

本科毕业论文（设计）

英文文献翻译

|  |  |
| --- | --- |
| **文献英文题目** | **The Spring Framework: An Open Source Java Platform for Developing Robust Java Applications** |
| **学 生 姓 名** | **谢仁义** |
| **学号** | **2016081187** |
| **专业** | **软件工程** |
| **年级班级** | **2016级5班** |
| **指导教师** | **王嘉昀（讲师）** |
| **所在学院** | **软件工程学院** |
| **提交日期** | **2020年4月15日** |

2020 年 4 月

成都信息工程大学 软件工程学院

Spring框架：用于开发健壮的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框架是一个开放源代码Java平台，为轻松，快速地开发强大的Java应用程序提供了全面的基础架构支持。 Spring框架是一个轻量级的解决方案，是一个潜在的一站式商店，用于构建企业就绪的应用程序。

关键词：面向方面的编程；依赖注入；IoC容器；ORM；

# 介绍

Spring是最流行的企业Java应用程序开发框架。全球数以百万计的开发人员使用Spring Framework来创建高性能，易于测试的可重用代码。 Spring框架是一个开放源Java平台，最初由Rod Johnson编写，并于2003年6月根据Apache 2.0许可证首次发布。

在大小和透明度方面，Spring是轻量级的。 Spring框架的基本版本约为2MB。Spring框架的核心功能可用于开发任何Java应用程序，但在Java EE平台上有一些用于构建Web应用程序的扩展。 Spring框架旨在通过启用基于POJO的编程模型来使J2EE开发更易于使用并促进良好的编程实践。 Spring框架可在任何类型的部署平台上为基于Java的现代企业应用程序提供全面的编程和配置模型。 Spring的一个关键元素是在应用程序级别的基础架构支持：Spring专注于企业应用程序的“管道”，以便团队可以专注于应用程序级别的业务逻辑，而不必与特定的部署环境建立不必要的联系。Spring包括：

* 灵活的依赖项注入以及基于XML注释的配置样式。
* 对基于方面的编程的高级支持，包括基于代理和基于AspectJ的变体。
* 对常见开源框架（例如Hibernate和Quartz）的一流支持。
* 用于构建RESTful MVC的灵活Web框架应用程序和服务端点。

Spring在设计上是模块化的，允许增量采用各个部分，例如核心容器或JDBC支持。 尽管所有Spring服务都非常适合Spring核心容器，但是许多服务也可以在容器外部以编程方式使用。

受支持的部署平台从独立应用程序到Tomcat和Java EE服务器（例如WebSphere）。 在Java等主要云平台上，Spring还是一流的公民。 在Heroku，Google App Engine，Amazon Elastic Beanstalk和VMware的Cloud Foundry上运行[1]。

# Spring框架架构

Spring可能是您所有企业应用程序的一站式服务。 但是，Spring是模块化的，因此您可以选择适合自己的模块，而不必引入其他模块。

Spring框架提供了大约20个模块，可以根据应用程序需求使用它们。

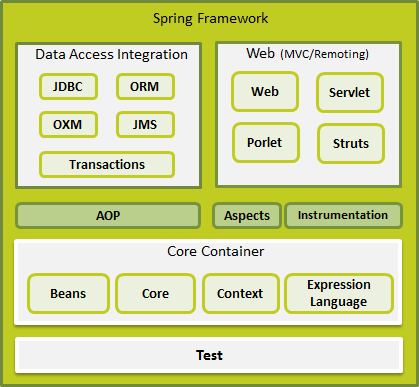


图1 Spring框架结构

A: 核心容器

核心容器由核心，Bean，上下文和表达式语言模块组成，其详细信息如下：

* 核心模块提供了框架的基本部分，包括IoC和依赖注入功能。
* Bean模块提供BeanFactory，这是工厂模式的复杂实现。
* 上下文模块建立在Core和Beans模块提供的坚实基础上，它是访问定义和配置的任何对象的媒介。 ApplicationContext接口是Context模块的焦点。
* 表达式语言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。

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（包括JPA，JDO，Hibernate和iBatis）提供了集成层。
* OXM模块提供了一个抽象层，该层支持JAXB，Castor，XMLBeans，JiBX和XStream的对象/ XML映射实现。
* Java Messaging Service JMS模块包含用于生成和使用消息的功能。
* 事务模块支持对实现特殊接口的类以及所有POJO进行编程和声明式事务管理。

C：网页

* 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容器进行初始化。
* Web-Servlet模块包含用于Web应用程序的Spring的模型视图控制器（MVC）实现。
* Web-Struts模块包含用于在Spring应用程序中集成经典Struts Web层的支持类。

D：其他

* AOP模块提供了面向方面的编程实现，使您可以定义方法拦截器和切入点，以清晰地解耦实现应分离功能的代码。
* Aspects模块提供了与AspectJ的集成，后者又是一个功能强大且成熟的面向方面的编程（AOP）框架。
* Instrumentation模块提供了在某些应用程序服务器中使用的类检测支持和类加载器实现。
* Test模块支持使用JUnit或TestNG框架测试Spring组件。

# 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，进行配置，并管理从创建到销毁的整个生命周期。 Spring容器使用依赖项注入（DI）来管理组成应用程序的组件。 这些对象称为Spring Bean，我们将在下一章中进行讨论。

容器通过读取提供的配置元数据来获取有关要实例化，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工作方式的高级视图。 Spring IoC容器利用Java POJO类和配置元数据来生成完全配置且可执行的系统或应用程序。

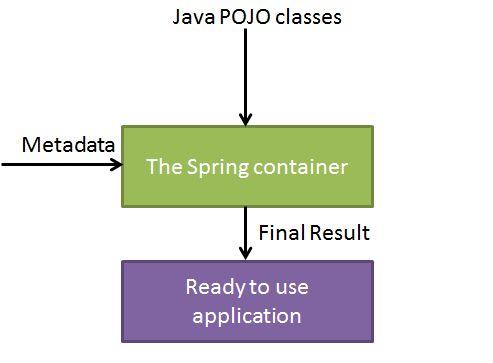


图2.Spring IOC 容器

Spring提供以下两种不同类型的容器。

A：Spring BeanFactory容器

这是为DI提供基本支持的最简单的容器。 为了向后兼容与Spring集成的大量第三方框架，BeanFactory和相关接口（例如BeanFactoryAware，InitializingBean，DisposableBean）仍存在于Spring中。

B：Spring ApplicationContext容器

此容器添加了更多特定于企业的功能，例如从属性文件解析文本消息的功能以及将应用程序事件发布到感兴趣的事件侦听器的功能。 该容器由org.springframework.context.ApplicationContext接口定义。

ApplicationContext容器包含BeanFactory容器的所有功能，因此通常建议在BeanFactory上使用它。 BeanFactory仍可用于轻量级应用程序，例如移动设备或基于applet的应用程序，这些应用程序的数据量和速度非常重要。

C：实体

构成应用程序主干并由Spring IoC容器管理的对象称为bean。 Bean是由Spring IoC容器实例化，组装和以其他方式管理的对象。 这些bean是使用您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配置注解

Spring IoC容器与实际写入此配置注解的格式完全挂钩。 有以下三种重要的方法可以为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

# 依赖注射（DI）

Spring最受瞩目的技术是控制反转的依赖注入（DI）风格。控制反转（IoC）是一个通用概念，可以用许多不同的方式表示，并且依赖注入只是控制反转的一个具体示例。

在编写复杂的Java应用程序时，应用程序类应尽可能独立于其他Java类，以增加重用这些类并在进行单元测试时独立于其他类进行测试的可能性。依赖注入有助于将这些类粘合在一起，同时保持它们独立。到底什么是依赖注入？让我们分别看这两个词。在这里，依赖关系部分转换为两个类之间的关联。例如，类A依赖于类B。现在，让我们看第二部分，注入。所有这些意味着B类将被IoC注入A类。

依赖注入可以通过将参数传递给构造函数的方式进行，也可以通过使用setter方法进行后构造的方式进行。考虑到您有一个具有文本编辑器组件的应用程序，并且希望提供拼写检查。您的标准代码如下所示：

1. **public** **class** TextEditor {
2. privateSpellCheckerspellChecker;
3. publicTextEditor() {
4. spellChecker = **new** SpellChecker();
5. }
6. }

我们在这里所做的是在TextEditor和SpellChecker之间创建一个依赖项。 在控制方案反转的情况下，我们将改为执行以下操作：

1. **public** **class** TextEditor {
2. privateSpellCheckerspellChecker;
3. publicTextEditor(SpellCheckerspellChecker)  {
4. **this**.spellChecker = spellChecker;
5. }
6. }

在这里，TextEditor不必担心SpellChecker的实现。 SpellChecker将独立实现，并将在TextEditor实例化时提供给TextEditor，整个过程由Spring框架控制。 我们已经从TextEditor中删除了总控件，并将其保留在其他位置（即XML配置文件），并且依赖项（即SpellChecker类）已通过类构造器注入到TextEditor类中。 因此，由于您已将依赖项有效地委派给了某些外部系统，因此控制流已被依赖项注入（DI）“反转”了。

注入依赖项的第二种方法是通过TextEditor类的Setter Methods，在此我们将创建SpellChecker实例，该实例将用于调用setter方法以初始化TextEditor的属性。 依赖注入有几个重要的好处。 例如：

* 因为组件不需要在运行时查找协作者，所以它们的编写和维护要简单得多。在Spring的IoC版本中，组件通过公开JavaBean setter方法或通过构造函数参数来表达对其他组件的依赖性。等效的EJB将是JNDI查找，这需要开发人员编写进行环境假设的代码。
* 出于相同的原因，应用程序代码更容易测试。例如，JavaBean属性是简单的Java核心，并且易于测试：只需编写一个独立的JUnit测试方法即可创建对象并设置相关属性。
* 良好的IoC实现可保留强类型。如果需要使用通用工厂来查找协作者，则必须将结果转换为所需的类型。这不是一个主要问题，但是它并不优雅。使用IoC，您可以在代码中表达强类型的依赖关系，并且框架负责类型转换。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强制转换异常。
* 依赖性是明确的。例如，如果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，则在不读取代码的情况下，环境假设可能不明显（复杂的测试并降低了部署灵活性）。使用依赖注入方法，依赖关系是显式的，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IoC容器API。这使得易于使用遗留代码，并易于在IoC容器内部或外部使用对象。例如，Spring用户经常将Jakarta Commons DBCP数据源配置为Spring Bean：无需编写任何自定义代码即可执行此操作。我们说IoC容器不是侵入性的：使用它不会依赖于其API入侵您的代码。几乎任何POJO都可以成为Spring bean工厂中的组件。现有的JavaBean或具有多参数构造函数的对象工作得特别好，但是Spring还提供了对从静态工厂方法甚至由IoC容器管理的其他对象上的方法实例化对象的独特支持。

在将应用程序代码对容器的依赖性最小化方面，依赖性注入与EJB之类的传统容器体系结构不同。 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，也可以在任何框架外运行而无需更改代码。

依赖注入并不是一个新概念，尽管它只是最近才在J2EE社区中占了上风。 还有其他DI容器：值得注意的是PicoContainer和HiveMind。 PicoContainer特别轻巧，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，与Spring相比，它限制了其功能。 尽管HiveMind缺乏Spring项目的全面范围或同等规模的用户社区，但它在概念上与Spring更相似（其目标还不仅仅是IoC）。 EJB 3.0也将提供基本的DI功能。

# 面向切面的编程（AOP）

Spring的关键组件之一是面向方面的编程（AOP）框架。跨应用程序多个点的功能称为跨领域关注点，这些跨领域关注点在概念上与应用程序的业务逻辑分离。有很多方面的通用示例，包括日志记录，声明性事务，安全性和缓存等。

OOP中模块化的关键单元是类，而在AOP中模块化是方面。 DI可以帮助您将应用程序对象彼此分离，而AOP可以帮助您将横切关注点与其影响的对象分离。 Spring Framework的AOP模块提供了面向方面的编程实现，使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。 Spring AOP模块提供了拦截器来拦截应用程序，例如，当方法执行时，您可以在方法执行之前或之后添加额外的功能[2]。

A：AOP概念

* 方面：关注点的模块化，跨多个类。事务管理是J2EE应用程序中横切关注点的一个很好的例子。在Spring AOP中，方面是使用常规类（基于架构的方法）或使用@Aspect注释（@AspectJstyle）进行注释的常规类来实现的。
* 连接点：程序执行过程中的一点，例如方法的执行或异常的处理。在Spring AOP中，连接点始终代表方法的执行。
* 建议：方面在特定的连接点处采取的操作。不同类型的建议包括“周围”，“之前”和“之后”建议。 （建议类型将在下面讨论。）包括Spring在内的许多AOP框架都将建议建模为拦截器，并在连接点周围维护了一系列拦截器。
* 切入点：与连接点匹配的谓词。建议与切入点表达式关联，并在与该切入点匹配的任何连接点处运行（例如，执行具有特定名称的方法）。切入点表达式匹配的连接点的概念是AOP的核心，并且Spring默认使用AspectJpointcut表达式语言。简介：代表类型声明其他方法或字段。 Spring AOP允许您向任何建议对象引入新接口（和相应的实现）。例如，您可以使用简介使Bean实现IsModified接口，以简化缓存。 （在AspectJ社区中，介绍被称为类型间声明。）
* 目标对象：一个或多个方面建议的对象。也称为建议对象。由于Spring AOP是使用运行时代理实现的，因此该对象将始终是代理对象。
* AOP代理：由AOP框架创建的一个对象，用于实现方面协定（建议方法执行等）。在Spring框架中，AOP代理将是JDK动态代理或CGLIB代理。
* 编织：将方面与其他应用程序类型或对象链接以创建建议的对象。这可以在编译时（例如，使用AspectJ编译器），加载时或在运行时完成。像其他纯Java AOP框架一样，Spring AOP在运行时执行编织。

# Spring JDBC 框架

在使用普通的旧JDBC处理数据库时，编写不必要的代码来处理异常，打开和关闭数据库连接等变得很麻烦。但是Spring JDBC Framework会处理从打开连接，准备和启动开始的所有低级细节。

执行SQL语句，处理异常，处理事务并最终关闭连接。

因此，您要做的就是定义连接参数并指定要执行的SQL语句，并在每次从数据库中获取数据时进行每次迭代所需的工作。

Spring JDBC提供了几种方法以及与数据库相对应的不同类。我将采用经典且最受欢迎的方法，该方法利用了框架的JdbcTemplate类。这是管理所有数据库通信和异常处理的中央框架类。

JdbcTemplateclass执行SQL查询，更新语句和存储过程调用，对ResultSet执行迭代以及提取返回的参数值。它还捕获JDBC异常，并将其转换为org.springframework.dao包中定义的通用，信息量更大的异常层次结构。

一旦配置，JdbcTemplate类的实例是线程安全的。因此，您可以配置JdbcTemplate的单个实例，然后安全地将此共享引用注入到多个DAO中。使用JdbcTemplate类的常见做法是在Spring配置文件中配置DataSource，然后将共享的DataSource bean依赖注入到您的DAO类，并在设置器中为DataSource创建JdbcTemplate。

A：数据访问对象（DAO）

DAO代表通常用于数据库交互的数据访问对象。 DAO的存在是为了提供一种向数据库读取和写入数据的方式，它们应该通过接口公开此功能，其余的应用程序将通过该接口访问它们。 Spring中的数据访问对象（DAO）支持使以一致的方式轻松使用诸如JDBC，Hibernate，JPA或JDO之类的数据访问技术。

B：交易管理

数据库事务是被视为单个工作单元的一系列操作。 这些动作应完全完成或完全不起作用。 事务管理是面向RDBMS的企业应用程序的重要组成部分，可确保数据完整性和一致性。

Spring框架在不同的基础事务管理API之上提供了一个抽象层。 Spring的事务支持旨在通过向POJO添加事务功能来提供EJB（企业Java Bean）事务的替代方案。 Spring支持程序化和声明式事务管理。 EJB需要一个应用程序服务器，但是可以在不需要应用程序服务器的情况下实现Spring事务管理。

* 本地事务特定于单个事务资源（例如JDBC连接），而全局事务可以跨越多个事务资源，例如分布式系统中的事务。 在应用程序组件和资源位于单个站点的集中式计算环境中，本地事务管理非常有用，而事务管理仅涉及在单个计算机上运行的本地数据管理器。 本地交易更容易实现。
* 在所有资源跨多个系统分布的分布式计算环境中，需要全局事务管理。 在这种情况下，需要在本地和全局级别上进行事务管理。 分布式或全局事务在多个系统上执行，并且其执行需要全局事务管理系统与所有相关系统的所有本地数据管理器之间的协调。

Spring支持两种类型的事务管理：

* 程序化交易管理：这意味着您已经在编程的帮助下管理了交易。 这为您提供了极大的灵活性，但是很难维护。
* 声明式事务管理：这意味着您将事务管理与业务代码分开。 您仅使用注释或基于XML的配置来管理事务。

声明性事务管理比程序性事务管理更可取，尽管它不如程序性事务管理灵活，后者允许您通过代码控制事务。 但是作为一种横切关注点，可以使用AOP方法将声明式事务管理模块化。 Spring通过Spring AOP框架支持声明式事务管理。

# O / R映射集成

当然，通常您想使用O / R（对象关系）映射，而不是使用关系数据访问。 您的整个应用程序框架也必须支持这一点。 因此，Spring可以与Hibernate（版本2和3），JDO（版本1和2），TopLink和其他ORM产品集成在一起。 它的数据访问体系结构使其可以与任何基础数据访问技术集成。 Spring和Hibernate是一个特别受欢迎的组合。

为什么要使用ORM产品加上Spring，而不是直接使用ORM产品？ Spring在以下方面增加了重要的价值：

* 会话管理。 Spring为工作单元（例如Hibernate或TopLink Session）提供了高效，便捷和安全的处理方式。单独使用ORM工具的相关代码通常需要使用相同的“ Session”对象，以提高效率和正确进行事务处理。 Spring可以使用声明性AOP方法拦截器方法，或者在Java代码级别使用显式的“模板”包装器类，透明地创建会话并将其绑定到当前线程。因此，Spring解决了许多影响许多ORM技术用户的使用问题。
* 资源管理。 Spring应用程序上下文可以处理Hibernate SessionFactories，JDBC数据源和其他相关资源的位置和配置。这使得这些值易于管理和更改。
* 集成交易管理。 Spring允许您在Java代码级别使用声明性AOP方法拦截器或显式“模板”包装器类包装ORM代码。在这两种情况下，都将为您处理事务语义，并妥善处理异常情况下的事务处理（回滚等）。正如我们稍后讨论的那样，您还将获得能够使用和交换各种事务管理器的好处，而不会影响与ORM相关的代码。另外一个好处是，在大多数受支持的ORM工具的情况下，与JDBC相关的代码可以与ORM代码完全事务集成。这对于处理ORM不支持的功能很有用。
* 异常包装。 Spring可以包装来自ORM层的异常，将其从专有的（可能是经过检查的）异常转换为一组抽象的运行时异常。这使您可以仅在适当的层中处理大多数不可恢复的持久性异常，而不会烦人样板捕获/抛出和异常声明。您仍然可以在任何需要的地方捕获和处理异常。请记住，JDBC异常（包括特定于DB的方言）也将转换为相同的层次结构，这意味着您可以在一致的编程模型中使用JDBC执行某些操作。
* 为了避免供应商锁定。 ORM解决方案具有不同的性能和其他特性，并且没有完美的尺寸适合所有解决方案。另外，您可能会发现某些功能不适合使用ORM工具的实现。因此，将架构与数据访问对象接口的特定于工具的实现脱钩是有意义的。如果出于功能，性能或任何其他方面的考虑而可能需要切换到另一个实现，那么现在使用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，以及其IoC方法使您可以轻松地交换实现数据访问功能的mapper / DAO对象，从而轻松地将所有特定于ORM的代码隔离在应用程序的一个区域中牺牲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来实现持久层的变体，展示了Spring提供的可移植性优点。
* 易于测试。 Spring的控制方法反转使交换Hibernate会话工厂，数据源，事务管理器和mapper对象实现（如需要）等资源的实现和位置变得容易。这使得隔离和测试每个与持久性相关的代码段变得更加容易。

最重要的是，Spring促进了数据访问的混合匹配方法。尽管有些ORM供应商宣称，ORM并不是解决所有问题的方法，尽管在许多情况下ORM可以极大地提高生产率。即使您混合并匹配了持久性方法，即使不使用JTA，Spring都可以实现一致的体系结构和事务策略。

仅提取数据访问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，但是它是直接使用的繁琐API，因此许多J2EE开发人员曾经认为EJB CMT是事务管理的唯一合理选择。春天改变了这一点。

Spring的事务抽象是独特的，因为它与JTA或任何其他事务管理技术无关。 Spring使用事务策略的概念，该策略将应用程序代码与底层事务基础结构（例如JDBC）分离。

你为什么要关心这个？ JTA是否是所有事务管理的最佳答案？如果要编写仅使用单个数据库的应用程序，则不需要JTA的复杂性。您对XA事务或两阶段提交不感兴趣。您甚至可能不需要提供这些功能的高端应用程序服务器。但是，另一方面，如果您必须使用多个数据源，则不需要重写代码。

想象一下，您决定通过直接使用JDBC或Hibernate事务来避免JTA的开销。如果您需要使用多个数据源，则必须删除所有的事务管理代码，然后将其替换为JTA事务。这并不是很吸引人，因此大多数J2EE的作者建议仅使用全局JTA事务，从而有效地排除了使用简单的Web容器（例如Tomcat）进行事务处理的情况。但是，使用Spring事务抽象，只需要将Spring重新配置为使用JTA，而不是JDBC或Hibernate事务策略，就可以了。这是配置更改，而不是代码更改。因此，Spring使您能够编写可以按比例缩放的应用程序。

# Spring网络MVC框架

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

* 该模型封装了应用程序数据，通常它们将由POJO组成。
* 视图负责呈现模型数据，并且通常会生成客户端浏览器可以解释的HTML输出。
* 控制器负责处理用户请求并构建适当的模型，并将其传递给视图以进行渲染。

A：分派器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。 下图说明了Spring Web MVCDispatcherServlet的请求处理工作流：

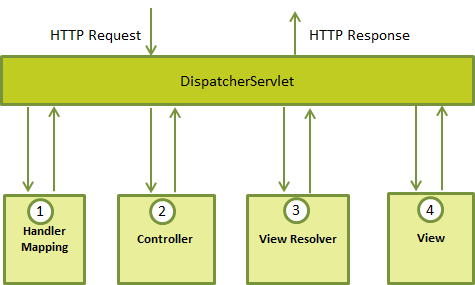


图3. Spring Web MVC DispatcherServlet的请求处理工作流程

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

* 收到HTTP请求后，DispatcherServlet进行咨询
* HandlerMapping调用适当的控制器。 控制器接收请求并基于使用的GET或POST方法调用适当的服务方法。 服务方法将基于定义的业务逻辑设置模型数据，并将视图名称返回给DispatcherServlet。
* DispatcherServlet将获得ViewResolver的帮助，以获取请求的已定义视图。
* 视图完成后，DispatcherServlet将模型数据传递到视图，该视图最终在浏览器中呈现[2]。

上述所有组件，即HandlerMapping，Controller和ViewResolver都是WebApplicationContext的一部分，它是普通ApplicationContext的扩展，具有Web应用程序必需的一些额外功能。您需要使用URL映射来映射希望DispatcherServlet处理的请求。在web.xml文件中。

定义控制器-DispatcherServlet将请求委托给控制器以执行特定于其的功能。 @Controller注释指示特定的类充当控制器的角色。 @RequestMapping批注用于将URL映射到整个类或特定的处理程序方法。 @Controller注释将类定义为Spring MVC控制器。

创建JSP视图-Spring MVC支持多种类型的视图用于不同的表示技术。这些包括-JSP，HTML，PDF，Excel工作表，XML，Velocity模板，XSLT，JSON，Atom和RSS提要，JasperReports等。但是最常见的是，我们使用用JSTL编写的JSP模板。

# 结论

Spring是一个功能强大的框架，可以解决J2EE中的许多常见问题。除了经典的J2EE，许多Spring功能还可以在广泛的Java环境中使用。

Spring提供了一种一致的方式来管理业务对象，并鼓励良好的实践，例如对接口进行编程，而不是对类进行编程。 Spring的体系结构基础是基于JavaBean属性的使用的Inversion of Control容器。但是，这只是整体情况的一部分：Spring的独特之处在于，它将IoC容器用作解决所有体系结构层的全面解决方案中的基本构建块。

Spring提供了独特的数据访问抽象，包括一个简单而高效的JDBC框架，该框架极大地提高了生产率并减少了出错的可能性。 Spring的数据访问体系结构还与TopLink，Hibernate，JDO和其他O / R映射解决方案集成。

Spring还提供了一个独特的事务管理抽象，它可以在各种基础事务技术（例如JTA或JDBC）上实现一致的编程模型。

Spring提供了一个用标准Java编写的AOP框架，该框架提供了声明式事务管理和其他企业服务，可应用于POJO，或者（如果需要）实现自己的自定义方面的能力。这个框架足够强大，可以使许多应用程序免除EJB的复杂性，同时享受传统上与EJB相关的关键服务。

参考文献

1. http://www.springsource.org/tutoial
2. <https://www.tutorialspoint.com/spring/index.htm>
3. <https://en.wikipedia.org/wiki/Spring_Framework>
4. <http://www.theserverside.com/news/1364527/Introduction-to-the->Spring-FrameWork
5. <http://www.theserverside.com/news/1363858/Introduction-to-the-> Spring-FrameWork
6. <https://www.tutorialspoint.com/spring/spring_dependency_injection.htm>
7. Seth Ladd, Darren Davison, Steven Devijver and Colin Yates, “Expert Spring MVC and Web Flow”
8. Gary Mak , “Spring Recipes”

**附英文原文：**

