans，上下文和表达式语言这些模块。

Core和Beans这两个模块是框架的基础，拥有IoC和依赖注入的特性。BeanFactory巧妙实现了工厂模式，隐去了单例相关的代码，帮助您解除了配置信息和实际编程逻辑的依赖规范之间的耦合。

上下文模块建立在由Core和Beans模块提供的坚实基础之上：这意味着以框架风格的方式来访问对象跟JNDI注册是类似的。上下文模块继承了Beans模块的特性，并且增加了对国际化的支持（如使用资源包），事件的预测，资源的加载和上下文隐式产物（如Servlet容器）。上下文模块还支持“Java EE”特性，如EJB，JMX和基本远程调用。ApplicationContext接口处于上下文模块的中心位置。

表达式语言模块提供了强大的表达式语言，在代码执行时可以用来查询和操作对象图。它是JSP 2.1规范中指定的统一表达式语言（统一EL）的一种扩展。这语言支持设置和获取属性值，属性赋值，方法调用，访问数组、集合和索引的上下文，逻辑操作符和运算操作符，命名的变量，还可以根据名称从Spring的IoC容器中获取对象。它也支持列表投影和选择，以及列表聚合。

数据访问/集成

数据访问/集成这层包含JDBC，ORM，OXM，JMS和事务模块。

JDBC模块提供JDBC抽象层，隐去了冗长乏味的JDBC代码，并为数据库厂商规定的错误代码进

行了解析。

ORM模块把那些受欢迎的对象关系映射API封装在集成层，这些API包括JPA、JDO和Hibernate。加入ORM包后，您可以使用任何O/R映射框架，同时还享有Spring框架所提供的其他特性，譬如之前提到的简洁的声明式事务管理特性。

OXM模块提供了支持对象/XML映射实现的抽象层，这些实现包括JAXB，Castor，XML Beans，JiBX以及XStream。

Java消息服务（JMS）模块拥有制造和消费消息的特性。

事务模块为那些实现了特定接口的类和您所有的POJO提供程式化的和隐式的事务管理的支持。

Web

Web层包含Web，Web-Servlet，WebSocket和Web-Portlet模块。

Spring的Web模块拥有基本的面向Web集成的特性，譬如multipart附件上传功能，用servlet监听器和面向Web应用上下文实现IoC容器的初始化。它也支持Spring的web相关部分的远程调用。

Web-Servlet模块内置了Spring的web应用的模型-视图-控制（MVC）实现。Spring的MVC框架清晰隔离了领域模型代码和web表单，同时享有Spring框架所提供的其他特性。

Web-Porlet模块提供了用于portlet环境的MVC实现，以及镜像到Web-Servlet模块功能。

AOP和基础组件

Spring的AOP模块内置了符合联盟标准的支持自定义的面对切面编程的实现，比如，使用方法拦截器和切入点可以清晰的解除那些实现了的并且应当分离的功能代码的耦合。您也能通过使用类似于.NET元数据方式的源码层元数据功能合并行为信息到您的代码中。

独立的Aspects模块集成了AspectJ。

基础组件模块提供类代码的基础支持，实现了能够在特定应用服务器内使用的类加载器。

测试

测试模块集成了附带JUnit或者TestNG的Spring组件以支持测试。它支持Spring的ApplicationContexts和对应上下文缓存的一致性加载。它还提供了模拟对象帮助您在隔离环境中测试您的代码。

使用场景

无论是小应用，还是完全成熟的同时运用Sprin的事务管理功能和她的web框架集成的企业应用，Spring依靠前文提及的构件块成为了这些场景下的一个合理选择。

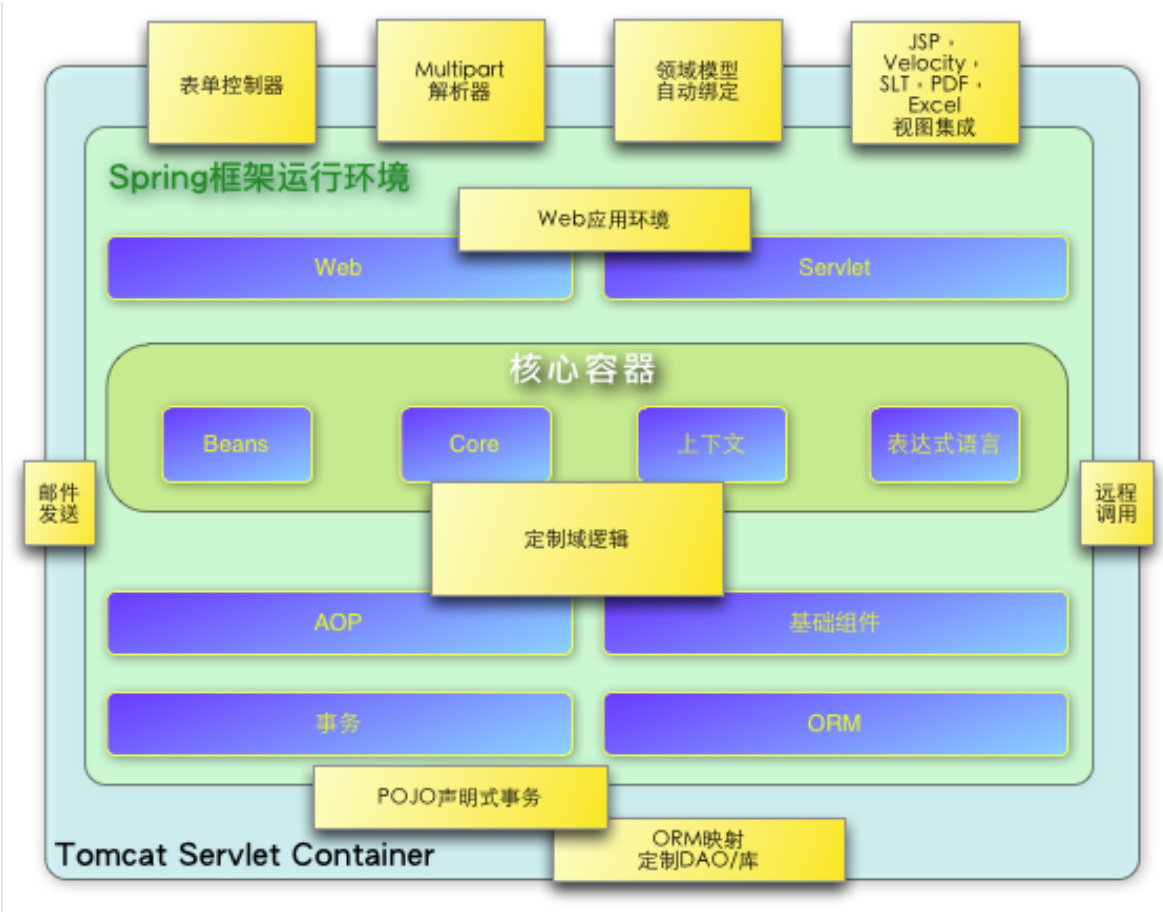

图 2.2 成熟的标准Spring的web应用场景

Spring的声明式事务管理特性使得web应用具有完整的事务性，就如同您使用EJB容器来管理事务一样。所有您定制的业务逻辑都可以用简单的POJO来实现，用Spring的IoC容器来管理。Spring还提供了邮件发送和独立于web层的验证器（可以让您选择需要进行规则验证的地方）的附加服务。Spring的ORM支持，集成了JPA，Hibernate和JDO；比方说，当您使用Hibernate时，您可以继续使用您已有的映射文件和标准的Hibernate的SessionFactory配置。具有领域模型的web层无缝集成了表单控制器，您的领域模型不再需要ActionForms，或者其他那些能把HTTP参数转化成相应值的类。

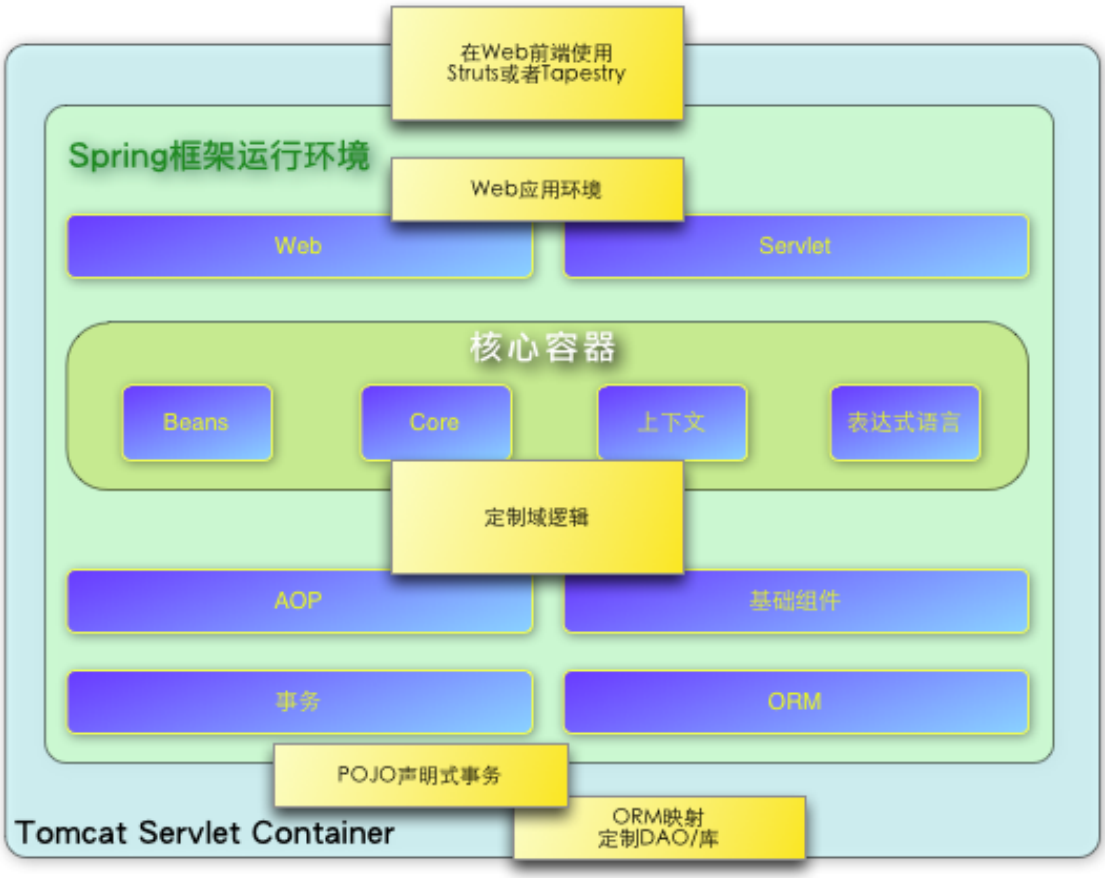

图 2.3 使用第三方web框架的Spring中间层

某些情况不允许您将现有框架彻底替换为其他不同的框架。Spring不会强制您使用她的全部功能；她不是那种非此即彼的的解决方案。基于Spring的中间层能够和已经采用了Struts，Tapestry，JSF或其他UI框架的的前端集成，并允许您使用Spring的事务特性。您仅仅需要做的是，使用ApplicationContext来装配您的业务逻辑，同时使用WebApplicationContext来集成您的web层。

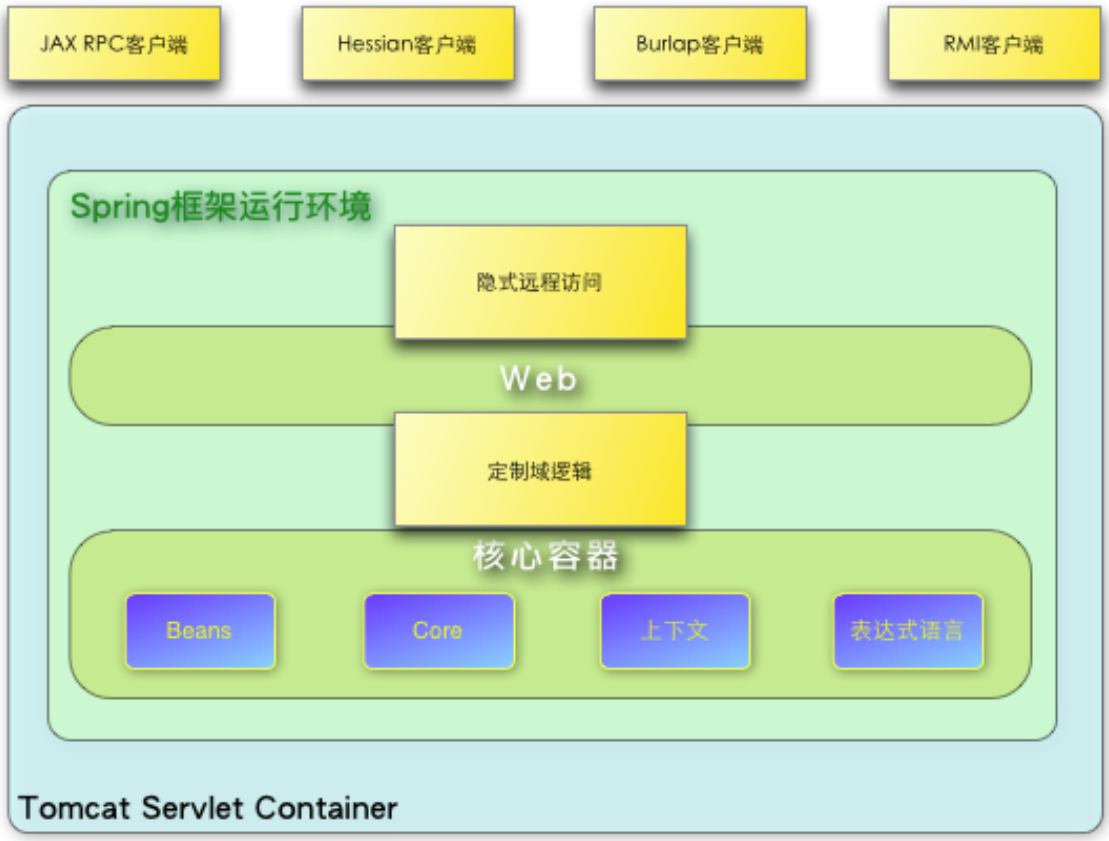

图 2.4 远程访问使用场景

当您需要通过web服务访问已有代码，您可以使用Spring的Hessian（或Burlap，或Rmi，或JaxRpc）的代理工厂类。实现远程访问已有应用，并不是很难的事。

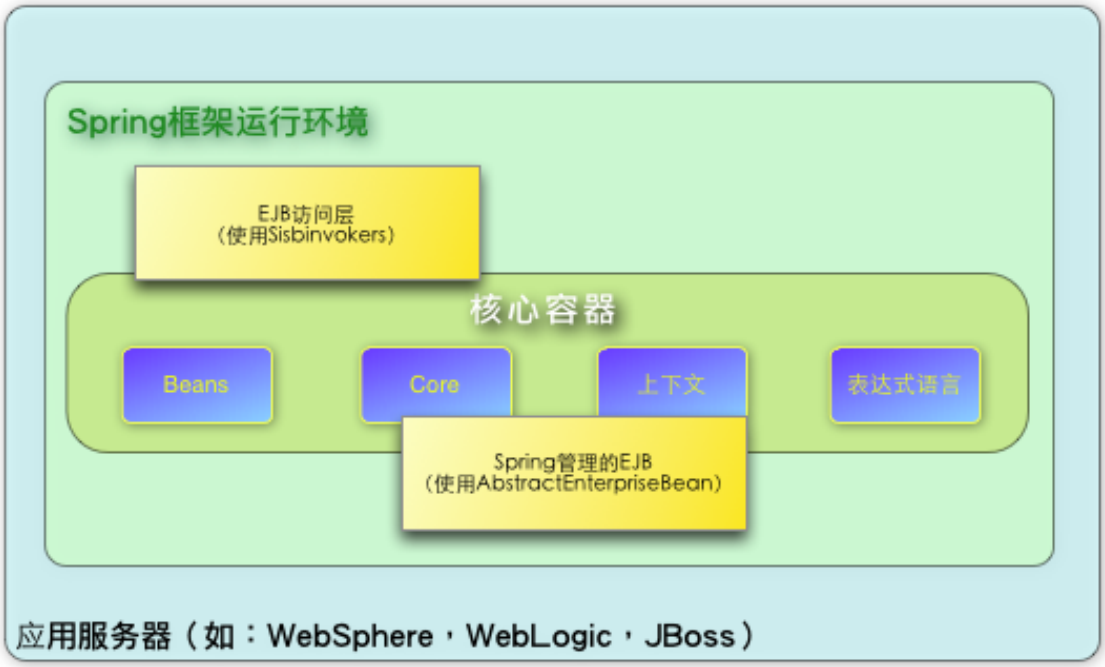

图 2.5 EJB - 封装已有的POJO

Spring框架给企业JavaBean也提供了访问和抽象层，使您能够复用您已有的POJO，并把它们封装到无状态会话bean里。这些无状态会话bean往往用于可扩展的可靠的还可能需要声明式安全的web应用。

依赖管理和命名约定

依赖管理和依赖注入是完全不同的概念。要在您的应用里获得Spring很赞的那些特性（譬如依赖注入），您需要装配所有必要的库（jar文件）并在运行时（也可能在编译时）把这些库放到您的类路径里。这些注入的依赖不是虚拟的部件，而是在您系统（代表性的）中的实际资源。依赖管理的过程牵扯到资源的定位，储存以及如何把资源载入到类路径里。依赖可以是直接的（譬如我的应用在运行时依赖于Spring），也可以是间接的（譬如我的应用依赖于commons-dbcp，而commons-dbcp又依赖于commons-pool）。间接的依赖也被称为“传递”，这是那些最难识别和管理的依赖。

如果您打算使用Spring，您要获得那些包含您所需的Spring部件的jar库副本。为了更容易给开发者使用，Spring由一系列模块打包而成，这些模块又尽可能分离它们的依赖。由此可知，假如您不想要编写web应用，您可以不引入spring-web模块。当这个教程提到Spring库时，我们使用一种速记的命名约定spring-*或者spring-*.jar，这里的*代表模块名的缩写（如spring-core，spring-webmvc，spring-jms，诸如此类）。您实际使用的jar文件名通常是模块名后面跟着版本号（如spring-core-4.0.0.RELEASE.jar）。

每个Spring框架的正式版都会发布工件到下列位置：

- Maven Central，Maven查询的默认库，不需要任何特殊配置就可以使用。很多Spring依赖的常用库也可以从Maven Central找到，而且很大一部分Spring社区用户使用Maven来管理依赖，这对他们来说很便捷。这里的jar是以spring-*-.jar的形式命名的，对应的Maven组id则是org.springframework。
- 专门用于托管Spring的公用Maven库。除了最终GA正式版外，这个库也托管开发快照版和里程碑版。jar文件的命名规则和Maven Central是一致的。为了更容易被下载，这个库还包含打包了所有Spring的jar文件的分发zip文件。

因此您首先需要决定怎么管理您的依赖：我们一般推荐使用自动化系统，譬如Maven，Gradle或者Ivy，您也可以亲自手工下载所有的jar文件来进行管理。在这个章节的后面我们会进行详细说明。

Spring依赖和依赖Spring

Spring为众多的企业级和其他额外工具提供了集成和支持。尽管如此，她还是有意识的保持她的强制依赖尽可能少。仅用Spring来构建简单的使用案例，您不应当去定位大量的jar库并下载（即使是自动的）。基本的依赖注入只需要一个额外的强制依赖用于日志管理（后面会更加详细的介绍日志选项）。

接着我们会描绘配置一个依赖Spring的应用所应遵循的基本步骤，依次会涉及到Maven，Gradle和Ivy。所有案例里，如果有任何不清楚的，参考您的依赖管理系统相对应的文档，或者看一些示例代码。Spring编译时，是用Gradle来管理依赖的，我们的示例大部分使用Gadle或者Maven。

Maven依赖管理

如果您使用Maven来管理依赖，您甚至不需要显式提供日志依赖。比方说，创建一个应用上下文，使用依赖注入来配置一个应用，您的Maven依赖如下所示：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.0.0.RELEASE</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

就是这样子。注意：如果您不需要跟Spring的API一起编译，scope可以声明为runtime，基本的依赖注入的使用案例就是这种情况下的典型案例。

上述例子用Maven Central库就可以运行。当要使用Spring的Maven库（譬如使用里程碑版的或者开发快照版的），您需要在您的Maven配置里指定Maven库位置。正式版配置如下：

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

里程碑版配置如下：

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

快照版配置如下：

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

Maven“物料清单”依赖

当使用Maven时，有可能会混淆Spring的jar库的不同版本。比方说，您可能寻找一个第三方库，或者另一个Spring项目，刚好依赖传递到一个较旧的正式版。如果您忘记显式声明一个直接依赖，可能会发生种种无法预料的问题。

为了解决这类问题，Maven支持“物料清单”（BOM）依赖的概念。您可以在dependencyManagement节点导入spring-framework-bom来保证所有Spring的依赖（无论是直接还是传递的）都采用了同样的版本。

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>4.0.0.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

使用BOM的额外好处是，当依赖于Spring框架工件时，您不需要再指定<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

用基于Gradle的构建系统来使用Spring库，需要在repositories节点设置合适的URL。

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

您可以根据自己需要改变repositories URL的值，从/release，/milestone，/shapshot中选择合适的。一旦配置好一个库，您就可以通过Gradle的常规方式来声明依赖了。

```
dependencies {
    compile("org.springframework:spring-context:4.0.0.RELEASE")
    testCompile("org.springframework:spring-test:4.0.0.RELEASE")
}
```

Ivy依赖管理

假如您更喜欢使用Ivy来管理依赖，也存在类似的配置选项。需要在您的ivysettings.xml增加下面的解析器来配置Ivy指向的Spring库。

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

您可以从/release，/milestone，/snapshot/中选择一个合适的来改变根URL。一旦配置好，您就能以一般的方式来增加依赖了。譬如（在ivy.xml里）：

```
<dependency org="org.springframework"
    name="spring-core" rev="4.0.0.RELEASE" conf="compile->runtime"/>
```

分发Zip文件

使用支持依赖管理的构建系统是获取Spring框架的推荐方式，但是仍有办法下载一个分发zip文件。

分发zip发布在Spring的Maven库（这只是为了方便我们自己，您不需要Maven或者其他任何构建系统就可以下载到它们）。

要下载一个分发zip，要先在web浏览器打开

http://repo.spring.io/release/org/springframework/spring，选择您想要的版本所对应的子目录。分发zip文件是以-dist.zip结尾的，如spring-framework-4.0.0.RELEASE-dist.zip。里程碑版和快照版同样也发布了分发包。

日志

日志是Spring一个非常重要的依赖。因为 a) 它是唯一强制的外部依赖，b) 每个人都喜欢看到他们所使用工具的输出信息，c) Spring集成了许多其他同样也选择了日志依赖的工具。应用开发者的诉求之一是想要拥有一个能在核心位置给整个应用包括所有的外部组件提供统一配置的日志。而自从有了太多的日志框架可供选择以后，这个局面变得更加困难。

Spring强制的日志依赖是Jakarta Commons Logging API（JCL）。我们编译时依赖JCL，我们同时创建了那些“JCL Log”对象，这些对象对继承了Spring框架的类是可见的。Spring所有版本使用了相同的日志库，这对用户很重要：由于维持了向前兼容，使得易于迁移，对那些继承于Spring的应用也是这样。我们是这样处理的，把Spring的模块之一显式依赖于commons-logging（JCL的标准实现）并在编译时把它作为其他模块的依赖。假设您使用Maven，想知道哪里依赖了commons-logging，那么就会发现是Spring，具体来说是那个叫做spring-core的核心模块。

有关commons-logging很赞的是，您不需要任何额外的东西就能使您的应用运行起来。它内置了一个运行时侦测算法：在类路径里众所周知的位置寻找其他日志框架，使用那个它认为合适的（或者如果您需要的话，您也可以给它设置一个）。假如找不到任何日志框架，您将从JDK（java.util.logging或者简称JUL）获得相当美观的日志。这时您应该就会看到您的Spring应用运行起来了，并且在大多数情况下日志信息立刻会欢乐的出现在控制台上，而这对您来说恰恰是重要的。

不使用Commons Logging

不幸的是，内置在commons-logging的运行时侦测算法是有问题的，尽管对终端用户来说很实用。如果我们能让时间回到过去，立刻开展一个叫做Spring的新工程，Spring将会使用不同的日志依赖。首选可能就是“Simple Logging Facade for Java”（SLF4J），在应用里，与Spring一起被人们广泛使用的其他工具也采用了它。

切换掉commons-logging是很容易的：只需要确认在运行时它不在类路径里。您可以按照Maven的规则移除这个依赖，而且由于Spring依赖的声明方式的关系，您只需要这样做一次。

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.0.0.RELEASE</version>
        <scope>runtime</scope>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

现在，这应用可能就瘫痪了，因为在类路径里没有JCL的API实现，因此要提供一个新的实现来修复。在下一章节我们将以SLF4J作为例子向你展示怎样提供可替代的JCL实现。