Spring技术指引

# Spring Framework介绍

Spring 是一个开源框架，是为了解决企业应用程序开发复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许您选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。

在这篇由三部分组成的 Spring 系列 的第 1 部分中，我将介绍 Spring 框架。我先从框架底层模型的角度描述该框架的功能，然后将讨论两个最有趣的模块：Spring 面向方面编程（AOP）和控制反转 （IOC） 容器。接着将使用几个示例演示 IOC 容器在典型应用程序用例场景中的应用情况。这些示例还将成为本系列后面部分进行的展开式讨论的基础，在本文的后面部分，将介绍 Spring 框架通过 Spring AOP 实现 AOP 构造的方式。

Spring 框架是一个分层架构，由 7 个定义良好的模块组成。Spring 模块构建在核心容器之上，核心容器定义了创建、配置和管理 bean 的方式，如图 1 所示。



图 1. Spring 框架的 7 个模块

组成 Spring 框架的每个模块（或组件）都可以单独存在，或者与其他一个或多个模块联合实现。每个模块的功能如下：

核心容器：核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转 （IOC） 模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。

Spring 上下文：Spring 上下文是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。

Spring AOP：通过配置管理特性，Spring AOP 模块直接将面向方面的编程功能集成到了 Spring 框架中。所以，可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

Spring DAO：JDBC DAO 抽象层提供了有意义的异常层次结构，可用该结构来管理异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量（例如打开和关闭连接）。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。

Spring ORM：Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具，其中包括 JDO、Hibernate 和 iBatis SQL Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。

Spring Web 模块：Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文。所以，Spring 框架支持与 Jakarta Struts 的集成。Web 模块还简化了处理多部分请求以及将请求参数绑定到域对象的工作。

Spring MVC 框架：MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

Spring 框架的功能可以用在任何 J2EE 服务器中，大多数功能也适用于不受管理的环境。Spring 的核心要点是：支持不绑定到特定 J2EE 服务的可重用业务和数据访问对象。毫无疑问，这样的对象可以在不同 J2EE 环境 （Web 或 EJB）、独立应用程序、测试环境之间重用。

包说明

|  |  |
| --- | --- |
| org.springframework.aop | 包含在应用中使用Spring的AOP特性时所需的类。 |
| org.springframework.asm | Spring独立的asm程序，Spring2.5.6的时候需要 asm.jar 包，3.0开始提供自己独立的asm.jar |
| org.springframework.aspects | 提供对AspectJ的支持，以方便的将面向切面的功能集成进IDE中，比如Eclipse AJDT. |
| Org.springframework.beans | 所有应用都要用到，它包含访问配置文件，创建和管理Bean以及进行反转共控制和依赖注入操作时所需的类 |
| Org.springframework.context | 为 Spring 核心提供了大量扩展。可以找到使用Spring ApplicationContext 特性时所需的全部类，JDNI所需的全部类，UI方面的用来与模板（Templating）引擎如 Velocity、 FreeMarker、 JasperReports 集成的类， 以及校验 Validation 方面的相关类。 |
| org.springframework.context.support | 包含支持缓存Cache（ehcache）JCA、JMX、邮件服务（Java Mail、Mail）COS、任务计划Scheduling（Timer、Quartz）方面的类。 |
| org.springframework.core | 包含 Spring 框架基本的核心工具类，Spring 其它组件要都要使用到这个包里的 类，是其它组件的基本核心。 |
| org.springframework.expression | 包含Spring 表达式语言相关的类，应用使用到SpEL时需要添加这个jar包。 |
| org.springframework.instrument.tomcat | Spring3.0对Tomcat的连接池的集成 |
| org.springframework.instrument | Spring3.0 对服务器的代理接口。 |
| org.springframework.jdbc | 包含对 Spring 对 JDBC 数据访问进行封装的所有类。 |
| org.springframework.jms | 提供了对JMS1.0.2/1.1 的支持类。 |
| org.springframework.orm | 包含Spring对DAO特性集进行了扩展，使其支持iBATIS、JDO、OJB、TopLink，因为Hibernate已经独立成包了，现在不包含在这个包里了。这个jar文件里大部分的类都要依赖spring-dao.jar里的类，用这个包时你需要同时包含 spring-dao.jar包。 |
| org.springframework.oxm | Spring 对 Object/XMl 的映射支持,可以让Java与XML之间来回切换。 |
| org.springframework.test | 对Junit等测试框架的简单封装。 |
| org.springframework.transaction | 为JDBC、Hibernate、JDO、JPA等提供的一致的声明式和编程式事务管理。 |
| org.springframework.web | 包含Web应用开发时，用到Spring框架时所需的核心类，包括自动载入WebApplicationContext特性的类、Struts与JSF集成类、文件上传的支持类、Filter类和大量工具辅助类。 |
| org.springframework.web.portlet | 对SpringMVC的增强，包含SpringMVC相关的类 |
| org.springframework.web.servlet | 对J2EE6.0的Servlet3.0的支持。 |
| org.springframework.web.struts | 对Struts框架支持，可以更方便更容易的集成Struts框架。 |

## 3.x版本特性

二、 增加的新特性：

1、 Spring Expression Language （Spring表达式语言）

2、 IoC enhancements/Java based bean metadata （增强的IOC，基于java的Bean元数据）

3、 General-purpose type conversion system and field formatting system （通用类型转换系统和字段格式化系统）

4、 Object to XML mapping functionality (OXM) moved from Spring Web Services project （对象和XML之间的映射，来自于Srping Web Services项目）

5、 Comprehensive REST support （广泛的REST支持）

6、 @MVC additions （Spring的MVC注解增强）

7、 Declarative model validation （声明试模型验证）

8、 Early support for Java EE 6 （提前对JEE6的支持）

9、 Embedded database support （嵌入式数据库的支持）

关于新特性，下面会有详细说明。

三、 API更新：

1、 BeanFactory接口尽可能的返回明确的实例，例如：

T getBean(Class<T> requiredType)

T getBean(String name, Class<T> requiredType)

Map<String, T> getBeansOfType(Class<T> type)

这样我们在通过容器获取bean时就可以使用泛型，而不必进行强制转化，另外一个方面又可以根据bean的类型来获取Bean的实例。

2、 Spring的TaskExecutor现在继承java.util.concurrent.Executor接口。

扩展的子接口 AsyncTaskExecutor 支持标准的具有返回结果 Futures 的 Callables.

3、 新的基于 Java5 的 API 和 SPI 转换器

无状态的 ConversionService 和 Converters

取代标准的 JDK PropertyEditors

4、 类型化的 ApplicationListener<E>,

这是一个实现"观察者设计模式"使用的事件监听器. 基于事件的编程模式, 好处多多, 在项目中应该考虑使用, 基于事件,状态迁移的设计思路, 有助于理清软件流程,和减少项目的耦合度.

四、 Spring Expression Language (Spring 表达式语言，简称SpEL)

Spring3.0引入了一种新的表达式语言：SpEL. SpEL使得java具有动态语言的风格，可以在运行时获取表达式的值。依赖注入时可以使用SpEL表达式语言, 不再只是原来简单的属性文件变量。xml配置

文件和java注解都支持SpEL，eg:

配置文件：

Xml代码 收藏代码

<bean class="mycompany.RewardsTestDatabase">

<property name="databaseName" value="#{systemProperties.databaseName}"/>

<property name="keyGenerator" value="#

{strategyBean.databaseKeyGenerator}"/>

</bean>

注解：

Java代码 收藏代码

@Repository

public class RewardsTestDatabase {

@Value("#{systemProperties.databaseName}")

public void setDatabaseName(String dbName){ … }

@Value("#{strategyBean.databaseKeyGenerator}")

public void setKeyGenerator(KeyGenerator kg) { … }

}

五、 IoC enhancements/Java based bean metadata（增强的IOC，基于java的bean配置）

增强的IOC，配置Bean时除了原来的XML和Annotation，现在支持一种java类和元数据结合的配置方式。新增注解：

@Configuration、@Bean、@DependsOn、@Primary、@Lazy、@Import、@ImportResource、@Value。eg:

java类配置文件：

Java代码 收藏代码

package org.example.config;

@Configuration

public class AppConfig {

private @Value("#{jdbcProperties.url}") String jdbcUrl;

private @Value("#{jdbcProperties.username}") String username;

private @Value("#{jdbcProperties.password}") String password;

@Bean

public FooService fooService() {

return new FooServiceImpl(fooRepository());

}

@Bean

public FooRepository fooRepository() {

return new HibernateFooRepository(sessionFactory());

}

@Bean

public SessionFactory sessionFactory() {

// wire up a session factory

AnnotationSessionFactoryBean asFactoryBean = new AnnotationSessionFactoryBean();

asFactoryBean.setDataSource(dataSource());

// additional config

return asFactoryBean.getObject();

}

@Bean

public DataSource dataSource() {

return new DriverManagerDataSource(jdbcUrl, username, password);

}

}

为了让这段代码开始生效，我们需要添加如下组件应用程序上下文配置文件中，

xml配置文件：

Xml代码 收藏代码

<context:component-scan base-package="org.example.config"/>

<util:properties id="jdbcProperties" location="classpath:org/example/config/jdbc.properties"/>

xml配置文件主要用于指定java类配置文件和引用到的属性文件的位置。这个文件是可选的，当时使用原来的ClassPathXMLApplicationContext等获取BeanFactory时，

这个文件是必须的。或者直接使用AnnotationConfigApplicationContext时,该文件可不要。eg:

Java代码 收藏代码

ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);

FooService fooService = ctx.getBean(FooService.class);

六、 General-purpose type conversion system and field formatting system

Spring3.0加入了一个通用的类型转换系统，目前它被SpEL用作类型转换，并且可能被一个Spring容器使用，用于当绑定bean的属性值的时候进行类型转换另外，

还增加了一个字段格式化系统，它提供了更简单的使用并且强大的功能以替代UI环境下的Javabean的PropertyEidtors,例如在SpringMVC中。

七、 Object to XML mapping functionality (OXM) moved from Spring Web Services project

对象到XML之间的映射功能原来是在Spring Web services project项目中的，现在已经迁移到Spring框架的核心中，位于org.springframework.oxm中。

提供JAXB,XmlBeans以及XStream等方式的实现。

八、 Comprehensive REST support （增强的REST支持）

增强的REST支持。在web层的新特性是增加REST风格的web服务和web应用的支持，另外还新增了一些任何web应用都可以使用的新的注解。

服务端对于REST风格的支持,是通过扩展既有的注解驱动的MVC web 框架实现的.

客户端的支持则是 RestTemplate 类提供的，基于HttpClient.

无论服务端还是客户端REST功能,都是使用HttpConverter来简化对HTTP请求和应答过程中的对象到表现层的转换过程.

MarshallingHttpMessageConverter 使用了上面提到的"对象到 XML 的映射机制".

九、 @MVC additions （SpringMVC注解增强）

SpringMVC注解有所增强，新增了诸如@CookieValue 和 @RequestHeaders 这样的注解等.

十、 Declarative model validation（声明试模型验证）

支持JSR303，使用Hibernate Validator 作为默认实现， eg:

Java代码 收藏代码

public class Person {

@NotNull

@Max(64)

private String name;

@Min(0)

private int age;

}

十一、 Early support for Java EE 6（JEE6早期支持）

提前对JEE6的支持。提供了使用@Async注解对于异步方法调用的支持（或者EJB3.1里的@Asynchronous）

另外，新增对JSR303，JSF2.0，JPA2.0等的支持。

十二、 Embedded database support（嵌入式数据库的支持）

对于嵌入式的Java数据库引擎提供了广泛而方便的支持，包括HSQL,H2和Derby（方便测试）.

## 4.x 版本特性

## 5.x 版本特性

# Spring路线

# Spring 核心技术

## IoC (反转控制)

### IoC 简介

控制反转模式（也称作依赖性介入）的基本概念是：不创建对象，但是描述创建它们的方式。在代码中不直接与对象和服务连接，但在配置文件中描述哪一个组件需要哪一项服务。容器 （在 Spring 框架中是 IOC 容器） 负责将这些联系在一起。

在典型的 IOC 场景中，容器创建了所有对象，并设置必要的属性将它们连接在一起，决定什么时间调用方法。下表列出了 IOC 的一个实现模式。

|  |  |
| --- | --- |
| 类型 1 | 服务需要实现专门的接口，通过接口，由对象提供这些服务，可以从对象查询依赖性（例如，需要的附加服务） |
| 类型 2 | 通过 JavaBean 的属性（例如 setter 方法）分配依赖性 |
| 类型 3 | 依赖性以构造函数的形式提供，不以 JavaBean 属性的形式公开 |

Spring 框架的 IOC 容器采用类型 2 和类型3 实现。

### IoC 容器

Spring 设计的核心是 org.springframework.beans 包，它的设计目标是与 JavaBean 组件一起使用。这个包通常不是由用户直接使用，而是由服务器将其用作其他多数功能的底层中介。下一个最高级抽象是 BeanFactory 接口，它是工厂设计模式的实现，允许通过名称创建和检索对象。BeanFactory 也可以管理对象之间的关系。

BeanFactory 支持两个对象模型。

单态 - 模型提供了具有特定名称的对象的共享实例，可以在查询时对其进行检索。Singleton 是默认的也是最常用的对象模型。对于无状态服务对象很理想。

原型 - 模型确保每次检索都会创建单独的对象。在每个用户都需要自己的对象时，原型模型最适合。

bean 工厂的概念是 Spring 作为 IOC 容器的基础。IOC 将处理事情的责任从应用程序代码转移到框架。正如我将在下一个示例中演示的那样，Spring 框架使用 JavaBean 属性和配置数据来指出必须设置的依赖关系。

BeanFactory 接口

因为 org.springframework.beans.factory.BeanFactory 是一个简单接口，所以可以针对各种底层存储方法实现。最常用的 BeanFactory 定义是 XmlBeanFactory，它根据 XML 文件中的定义装入 bean，如清单 1 所示。

清单 1. XmlBeanFactory

BeanFactory factory = new XMLBeanFactory(new FileInputSteam("mybean.xml"));

在 XML 文件中定义的 Bean 是被消极加载的，这意味在需要 bean 之前，bean 本身不会被初始化。要从 BeanFactory 检索 bean，只需调用 getBean() 方法，传入将要检索的 bean 的名称即可，如清单 2 所示。

清单 2. getBean()

MyBean mybean = (MyBean) factory.getBean("mybean");

每个 bean 的定义都可以是 POJO （用类名和 JavaBean 初始化属性定义） 或 FactoryBean。FactoryBean 接口为使用 Spring 框架构建的应用程序添加了一个间接的级别。

### IOC 示例

理解控制反转最简单的方式就是看它的实际应用。在对由三部分组成的 Spring 系列 的第 1 部分进行总结时，我使用了一个示例，演示了如何通过 Spring IOC 容器注入应用程序的依赖关系（而不是将它们构建进来）。

我用开启在线信用帐户的用例作为起点。对于该实现，开启信用帐户要求用户与以下服务进行交互：

信用级别评定服务，查询用户的信用历史信息。

远程信息链接服务，插入客户信息，将客户信息与信用卡和银行信息连接起来，以进行自动借记（如果需要的话）。

电子邮件服务，向用户发送有关信用卡状态的电子邮件。

## AOP 介绍 (切面编程)

# Spring 常用注解

事务管理

# SpringMVC

整合配置

常用注解

# 关键字

spring 3 特性、spring 4 特性、

# 引用资料

Spring 框架简介:https://www.ibm.com/developerworks/cn/java/wa-spring1/

Spring3.0的新特性及其分析 http://zwustudy.iteye.com/blog/1671499

Spring 4.x框架中的新特性---Spring4.0框架的新功能和改善 http://blog.csdn.net/think\_soft/article/details/49785619

Spring学习-01:学习路线综述 http://blog.csdn.net/dove\_knowledge/article/details/66472668