Spring框架：用于开发稳健Java应用程序的开源Java平台

**Dashrath Mane, Ketaki Chitnis, Namrata Ojha**

***摘要*— *本文介绍了Spring框架的基本概念。Spring框架是一个提供了全面基础架构支持的开源的Java平台，可以非常轻松、快速地开发健壮的Java应用程序。Spring框架是一个轻量级的解决方案，是构建企业级应用的潜在的一站式服务。***

***索引术语— 面向切面编程, 依赖注入, Ioc 容器, ORM.***

# 一、引言

Spring是企业级Java最流行的应用程序开发框架。全球数百万开发者使用Spring框架创建高性能、易测试、可复用的代码。Spring框架是一个开源的Java平台，最初由Rod Johnson编写，并于2003年6月首次根据Apache2.0许可发布。

Spring在大小和透明度方面是轻量级的。最初版本的Spring大约为2MB。Spring框架的核心功能可用于开发任何Java应用程序，但也有用作在Java EE平台上构建Web应用程序的扩展。Spring框架的目标是通过启用基于POJO的编程模型，使J2EE开发变得简易，促进良好的编程实践。Spring框架为现代基于Java的企业级应用程序提供了全面的编程和配置模型-适用于任何类型的部署平台。Spring的一个关键要素是应用程序级别的基础设施支持：Spring专注于企业级应用程序的“管道”，以便团队可以专注于应用程序级别上的业务逻辑，无需与特定部署环境产生不必要的联系。Spring包括：

* 使用基于XML注释的配置文件进行灵活的依赖注入。
* 使用基于代理和基于AspectJ的变体，对面向切面编程提供高级支持。
* 对Hibernate和Quartz等常见开源框架提供一流的支持。
* 构建RESTful MVC应用程序和服务端的灵活的Web框架。

**Manuscript published on 30 July 2013.**

**\***Correspondence Author(s)

**Mr Dashrath Mane**, Assistant Professor, Department of MCA, V.E.S. Institute of Technology, Mumbai, India.

**Miss NamrataOjha**, Final Year MCA Student, V.E.S. Institute of Technology, Mumbai, India.

**Miss KetakiChitnis**, Final Year MCA Student, V.E.S. Institute of Technology, Mumbai, India.

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an [open access](https://www.openaccess.nl/en/open-publications) article under the

CC-BY-NC-ND licens[e http://creativecommons.org/licenses/by-nc-nd/4.0/](http://creativecommons.org/licenses/by-nc-nd/4.0/)

Spring采用模块化设计，允许逐步采用各个部分，例如核心容器或JDBC支持。虽然所有的Spring服务都非常适合Spring核心容器，但也有许多服务可以在容器之外以编程方式使用。

支持的部署平台包括独立应用程序、Tomcat和WebSphere等Java EE服务器。Spring也是主要云平台上支持的Java的一等公民，例如Heroku、Google App Engine、Amazon Elastic Beanstalk和VMware的Cloud Foundry。[1]

# 二、Spring框架架构

Spring可能成为您所有企业应用程序的一站服务；然而Spring是模块化的，允许您提供适合您的模块，而不必引入其余模块。

Spring框架提供了大约20个模块，可以根据应用程序需求使用。

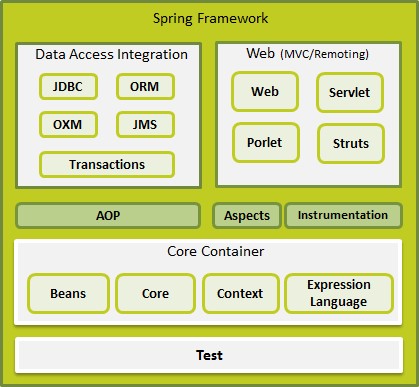


图1. Spring框架结构

## A. 核心容器

核心容器由Core、Beans、Context和Expression Language模块组成，其详细信息如下：

* Core模块提供了框架的基本部分，包括Ioc（控制反转）和依赖注入功能。
* Beans模块 提供了BeanFactory，它是工厂模式的一个复杂实现。
* Context模块建立在核心和Beans模块提供的坚实基础之上，它是访问任何已定义和配置对象的媒介。ApplicationContext接口是Context模块的焦点。
* Expression Language模块提供了一种强大的表达式语言，用于在运行时查询和操作对象图。

## B. 数据访问/集成

数据访问/集成层由JDBC、ORM、OXM、JMS和Transaction模块组成，其详细信息如下：

* JDBC模块提供了一个JDBC抽象层，从而无需进行繁琐的JDBC相关编码。
* ORM模块为流行的对象关系映射API提供了集成层，包括JPA、JDO、Hibernate和iBatis。
* OXM模块提供了一个抽象层，支持JAXB、Castor、XMLBeans、JiBX和XStream的对象/XML映射实现。
* JMS服务（Java Message Service，Java消息服务）。JMS模块包含生成和使用消息的功能。
* Transcation模块支持针对实现特殊接口的类和所有POJO的编程式和声明式事务管理。

## C. 网络

Web层由Web、Web-Servlet、Web-Struts、Web-Portlet模块组成，具体模块如下：

* 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，以及通过Servlet监听器 和面向Web的应用程序上下文来初始化Ioc容器。
* Web-Servlet模块包含Spring针对Web应用程序的模型视图控制器（MVC）实现。
* Web-Struts模块包含在Spring应用程序中集成经典Struts Web层的支持类。

***D. 其他***

* AOP模块提供面向切面的编程实现，允许您定义方法拦截器和切入点来干净地解耦应分离功能的代码。
* Aspects模块提供与AspectJ的集成，AspectJ是一个强大且成熟的面向切面编程（AOP）框架。
* Instrumentation模块提供类检测支持和类加载器实现，供特定应用程序使用。
* Test模块支持使用JUnit或TestNG框架测试Spring组件。

# 三、Spring IOC容器

Spring容器是Spring框架的核心。容器将创建对象、将它们连接在一起、配置它们并管理从创建到销毁的完整生命周期。Spring容器使用依赖注入（DI）来管理组成应用程序的组件。这些对象称为Spring Bean，我们将在下一章中讨论。

容器通过读取提供的配置元数据来获取有关实例化、配置和组装哪些对象的指令。配置元数据可以用XML、Java注释或Java代码表示。下图是Spring工作原理的高级视图。Spring Ioc容器利用Java POJO类和配置元数据生成完全配置且和执行的系统或应用程序。

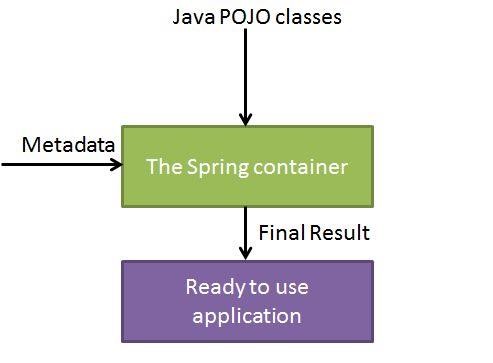


图2. Spring IoC 容器

Spring提供以下两种不同类型的容器。

## A. Spring BeanFactory容器

这是最简单的容器，提供对DI的基本支持。BeanFactory和相关接口（如BeanFactoryAware、InitializingBean、DisposableBean）仍然存在于Spring中，以便与大量集成了Spring的第三方框架保持向后兼容。

***B. Spring ApplicationContext容器***

这个容器增加了更多面向企业的功能，例如从属性文件中解析文本消息的能力，以及向感兴趣的事件监听器发布应用程序事件的能力。这个容器由 org.springframework.context.ApplicationContext接口定义。

ApplicationContext 容器包括了BeanFactory容器的所有功能，因此通常推荐使用ApplicationContext而不是BeanFactory。BeanFactory仍然可以用于轻量级应用程序，比如移动设备或基于小程序的应用程序，这些场景对数据量和速度有较高的要求。

## C. Beans

构成你应用程序骨干并由Spring IoC容器管理的对象称为bean。bean是由Spring IoC容器实例化、组装和管理的对象。这些bean是通过你提供给容器的配置元数据创建的，例如，以XML <bean/> 定义的形式。

## D.Spring配置元数据

Spring IoC容器与配置元数据的实际编写格式完全解耦。有以下三种重要方法可以向Spring容器提供配置元数据：

* 基于XML的配置文件
* 基于注释的配置
* 基于Java的配置

# 四、依赖注入（DI）

Spring最具代表性的技术是控制反转的依赖注入（DI）。控制反转是一个通用概念，它可以通过许多不同的方法来表达，而依赖注入只是控制反转的一个具体例子。

编写复杂的Java应用程序时，应用程序类应尽可能独立于其他Java类，以增加复用这些类的可能性，并在进行单元测试时独立于其他类对它们进行测试。依赖注入有助于将这些类粘合在一起，同时保持它们的独立性。依赖注入到底是什么？让我们分别看看这两个词。这里的依赖部分，转化为两个类之间的关系。例如，类A依赖于类B。现在，让我们看看第二部分，注入。这一切意味着类B将被Ioc注入到类A中。

依赖注入可以通过将参数传递给构造函数的方式进行。假设您有一个包含文本编辑器组件的应用程序，并且您想要提供拼写检查。您的标准代码将如下所示：

public class TextEditor{

privateSpellCheckerspellChecker;

publicTextEditor() {

spellChecker = new SpellChecker();

}

}

我们在这里所作的就是在TextEditor和SpellChecker之间创建依赖关系。在控制反转场景中，我们应该这样做：

public class TextEditor{

privateSpellCheckerspellChecker; publicTextEditor(SpellCheckerspellChecker) {

this.spellChecker = spellChecker;

}

}

在这里，TextEditor不必担心SpellChecker的实现。SpellChecker将被独立实现，并在TextEditor实例化时提供给它，这整个过程由Spring框架控制。我们将控制权从TextEditor中移除，并放在其他地方（即XML配置文件），通过类构造函数将依赖关系（即SpellChecker类）注入到TextEditor类中。因此，控制流通过依赖注入（DI）被“反转”，因为你有效地将依赖关系委托给了某个外部系统。

注入依赖项的第二种方法就是通过TextEditor类的Setter方法，我们将创建SpellChecker实例，并且该实例将用于调用setter方法来初始化TextEditor的属性。

依赖注入有几个重要的好处。例如：

* 由于组件不需要再运行时查找协作对象，因此编写和维护起来更简单。在Spring的Ioc实现中，组件将通过公开JavaEban的setter方法或构造函数参数来表达它们对其他组件的依赖。而EJB的等效实现是通过JNDI查找，这需要开发人员编写假设环境的代码。
* 出于同样的原因，应用程序代码更容易测试。例如，JavaBean属性简单、核心Java易于测试 ：只需要编写创建对象并设置相关属性的JUnit测试方法。
* 良好的Ioc实现保留了强类型。如果您需要使用通用工厂来查找协作对象，则必须将结果转换为所需的类型。这不是一个大问题，但它不够优雅。使用Ioc，您可以在代码中表达强类型依赖关系，而框架负责类型转换。这意味着当框架配置应用程序时，类型不匹配将作为错误引发；您不必担心代码中的类转换异常。
* 依赖关系是明确的。例如，如果一个应用程序类尝试在实例化时加载属性文件或连接到数据库，那么环境假设可能在不阅读代码的情况下并不明显（这会使测试复杂化并降低部署的灵活性）。通过依赖注入方法，依赖关系是明确的，并且在构造函数或JavaBean属性中显而易见。
* 大多数业务对象不依赖于IoC容器的API。这使得使用遗留代码变得容易，并且无论在IoC容器内部还是外部使用对象都很方便。例如，Spring用户经常将Jakarta Commons DBCP DataSource配置为Spring bean：无需编写任何自定义代码即可实现这一点。我们说IoC容器是非侵入性的：使用它不会使你的代码依赖于它的API。几乎任何POJO（Plain Old Java Object）都可以成为Spring bean工厂中的组件。现有的JavaBeans或具有多参数构造函数的对象特别适合，但Spring还提供了独特的支持，可以通过静态工厂方法甚至是由IoC容器管理的其他对象的方法来实例化对象。

依赖注入与传统的容器架构（如EJB）不同，它最大程度地减少了应用程序代码对容器的依赖。这意味着您的业务对象可以在不同的依赖注入框架中运行——或者在任何框架之外运行，而无需更改代码。尽管依赖注入并不是一个新概念，但它最近才在J2EE社区中占据主导地位。

有一些替代的依赖注入容器：特别是PicoContainer和HiveMind。PicoContainer特别轻量，并强调通过构造函数而不是JavaBean属性来表达依赖关系。它不使用Java代码之外的元数据，这使得它在功能上与Spring相比有所限制。HiveMind在概念上更类似于Spring（也不仅仅专注于Ioc），尽管它缺乏Spring项目的全面范围和相同规模的用户社区。EJB 3.0也将提供基本的依赖注入功能。

# 五、面向切面编程（AOP）

Spring的关键组件之一时面向切面编程（AOP）框架。跨越应用程序多个点的功能被称为横切关注点。这些横切关注点在概念上与应用程序的业务逻辑是分开的。

这里有很多常见的很好的例子，包括日志记录、声明式事务、安全性和缓存等方面。

面向对象编程（OOP）中的关键模块化单元是类，而面向切面编程（AOP）中的模块化单元是切面。依赖注入（DI）帮助你将应用程序对象彼此解耦，而AOP则帮助你将横切关注点和它们影响的对象解耦。Spring框架的AOP模块提供了面向切面编程的实现，允许你定义方法拦截器和切入点，从而干净地解耦那些应当分离的功能代码。Spring AOP模块提供了拦截器来拦截应用程序的操作，例如，当一个方法执行时，你可以在方法执行之前或之后添加额外的功能。

## A. AOP概念

* 切面（Aspect）：横切多个类的关注点的模块化。事务管理是J2EE应用程序中一个很好的横切关注点的例子。在Spring AOP中，切面使用常规类（基于模式的方法）或使用@Aspect注解的常规类（@AspectJ风格）实现。
* 连接点（Join point）：程序执行过程中的某个点，例如方法的执行或异常的处理。在Spring AOP中，连接点总是代表方法的执行。
* 通知（Advice）：切面在特定连接点采取的行动。不同类型的通知包括"环绕通知"、"前置通知"和"后置通知"。（通知类型将在下文讨论。）许多AOP框架，包括Spring，将通知建模为拦截器，在连接点周围维护一条拦截器链。
* 切入点（Pointcut）：匹配连接点的断言。通知与切入点表达式相关联，并在任何匹配切入点的连接点运行（例如，执行具有特定名称的方法）。通过切入点表达式匹配的连接点的概念是AOP的核心，Spring默认使用AspectJ切入点表达式语言。
* 引入（Introduction）：代表某个类型声明额外的方法或字段。Spring AOP允许你为任何被通知的对象引入新接口（以及相应的实现）。例如，你可以使用引入使一个bean实现IsModified接口，以简化缓存。（在AspectJ社区中，引入被称为类型间声明。）
* 目标对象（Target object）：被一个或多个切面通知的对象，也称为被通知对象。由于Spring AOP是使用运行时代理实现的，所以这个对象总是一个代理对象。
* AOP代理（AOP proxy）：由AOP框架创建的对象，以实现切面合同（通知方法执行等）。在Spring框架中，AOP代理将是一个JDK动态代理或CGLIB代理。
* 织入（Weaving）：将切面与其他应用程序类型或对象链接以创建被通知对象。这可以在编译时（例如，使用AspectJ编译器）、加载时或运行时完成。Spring AOP和其他纯Java AOP框架一样，在运行时执行织入。

# 六、Spring JDBC框架

在使用普通的旧方式处理数据库时，需编写不必要的代码来处理异常、打开和关闭数据库连接等，非常麻烦。但是Spring JDBC框架负责处理所有的低级细节，从打开连接、准备和执行SQL语句、处理异常、处理事务到最后关闭连接。

因此，你要做的只是定义连接参数并指定要执行的SQL语句，并从数据库获取数据时为每次迭代执行所需的工作。

Spring JDBC提供了几种方法和相应的不同类来与数据库交互。我将采用最经典和最流行的方法，即利用框架的JdbcTemplate 类。这是管理所有数据库通信和异常处理 的核心框架类。JdbcTemplate类执行SQL查询、更新语句和存储过程调用，对 ResultSet执行迭代并提取返回的参数值。它还捕获JDBC异常并将其转换为org.springframework.dao包中定义的通用的、更具信息量的异常层次结构。

JdbcTemplate类的实例一旦配置完成，就是线程安全的。因此，您可以配置JdbcTemplate的单个实例，然后将此共享引用安全地注入到多个DAO中。使用JdbcTemplate类的常见做法是在Spring配置文件中配置DataSource，然后将该共享DataSource bean依赖注入到您的DAO类中，然后在DataSource的setter中创 建JdbcTemplate。[2]

## A. 数据访问对象（DAO）

DAO代表数据访问对象，通常用于数据库交互。DAO的存在是为了提供一种读取和写入数据库数据的方法，它们应该通过一个接口公开此功能，应用程序的其余部分将通过该接口访问他们。Spring中的数据访问对象（DAO）支持简单地以一致的方式使用JDBC、Hibernate、JAP或JDO等数据访问技术。

## B. 事务管理

数据库事务是一系列被视为一个工作单元的操作。这些操作要么完全完成，要么完全不生效。事务管理是面向RDBMS（关系数据库管理系统）的企业应用程序中重要的一部分，以确保数据的完整性和一致性。

Spring框架在不同的底层事务管理API之上提供了一个抽象层。Spring的事务支持旨在通过为POJOs（普通Java对象）添加事务功能，提供EJB（企业Java Bean）事务的替代方案。Spring支持编程式和声明式的事务管理。EJB需要应用服务器，而Spring的事务管理可以在不需要应用服务器的情况下实现。

* 本地事务特定于单个事务资源，例如JDBC连接，而全局事务可以跨多个事务资源，如分布式系统中的事务。本地事务管理在集中式计算环境中非常有用，在这种环境中，应用程序组件和资源位于单一站点，事务管理仅涉及在单台机器上运行的本地数据管理器。本地事务更容易实现。
* 全局事务管理在分布式计算环境中是必需的，在这种环境中，所有资源分布在多个系统中。在这种情况下，事务管理需要在本地和全局层面进行。分布式或全局事务跨多个系统执行，其执行需要全局事务管理系统与所有相关系统的本地数据管理器之间的协调。

Spring支持两种类型的事务管理：

* 编程式事务管理：这意味着您通过编程来管理事务。这为您提供了极大的灵活性，但很难维护。
* 声明式事务管理：这意味着您将事务管理与业务代码分开。您只需注释或基于XML的配置来管理事务。

声明式事务管理比编程式事务管理更可取，尽管它不如编程式事务管理灵活，后者允许您通过代码控制事务。但作为一种横切关注点，声明式事务管理可以使用 AOP 方法进行模块化。 Spring通过Spring AOP框架支持声明式事务管理。

# 七、O/R 映射集成

当然，通常你会希望使用对象关系（O/R）映射，而不是使用关系型数据访问。你的整体应用框架也必须支持这一点。因此，Spring开箱即用地与Hibernate（2和3版本）、JDO（1和2版本）、TopLink以及其他ORM产品集成。它的数据访问架构允许它与任何底层数据访问技术集成。Spring和Hibernate是特别受欢迎的组合。

为什么你会使用ORM产品加上Spring，而不是直接ORM产品？Spring在以下几个方面增加了显著的价值：

* 会话管理。Spring提供了高效、简便且安全的工作单元处理，例如Hibernate或TopLink会话。单独使用ORM工具的相关代码通常需要使用相同的“会话”对象，以确保效率和正确的事务处理。Spring可以透明地创建并绑定一个会话到当前线程，使用声明式的AOP方法拦截器方法，或者在Java代码层面使用显式的“模板”包装类。因此，Spring解决了许多影响ORM技术用户的使用问题。
* 资源管理。Spring应用上下文可以处理Hibernate SessionFactory、JDBC数据源以及其他相关资源的定位和配置。这使得这些值易于管理和更改。
* 集成的事务管理。Spring允许你使用声明式的AOP方法拦截器，或者在Java代码层面使用显式的“模板”包装类来包装你的ORM代码。在这两种情况下，事务语义都由Spring处理，并且在发生异常的情况下，会处理适当的事务处理（回滚等）。如我们稍后讨论的，你还可以享受到使用和交换各种事务管理器的好处，而不会影响到你的ORM相关代码。作为一个额外的好处，在大多数支持的ORM工具中，JDBC相关代码可以与ORM代码在事务上完全集成。这对于处理不适合ORM的功能非常有用。
* 异常包装。Spring可以将ORM层的异常包装起来，将它们从专有（可能是受检的）异常转换为一组抽象的运行时异常。这使你能够在适当的层中处理大多数持久化异常（这些异常通常是不可恢复的），无需烦人的样板捕获/抛出和异常声明。当然，你仍然可以在需要的地方捕获和处理异常。请记住，JDBC异常（包括数据库特定的方言）也被转换为相同的层次结构，这意味着你可以在一致的编程模型中使用JDBC执行某些操作。
* 避免供应商锁定。ORM解决方案在性能和其他特性上有所不同，没有一种万能的解决方案。或者，你可能会发现某些功能并不适合使用你的ORM工具实现。因此，将你的架构与数据访问对象接口的工具特定实现解耦是有意义的。如果你将来因为功能、性能或其他任何问题需要切换到另一个实现，现在使用Spring可以使最终的切换更加容易。Spring对ORM工具的事务和异常的抽象，以及它的IoC方法，使你可以轻松地交换实现数据访问功能的映射器/DAO对象，从而使你能够在应用程序的一个区域中隔离所有特定于ORM的代码，而不牺牲ORM工具的任何功能。Spring附带的PetClinic示例应用程序通过提供使用JDBC、Hibernate、TopLink和Apache OJB实现持久层的不同版本，展示了Spring提供的可移植性优势。
* 测试的简便性。Spring的控制反转方法使得交换Hibernate会话工厂、数据源、事务管理器和映射器对象实现（如果需要）的实现和位置变得容易。这使得在隔离的环境中测试每一部分持久化相关代码变得更加简单。

最重要的是，Spring促进了数据访问的混合搭配方法。尽管一些ORM供应商声称ORM是解决所有问题的方案，但实际上它在许多情况下虽然是一个有价值的生产力提升，却并非万能。

Spring实现了一种一致的架构和事务策略，即使你混合和匹配持久化方法，也能保持一致，即使不使用JTA也是如此。抽象一个数据访问API是不够的，我们还需要考虑事务管理。JTA是显而易见的解决方案，但它是一个直接使用起来很繁琐的API，因此许多J2EE开发人员过去认为EJB CMT是事务管理的唯一合理选择。Spring改变了这一现状。

Spring的事务抽象是独特的，因为它不仅限于JTA或任何其他事务管理技术。Spring使用事务策略的概念，将应用程序代码与底层事务基础设施（如JDBC）解耦。

为什么你需要关心这个？JTA不是所有事务管理的最佳答案吗？如果你编写的应用程序仅使用单个数据库，你不需要JTA的复杂性。你不需要关心XA事务或两阶段提交。你甚至可能不需要提供这些功能的高端应用服务器。但是，另一方面，你不希望在需要使用多个数据源时重写代码。

想象一下，你决定通过直接使用JDBC或Hibernate事务来避免JTA的开销。如果你需要处理多个数据源，你将不得不删除所有事务管理代码，并用JTA事务替换它们。这并不吸引人，这导致大多数J2EE作者建议专门使用全局JTA事务，从而有效地排除了使用像Tomcat这样的简单Web容器来处理事务应用程序的可能性。然而，使用Spring的事务抽象，你只需重新配置Spring以使用JTA而不是JDBC或Hibernate事务策略，就可以完成。这是一个配置更改，而不是代码更改。因此，Spring使你能够编写既可以向下扩展也可以向上扩展的应用程序。

# 八、Spring Web MVC框架

Spring Web MVC框架提供了模型-视图-控制器（MVC）架构和现成的组件，可用于开发灵活且松散耦合的Web应用程序。MVC模式将应用程序的不同方面（输入逻辑、业务逻辑和UI逻辑）分离开来，同时在这些元素之间提供松散的耦合。

* 模型（Model） 封装了应用程序数据，通常由POJO（简单旧Java对象）组成。
* 视图（View） 负责渲染模型数据，通常会生成客户端浏览器可以解释的HTML输出。
* 控制器（Controller） 负责处理用户请求，构建适当的模型并将其传递给视图进行渲染。

## A. Dispatcher Servlet

Spring Web 模型‑视图‑控制器 (MVC) 框架是围绕处理所有HTTP请求和响应的DispatcherServlet 设计的。Spring Web MVC DispatcherServlet 的请求处理工作流程如下图所示：

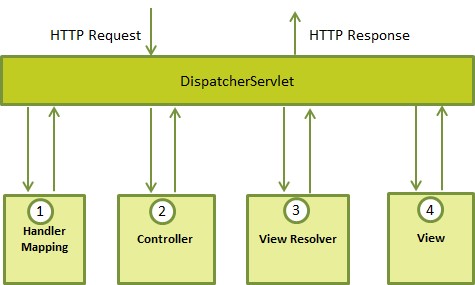


图3. Spring MVC DispatcherServlet 的请求处理流程

以下是与传入DispatcherServlet的HTTP请求相对应的事件序列：

* 在接收到HTTP请求后，DispatcherServlet会咨询HandlerMapping以调用适当的控制器（Controller）。
* 控制器（Controller）接收请求并根据使用的GET或POST方法调用相应的服务方法。服务方法将根据定义的业务逻辑设置模型数据并返回视图名称给DispatcherServlet。
* DispatcherServlet将借助ViewResolver选择定义的视图。
* 一旦视图确定，DispatcherServlet会将模型数据传递给视图，最终在浏览器上呈现。

以上提到的所有组件，即HandlerMapping、Controller和ViewResolver，都是WebApplicationContext的一部分，它是ApplicationContext的扩展，具有一些Web应用程序所需的额外功能。你需要在web.xml文件中使用URL映射来映射希望由DispatcherServlet处理的请求。

定义控制器 - DispatcherServlet将请求委派给控制器，以执行特定的功能。@Controller注解表示某个类充当控制器的角色。@RequestMapping注解用于将URL映射到整个类或特定的处理方法。@Controller注解将该类定义为Spring MVC控制器。

创建JSP视图 - Spring MVC支持多种类型的视图以适应不同的展示技术。这些包括：JSP、HTML、PDF、Excel工作表、XML、Velocity模板、XSLT、JSON、Atom和RSS feeds、JasperReports等。但最常用的是使用JSTL编写的JSP模板。

# 九、总结

Spring是一个强大的框架，解决了J2EE中的许多常见问题。许多Spring的功能不仅适用于经典的J2EE环境，还可用于各种Java环境。

Spring提供了一种一致的方式来管理业务对象，并鼓励使用编程接口而不是类的良好实践。Spring的架构基础是一个基于JavaBean属性的控制反转（IoC）容器。然而，这只是整体的一部分：Spring的独特之处在于，它将其IoC容器作为综合解决方案的基本构建块，涵盖所有架构层次。

Spring提供了独特的数据访问抽象，包括一个简单且高效的JDBC框架，大大提高了生产力并减少了错误的可能性。Spring的数据访问架构还与TopLink、Hibernate、JDO和其他O/R映射解决方案集成。

Spring还提供了独特的事务管理抽象，使得在各种底层事务技术（如JTA或JDBC）上实现一致的编程模型成为可能。

Spring提供了一个用标准Java编写的AOP框架，允许将声明性事务管理和其他企业服务应用于POJO，或者根据需要实现自定义切面。这个框架足够强大，能够使许多应用程序摆脱EJB的复杂性，同时享受传统上与EJB相关的关键服务。

# 参考

1. http://www.springsource.org/tutorial
2. http://www.tutorialspoint.com/spring/index.htm
3. http://en.wikipedia.org/wiki/Spring\_Framework
4. http://www.theserverside.com/news/1364527/Introduction-to-theSpring-Framework
5. http://www.theserverside.com/news/1363858/Introduction-to-theSpring-Framework
6. http://www.tutorialspoint.com/spring/spring\_dependency\_injection.ht m
7. Seth Ladd, Darren Davison, Steven Devijver and Colin Yates,

“Expert Spring MVC and Web Flow”

1. Gary Mak , “Spring Recipes”