# Spring框架是什么？

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**1**

[下一篇](http://www.yiibai.com/spring/spring_architecture.html)

Spring是为企业Java最流行的应用程序开发框架。数以百万计的世界各地的开发人员使用Spring框架来创建高性能，易于测试的，可重用的代码。

Spring框架是一个开源的Java平台，它最初是由Rod Johnson编写并在2003年6月在Apache2.0许可下首次发布。

Spring是轻量级的，当涉及到大小和透明度。 spring框架的基本版本是大约2MB。

Spring框架的核心功能可以在任何Java应用程序中使用，但也有扩展的Java EE平台上构建Web应用程序。 Spring框架的目标，以使J2EE开发更容易使用，并且通过使基于POJO的编程模型，促进良好的编程习惯。

## 使用Spring框架的好处：

以下是一些使用Spring框架的极大的好处列表：

* Spring 使开发人员使用POJO来开发企业级应用程序。仅使用POJO的好处是不需要EJB容器产品，如应用服务器，但必须只使用一个强大的servlet容器，如Tomcat 或某些商业产品的选择。
* Spring 被组织在一个模块化的方式。虽然包和类的数量是巨大的，你只有担心那些需要的，而其它的就忽略了。
* Spring不重新发明轮子，而是真正利用了一些像一些ORM框架，日志框架，JEE，Quartz和JDK定时器，其他视图技术的现有技术。
* 编写测试Spring的应用程序很简单，因为环境相关的代码被移动到这个框架中。此外，通过使用JavaBean风格的POJO，它变得更容易使用依赖注入注入测试数据。
* Spring web框架是一个设计良好的Web MVC框架，它提供了一个很好的替代Web框架，如Struts或其他过度设计或不太流行的Web框架。
* Spring提供了一个方便的API平移技术特定的异常（通过JDBC，Hibernate，或者JDO抛出，例如）成一致，未经检查的异常。
* 轻量级的IoC容器往往是轻量级的，尤其是相对EJB容器，例如。这有利于开发和有限的内存和CPU资源的计算机上部署应用程序。
* Spring提供了一致的事务管理接口，可向下扩展到（使用一个单一的数据库，例如）本地事务并扩展到全局事务（使用JTA，例如）。

## 依赖注入（DI）：

Spring最有确定的技术是依赖注入控制反转（DI）。控制反转（IoC）是一个笼统的概念，它可以表现在许多不同的方式和依赖注入仅仅是控制反转的一个具体的例子。

当编写一个复杂的Java应用程序，应用程序类应该尽可能独立其他Java类来增加重复使用这些类，并独立于其他类别的测试它们，而这样做单元测试的可能性。依赖注入有助于粘合这些类在一起，同时保持他们的独立。

什么是依赖注入是什么呢？让我们来看看这两个词分开。这里的依赖性部分转化为两个类之间的关联。例如，A类是依赖B类，现在，让我们来看看第二部分，注入。这一切都意味着，B类将由IOC注入到A类得到。

依赖注入可以将参数传递给构造函数的方式或使用后建设setter方法​​发生。依赖注入是Spring框架的核心，所以会在一个单独的章节，一个不错的例子解释这个概念。

## 面向切面编程（AOP）：

Spring的一个关键组件是面向切面编程（AOP）框架。跨越多个点的应用程序的功能被称为横切关注点和这些横切关注点是从应用程序的业务逻辑概念上区分开来。有方面的各种常见很好的例子包括日志记录，声明性事务，安全性和高速缓存等。

模块化的OOP关键单元是类，而在AOP中模块化的单元则是切面。DI帮助您从彼此分离的应用程序对象，AOP可以帮助你消除来自他们影响的对象横切关注点。

Spring框架AOP模块提供了面向方面编程实现，允许您定义方法拦截器和切入点，以干净脱钩，实现的功能，应该分开代码。我将在单独的一章讨论Spring AOP的概念。

# Spring框架的体系结构

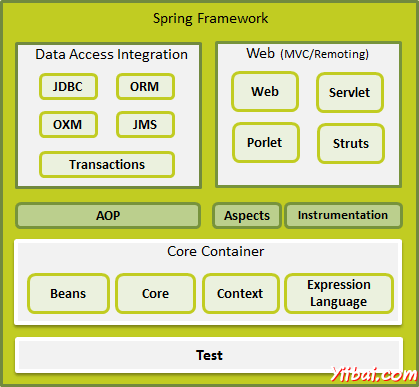
 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_overview.html)[下一篇](http://www.yiibai.com/spring/spring_environment_setup.html)

Spring有可能成为所有企业应用程序的一站式服务，然而，Spring是模块化的，使您可以挑选哪些模块是适用的，而不必把在余下的也引入。以下部分给出详细介绍在Spring框架中所有可用的模块。

Spring框架提供约20个模块，可以根据应用程序的要求来使用。



## 核心容器：

核心容器组成的核心，Beans，上下文和表达式语言模块，其细节如下：

* 核心模块提供了框架的基本组成部分，包括IoC和依赖注入功能。
* Bean模块提供BeanFactory是工厂模式的经典实现。
* Context 上下文模块建立在由核心和Bean类模块提供了坚实的基础，它是访问定义和配置的任何对象的媒介。在ApplicationContext接口是上下文模块的焦点。.
* 表达式语言模块提供了在运行时查询和操作对象图的强大的表达式语言。

## 数据访问/集成：

数据访问/集成层包括[JDBC](http://www.yiibai.com/jdbc)，ORM，OXM，JMS和事务处理模块，其细节如下：

* JDBC模块提供了不再需要冗长的JDBC编码相关了JDBC的抽象层。
* ORM模块提供的集成层。流行的对象关系映射API，包括JPA，JDO，Hibernate和iBatis。
* OXM模块提供了一个支持对象/ XML映射实现对JAXB，Castor，使用XMLBeans，JiBX和XStream 的抽象层。
* Java消息服务JMS模块包含的功能为生产和消费的信息。
* 事务模块支持编程和声明式事务管理实现特殊接口类，并为所有的POJO。

## Web:

在Web层包括网络，Web-Servlet，Web-Struts和网络的Portlet 组件，其细节如下：

* Web模块提供了基本的Web开发的集成特性，例如多方文件上传功能和使用的servlet监听器的IoC容器初始化和针对Web的应用程序上下文。
* Web-Servlet 模块包含Spring的模型 - 视图 - 控制器（MVC）实现Web应用程序。
* Web-Struts 模块包含支持类内的Spring应用程序集成了经典的Struts Web层。
* Web-Portlet 模块提供了MVC实现在portlet环境中使用和镜子Web的Servlet的模块的功能。

## 其他：

还有像AOP，切面，规范，网络和测试模块，其详细情况如下其他一些重要的模块：

* AOP模块提供了面向方面编程实现，允许您定义方法拦截器和切入点，以干净解耦，实现的功能，应该分开的代码。
* Aspects 模块提供与AspectJ的集成这又是一个功能强大且成熟的面向方面编程（AOP）框架。
* Instrumentation 模块提供了一定的应用服务器中使用类工具的支持和类加载器实现。
* 测试模块支持Spring 组件使用 [JUnit](http://www.yiibai.com/junit)或 [TestNG](http://www.yiibai.com/testng)框架的测试。

# Spring 环境安装配置

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_architecture.html)[下一篇](http://www.yiibai.com/spring/spring_hello_world_example.html)

本教程将指导您如何准备开发环境并使用 Spring 框架工作。本教程还将教你如何安装JDK，Tomcat和Eclipse 在安装Spring框架之前：

## 第1步 - 安装Java开发工具包（JDK）：

JavaSE下载：可以从Oracle公司的**Java**网站下载**SDK**的最新版本。会发现说明在下载的文件中教你如何安装JDK，按照说明安装和配置设置的说明。最后，设置**PATH**和***JAVA\_HOME***环境变量指向包含java和javac，通常分别为**java\_install\_dir/bin**和**java\_install\_dir**目录。

如果运行的是Windows并安装了JDK在***C:\jdk1.6.0\_15***, 你可以把下面这行写入到**C:\autoexec.bat**文件.

set PATH=C:\jdk1.6.0\_15\bin;%PATH%

set JAVA\_HOME=C:\jdk1.6.0\_15

另外，在Windows NT/2000/XP上，也可以右键单击“我的电脑”，选择“属性”，然后“高级”，然后“环境变量”。然后更新PATH 值，然后按OK按钮。

在Unix（Solaris和Linux等），如果SDK安装在/usr/local/jdk1.6.0\_15 如果使用的是C shell，干脆把下面的内容写到 .cshrc文件。

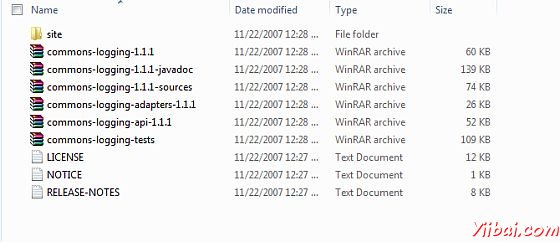
setenv PATH /usr/local/jdk1.6.0\_15/bin:$PATH

setenv JAVA\_HOME /usr/local/jdk1.6.0\_15

另外，如果使用的集成开发环境（IDE），如Borland公司的JBuilder，Eclipse，IntelliJ IDEA或Sun ONE Studio 中编译和运行一个简单的程序，以确认IDE知道安装Java，否则应该做适当的设置。

## 步骤2 - 安装Apache通用日志API：

你可以从这里下载的Apache 通用日志API 的最新版本 [http://commons.apache.org/logging/](http://commons.apache.org/logging/download_logging.cgi). 一旦下载安装，解压二进制分发到一个方便的位置。例如在C:\commons-logging-1.1.1 ，在Linux/Unix 为 /usr/local/commons-logging-1.1.1。该目录将有如下jar文件和其他证明文件等。



请确保在这个目录上正确设置CLASSPATH变量，否则会在运行应用程序面临的问题。

## 第3步 - 安装Eclipse集成开发环境

在本教程中的所有例子使用Eclipse IDE。所以我建议你应该有安装在机器上 [Eclipse](http://www.yiibai.com/eclipse/)的最新版本。

安装Eclipse IDE，请从最新的Eclipse二进制文件 <http://www.eclipse.org/downloads/>. 一旦下载安装，解压二进制放到一个方便的位置。例如在**C:\eclipse** ，或**/usr/local/eclipse** 在Linux/Unix最后恰当设置PATH变量。

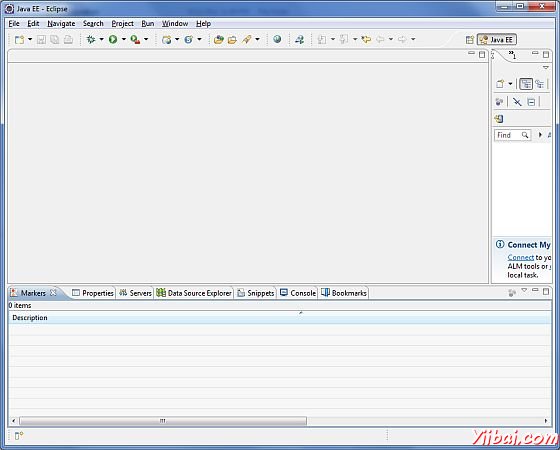
Eclipse可以通过在Windows机器上执行以下命令来启动，或者可以简单地双击 eclipse.exe

%C:\eclipse\eclipse.exe

Eclipse可以通过执行在Unix（Solaris和Linux等）下面的命令机器启动：

$/usr/local/eclipse/eclipse

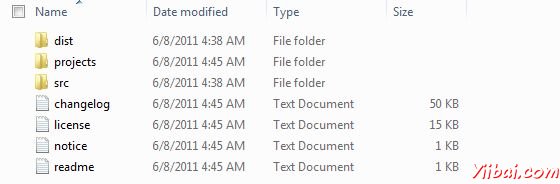
启动成功后，如果一切正常，那么它应该显示如下结果：



## 第4步 - 安装Spring框架库

现在，如果一切正常，那么你可以继续设置 Spring 框架。以下是简单的步骤下载并安装框架的机器上。

* 选择是否要在 Windows 或 UNIX 上安装 Spring，然后进行下一个步骤，下载的zip文件适用于 Windows 和 Unix 为 .tz 文件。
* 从下载的Spring框架的二进制文件的最新版本<http://www.springsource.org/download>.
* 在写这篇教程的时候，我下载**spring-framework-3.1.0.M2.zip** 在Windows ，当解压缩下载的文件，它目录结构：**C:\spring-framework-3.1.0.M2** 如下。



会发现所有的Spring库在目录**C:\spring-framework-3.1.0.M2\dist**。请确保在这个目录上正确设置CLASSPATH变量，否则在运行应用程序面临的问题。如果使用的是Eclipse那么就不需要设置CLASSPATH，因为所有的设置将通过Eclipse中完成。

一旦用这最后一步完成后，你就可以继续第一个Spring例子，将在下一章看到。

# Spring Hello World 实例

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**2**

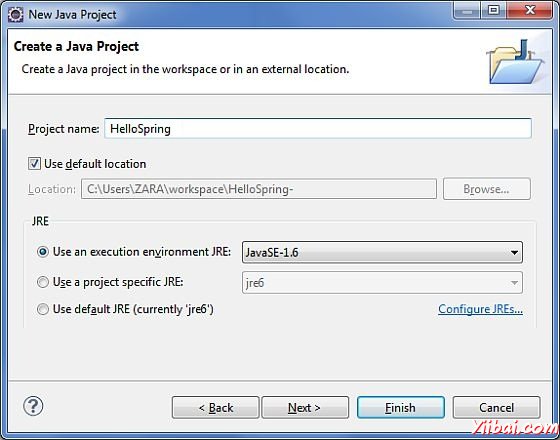
[上一篇](http://www.yiibai.com/spring/spring_environment_setup.html)[下一篇](http://www.yiibai.com/spring/spring_ioc_containers.html)

让我们从实际编程中使用Spring框架。在开始编写使用Spring框架的第一个例子，你必须确保已经设置Spring环境。需要对Eclipse IDE有一点知识。

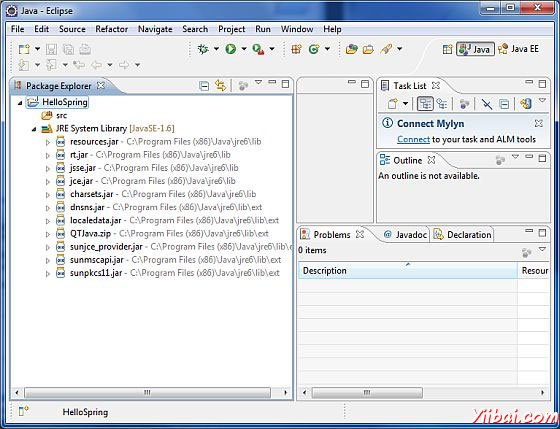
因此，让我们着手编写一个简单的Spring应用程序将打印 “Hello World！” 或基于Spring bean 配置文件来完成配置的其他信息。

## 第1步 - 创建Java项目：

第一步是使用**Eclipse IDE**创建一个简单的Java项目。选项 **File -> New -> Project** 最后选择**Java Project**从向导列表向导。现在，项目命名为 **HelloSpring**使用向导窗口，如下所示：

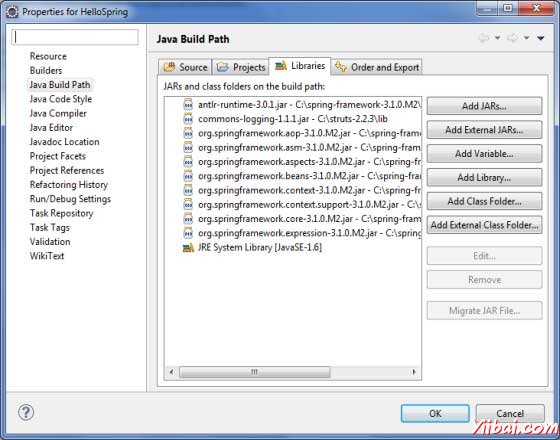


项目创建成功后，将在**Project Explorer** 看到下面的内容:



## 第2步 - 添加必需的库：

第二步添加 **Spring**框架，并在我们的项目中通用的日志API库。要做到这一点，右键单击您的项目名称***HelloSpring*** 然后按照上下文菜单中提供以下选项： **Build Path -> Configure Build Path** 如下显示的**Java**构建路径窗口：



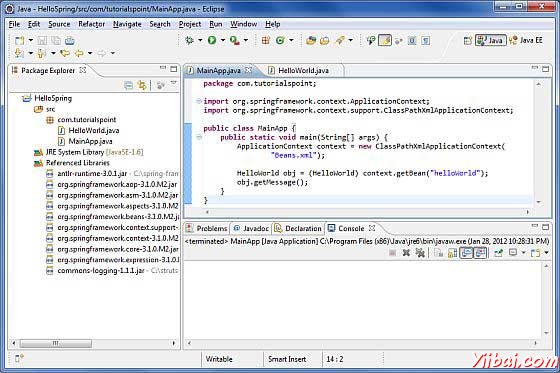
现在，使用**Add External JARs** 选项 **Libraries**卡提供外部JAR按钮，添加从Spring框架和通用日志安装目录下面的核心JAR文件：

* antlr-runtime-3.0.1
* org.springframework.aop-3.1.0.M2
* org.springframework.asm-3.1.0.M2
* org.springframework.aspects-3.1.0.M2
* org.springframework.beans-3.1.0.M2
* org.springframework.context.support-3.1.0.M2
* org.springframework.context-3.1.0.M2
* org.springframework.core-3.1.0.M2
* org.springframework.expression-3.1.0.M2
* commons-logging-1.1.1

## 第3步 - 创建源文件：

现在让我们根据实际创建的源文件 **HelloSpring**项目。首先，我们需要创建一个名为包 **com.yiibai**. 要做到这一点，右键点击 src 在包资源管理部分，并按照选项：**New -> Package**.

下一步我们创建 **HelloWorld.java**和 **MainApp.java** 在**com.yiibai**包下文件。



这里是**HelloWorld.java**的文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

下面是第二个文件 **MainApp.java**的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

}

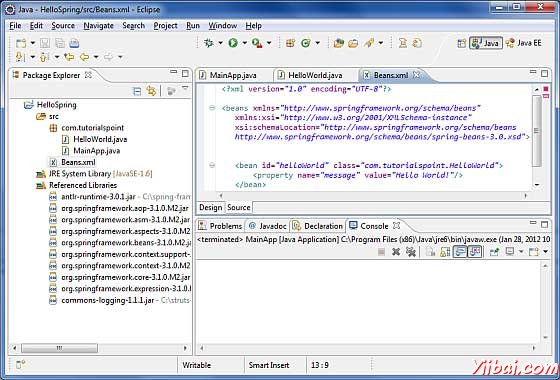
}

有以下两个要点需要注意在主要程序中：

1. 第一步是创建在这里我们使用框架API **ClassPathXmlApplicationContext()**应用程序上下文。这个API加载 **beans**的配置文件，并最终基于所提供的API，它需要创建并初始化所有的对象。在配置文件中提到的beans 类。
2. 第二个步骤是用来使用创建的上下文的 **getBean()**方法获得所需的bean。此方法使用 **bean**的 **id**返回，最终可以创建实际对象的通用对象。一旦有了对象，就可以使用这个对象调用任何类方法。

## 第4步 - 创建bean配置文件：

需要创建一个**bean**配置文件，该文件是一个XML文件，并作为 **cement**等类组合在一起。这个文件需要在**src**目录下创建如下图所示：



通常开发人员保留此文件的名称为 **beans.xml** 文件，但是必须为独立的，选择你喜欢的任何名称。必须确保这个文件在 **CLASSPATH** 中可用，并使用相同的名称在主应用程序而创建的应用程序上下文，如图**MainApp.java**文件。

在 **beans.xml**中用于分配唯一的ID，以不同的 **bean**和控制创造不同的值对象，而不会影响任何 Spring 的源文件。例如，使用下面的文件，可以通过为“message”变量的任何值，这样你就可以打印消息的不同的值，而不会影响的 **HelloWorld.java** 和 **MainApp.java**文件。让我们来看看它是如何工作的：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld">

<property name="message" value="Hello World!"/>

</bean>

</beans>

在**Spring**应用程序被加载到内存中，框架利用了上面的配置文件来创建所有定义的 **Bean**，并在**<bean>** 标记定义为它们分配一个唯一的 ID。您可以使用**<property>** 标签来传递在创建对象时使用不同的变量的值。

## 第5步 - 运行程序：

创建源代码和 **bean**配置文件来完成，准备好这一步是编译和运行程序。要做到这一点，保持**MainApp.Java** 文件选项卡激活并使用在**Eclipse IDE**中提供 ***Run*** 选项，或使用**Ctrl + F11**可以编译并运行应用程序**MainApp**。如果一切顺利那么应用程序这将打印以下信息在 Eclipse IDE 控制台：

Your Message : Hello World!

恭喜，你已经创建了第一个Spring应用并成功执行。您可以通过更改“message”属性的值，并保持两者的源文件不变见上 Spring 应用程序的灵活性。

# Spring IoC容器

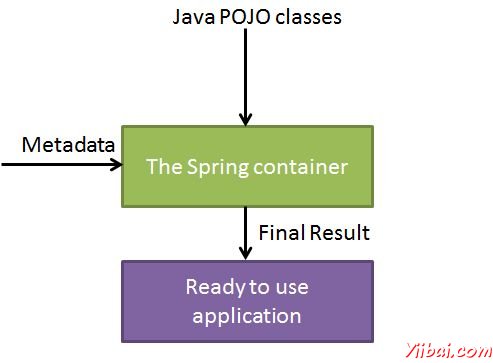
 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**2**

[上一篇](http://www.yiibai.com/spring/spring_hello_world_example.html)[下一篇](http://www.yiibai.com/spring/spring_beanfactory_container.html)

Spring容器是Spring框架的核心。容器将创建对象，它们连接在一起，配置它们，并从创建到销毁管理他们的整个生命周期。在Spring容器使用依赖注入（DI）来管理组成应用程序的组件。这些对象被称为Spring Beans，我们将在下一章中讨论。

容器获得其上的哪些对象进行实例化，配置和组装通过阅读提供的配置元数据的说明。配置元数据可以通过XML，Java注释或Java代码来表示。下面的图是Spring如何工作的高层次图。 Spring IoC容器是利用Java的POJO类和配置元数据的产生完全配置和可执行的系统或应用程序。



Spring提供了以下两种不同类型的容器。

|  |  |
| --- | --- |
| **S.N.** | **容器& 描述** |
| 1 | [Spring BeanFactory 容器](http://www.yiibai.com/spring/spring_beanfactory_container.html) 这是最简单的容器DI提供基本的支持和定义由*org.springframework.beans.factory.BeanFactory* 接口. BeanFactory或者相关的接口，例如实现BeanFactoryAware，InitializingBean，DisposableBean，仍然存在在Spring向后兼容性与大量的与Spring整合第三方框架的目的。 |
| 2 | [Spring ApplicationContext 容器](http://www.yiibai.com/spring/spring_applicationcontext_container.html)  此容器添加了更多的企业特定的功能，例如从一个属性文件解析文本消息的能力，并发布应用程序事件感兴趣的事件监听器的能力。此容器是由 *org.springframework.context.ApplicationContext*接口定义. |

在ApplicationContext 容器包括BeanFactory的容器的所有功能，所以因此通常建议在BeanFactory。 BeanFactory仍然可以用于重量轻的应用，如移动装置或基于小应用程序的应用中的数据量和速度是显著。

# Spring bean定义

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_applicationcontext_container.html)[下一篇](http://www.yiibai.com/spring/spring_bean_scopes.html)

形成应用程序的骨干是由Spring IoC容器所管理的对象称为bean。bean被实例化，组装，并通过Spring IoC容器所管理的对象。这些bean由容器提供，例如，在XML的<bean/>定义，已经看到了前几章的形式配置元数据创建。

bean定义包含所需要的容器要知道以下称为配置元数据的信息：

* 如何创建一个bean
* Bean 生命周期的详细信息
* Bean 依赖关系

上述所有配置元数据转换成一组的下列属性构成每个bean的定义。

|  |  |
| --- | --- |
| **属性** | **描述** |
| class | 此属性是强制性的，并指定bean类被用来创建bean。 |
| name | 此属性指定唯一bean标识符。在基于XML的配置元数据时，您可以使用id和/或name属性来指定bean标识符 |
| scope | 该属性指定一个特定的bean定义创建，它会在bean作用域本章要讨论的对象范围。 |
| constructor-arg | 这是用来注入的依赖关系，并在接下来的章节中进行讨论。 |
| properties | 这是用来注入的依赖关系，并在接下来的章节中进行讨论。 |
| autowiring mode | 这是用来注入的依赖关系，并在接下来的章节中进行讨论。 |
| lazy-initialization mode | 延迟初始化的bean告诉IoC容器创建bean实例时，它首先要求，而不是在启动时。 |
| initialization method | 回调只是在bean的所有必要属性后调用已设置的容器。它会在bean的生命周期章节中讨论。 |
| destruction method | 当包含该bean容器被销毁所使用的回调。它会在bean的生命周期章节中讨论。 |

## Spring配置元数据

Spring IoC容器完全由在此配置元数据实际写入的格式解耦。有下列提供的配置元数据的Spring容器三个重要的方法：

1. 基于XML的配置文件。
2. 基于注解的配置
3. 基于Java的配置

我们已经看到了基于XML的配置元数据如何提供给容器，但让我们看到了不同的bean定义，包括延迟初始化，初始化方法和销毁方法基于XML配置文件的另一个示例：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- A simple bean definition -->

<bean id="..." class="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- A bean definition with lazy init set on -->

<bean id="..." class="..." lazy-init="true">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- A bean definition with initialization method -->

<bean id="..." class="..." init-method="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- A bean definition with destruction method -->

<bean id="..." class="..." destroy-method="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- more bean definitions go here -->

</beans>

可以查看[Spring Hello World 示例](http://www.yiibai.com/spring/spring_hello_world_example.html) 来了解如何定义，配置和创建Spring Bean。

有关基于注解的配置在一个单独的章节讨论。在一个单独的章节刻意保留它，因为希望能掌握一些Spring其他的重要概念，在开始用注解依赖注入来编程。

# Spring Bean Scopes作用域

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_bean_definition.html)[下一篇](http://www.yiibai.com/spring/spring_bean_life_cycle.html)

当定义一个Spring的<bean>，必须声明bean 作用域的选项。例如，要强制Spring需要产生一个新的bean实例，应该声明bean的scope属性为prototype。如果你希望Spring 每次都返回同一个bean实例，应该声明bean的作用域，方式类似属性是单例。

Spring框架支持以下五个作用域，其中三个只有当您使用Web感知的 ApplicationContext 可用。

|  |  |
| --- | --- |
| **范围** | **描述** |
| singleton | This scopes the bean definition to a single instance per Spring IoC container (default). |
| prototype | This scopes a single bean definition to have any number of object instances. |
| request | This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext. |
| session | This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |
| global-session | This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |

本章将讨论前两个范围和其余三将讨论的时候，我们将讨论有关Web感知Spring的ApplicationContext。

## singleton作用域：

如果范围设置为单例，Spring IoC容器创建了一个由该bean定义的对象只有一个实例。这个单一实例存储在这样的单例bean的高速缓存，以及所有后续请求和引用针对该bean返回缓存对象。

默认范围是始终单例，但是当你需要bean的一个实例，可以设置的范围属性单例在bean配置文件中，如下图所示：

<!-- A bean definition with singleton scope -->

<bean id="..." class="..." scope="singleton">

<!-- collaborators and configuration for this bean go here -->

</bean>

### 示例：

让我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld* and *MainApp* under the *com.yiibai* package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是HelloWorld.java 文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

objA.setMessage("I'm object A");

objA.getMessage();

HelloWorld objB = (HelloWorld) context.getBean("helloWorld");

objB.getMessage();

}

}

以下是需要singleton作用域配置文件beans.xml文件：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld"

scope="singleton">

</bean>

</beans>

一旦创建源代码和bean配置文件来完成，运行应用程序。如果一切顺利，这将打印以下信息：

Your Message : I'm object A

Your Message : I'm object A

## 原型作用域：

如果范围设置为原型，那么Spring IoC容器创建对象的新的bean实例为每个特定的bean发出请求时的时间。作为一项规则，使用prototype作用域为所有状态的bean类和singleton作用域为无状态的bean。

要定义一个原型作用域，可以设置的范围属性为原型的bean配置文件中，如下图所示：

<!-- A bean definition with singleton scope -->

<bean id="..." class="..." scope="prototype">

<!-- collaborators and configuration for this bean go here -->

</bean>

### 示例：

让我们在地方工作的Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld* and *MainApp* under the *com.yiibai* package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是HelloWorld.java 文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

objA.setMessage("I'm object A");

objA.getMessage();

HelloWorld objB = (HelloWorld) context.getBean("helloWorld");

objB.getMessage();

}

}

以下是必需的原型作用域的配置文件beans.xml：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld"

scope="prototype">

</bean>

</beans>

创建源代码和bean配置文件完成后，让我们运行应用程序。如果一切顺利，这将打印以下信息：

Your Message : I'm object A

Your Message : null

# Spring bean 生命周期

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_bean_scopes.html)[下一篇](http://www.yiibai.com/spring/spring_bean_post_processors.html)

Spring bean的生命周期是很容易理解。当一个bean实例化时，它可能需要执行一些初始化把它转换成可用状态。类似地，当bean不再需要，并且从容器中取出，一些清理的工作可能也需要做。

不过，还有把bean背后的实例化和销毁时间之间的场景发生的活动，但是本章将只讨论其中两个是需要在bean的初始化和销毁的时候，重要bean的生命周期回调方法。

要定义安装和拆卸一个bean，我们只是声明了初始化方法和/或销毁，方法的参数<bean>。在init-method属性指定一个方法，是被调用bean后立即实例化。同样，销毁方法规定了被调用当bean被从容器中取出之前的方法。

## 初始化回调：

org.springframework.beans.factory.InitializingBean 接口指定一个单一的方法：

void afterPropertiesSet() throws Exception;

因此，可以简单地实现上述接口和初始化工作可以在里面afterPropertiesSet() 方法，如下所示：

public class ExampleBean implements InitializingBean {

public void afterPropertiesSet() {

// do some initialization work

}

}

在基于XML的配置元数据的情况下，可以使用init-method 属性来指定具有void无参数签名的方法的名称。例如：

<bean id="exampleBean"

class="examples.ExampleBean" init-method="init"/>

下面是类的定义：

public class ExampleBean {

public void init() {

// do some initialization work

}

}

## 销毁回调

org.springframework.beans.factory.DisposableBean接口指定一个单一的方法：

void destroy() throws Exception;

因此，你可以简单地实现上述接口和定稿工作可以做里面的destroy() 方法，如下所示：

public class ExampleBean implements DisposableBean {

public void destroy() {

// do some destruction work

}

}

在基于XML的配置元数据的情况下，您可以使用destroy-method属性来指定具有void无参数签名的方法的名称。例如：

<bean id="exampleBean"

class="examples.ExampleBean" destroy-method="destroy"/>

下面是类的定义：

public class ExampleBean {

public void destroy() {

// do some destruction work

}

}

如果您在非web应用环境中使用Spring的IoC容器，例如在桌面富客户端环境; 注册关闭钩子在JVM中。这样做可以确保正常关机，并让所有的资源都被释放调用singleton bean上的相关destroy方法。

建议不要使用的InitializingBean或者DisposableBean的回调，因为XML配置提供极大的灵活性在命名你的方法方面。

## 例如：

使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld* and *MainApp* under the *com.yiibai* package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是HelloWorld.java的文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

public void init(){

System.out.println("Bean is going through init.");

}

public void destroy(){

System.out.println("Bean will destroy now.");

}

}

以下是MainApp.java文件的内容。在这里，您需要注册，是在AbstractApplicationContext 类中声明一个关机hookregisterShutdownHook() 方法。这将确保正常关机，并调用相关的destroy方法。

package com.yiibai;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

AbstractApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

context.registerShutdownHook();

}

}

下面是需要的init和destroy方法配置文件beans.xml文件：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld"

class="com.yiibai.HelloWorld"

init-method="init" destroy-method="destroy">

<property name="message" value="Hello World!"/>

</bean>

</beans>

一旦创建源和bean配置文件来完成，让我们运行应用程序。如果一切顺利，这将打印以下信息：

Bean is going through init.

Your Message : Hello World!

Bean will destroy now.

## 缺省的初始化和销毁方法：

如果你有过多的Bean初始化和销毁或者具有相同名称的方法，不需要声明的初始化方法和销毁方法在每一个bean上。相反框架提供了灵活使用<beans>元素default-init-method和default-destroy-method 属性如下配置这样的情况：

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"

default-init-method="init"

default-destroy-method="destroy">

<bean id="..." class="...">

<!-- collaborators and configuration for this bean go here -->

</bean>

</beans>

# Spring发布处理器

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

BeanPostProcessor 的接口定义，可以实现提供自己的实例化逻辑，依赖解析逻辑等，也可以以后在Spring容器实例化完毕，配置和初始化一个bean通过插入一个或多个的BeanPostProcessor实现一些自定义逻辑回调方法实现。

可以配置多个的BeanPostProcessor接口，控制这些的BeanPostProcessor接口，通过设置属性顺序执行顺序提供的BeanPostProcessor实现了Ordered接口。

BeanPostProcessor可以对bean（或对象）操作实例，这意味着Spring IoC容器实例化一个bean实例，然后BeanPostProcessor的接口做好自己的工作。

ApplicationContext会自动检测已定义实现的BeanPostProcessor接口和注册这些bean类为后置处理器，可然后通过在容器创建bean，在适当时候调用任何bean。

## 示例:

下面的示例显示如何编写，注册和使用BeanPostProcessor 可以在一个ApplicationContext 的上下文。

使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld*, *InitHelloWorld* and *MainApp* under the *com.yiibai*package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是 HelloWorld.java 文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

public void init(){

System.out.println("Bean is going through init.");

}

public void destroy(){

System.out.println("Bean will destroy now.");

}

}

这是实现BeanPostProcessor，之前和之后的任何bean的初始化它打印一个bean的名字非常简单的例子。可以因为有两个后处理器的方法对内部bean对象访问之前和实例化一个bean后执行更复杂的逻辑。

这里是InitHelloWorld.java文件的内容：

package com.yiibai;

import org.springframework.beans.factory.config.BeanPostProcessor;

import org.springframework.beans.BeansException;

public class InitHelloWorld implements BeanPostProcessor {

public Object postProcessBeforeInitialization(Object bean,

String beanName) throws BeansException {

System.out.println("BeforeInitialization : " + beanName);

return bean; // you can return any other object as well

}

public Object postProcessAfterInitialization(Object bean,

String beanName) throws BeansException {

System.out.println("AfterInitialization : " + beanName);

return bean; // you can return any other object as well

}

}

以下是MainApp.java 文件的内容。在这里，需要注册一个关闭挂钩registerShutdownHook() 是在AbstractApplicationContext类中声明的方法。这将确保正常关闭，并调用相关的destroy方法。

package com.yiibai;

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

AbstractApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

context.registerShutdownHook();

}

}

下面是init和destroy方法需要的配置文件beans.xml文件：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld"

init-method="init" destroy-method="destroy">

<property name="message" value="Hello World!"/>

</bean>

<bean class="com.yiibai.InitHelloWorld" />

</beans>

创建源代码和bean配置文件完成后，让我们运行应用程序。如果一切顺利，这将打印以下信息：

BeforeInitialization : helloWorld

Bean is going through init.

AfterInitialization : helloWorld

Your Message : Hello World!

Bean will destroy now.

# Spring bean定义继承

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_bean_post_processors.html)[下一篇](http://www.yiibai.com/spring/spring_dependency_injection.html)

bean定义可以包含很多的配置信息，包括构造函数的参数，属性值，比如初始化方法，静态工厂方法名等容器的具体信息。

子bean定义从父定义继承配置数据。子的定义可以覆盖一些值，或者根据需要添加其他。

Spring bean定义继承无关，与Java类的继承，但继承的概念是一样的。你可以定义一个父bean定义为模板和其他孩子bean可以从父bean继承所需的配置。

当使用基于XML的配置元数据，指明一个子bean定义使用所在的当前属性指定的父bean作为这个属性的值。

## 例如：

让我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld*, *HelloIndia* and *MainApp* under the *com.yiibai*package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

以下是我们定义的“HelloWorld”豆里面有两个属性message1和message2配置文件beans.xml中。下一步“helloIndia”豆已经被定义为“HelloWorld”的子bean使用parent属性。该子bean继承message2属性原状，并覆盖message1 属性，并引入多一个属性message3。

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld">

<property name="message1" value="Hello World!"/>

<property name="message2" value="Hello Second World!"/>

</bean>

<bean id="helloIndia" class="com.yiibai.HelloIndia"

parent="helloWorld">

<property name="message1" value="Hello India!"/>

<property name="message3" value="Namaste India!"/>

</bean>

</beans>

这里是HelloWorld.java 文件的内容：

package com.yiibai;

public class HelloWorld {

private String message1;

private String message2;

public void setMessage1(String message){

this.message1 = message;

}

public void setMessage2(String message){

this.message2 = message;

}

public void getMessage1(){

System.out.println("World Message1 : " + message1);

}

public void getMessage2(){

System.out.println("World Message2 : " + message2);

}

}

这里是HelloIndia.java文件的内容：

package com.yiibai;

public class HelloIndia {

private String message1;

private String message2;

private String message3;

public void setMessage1(String message){

this.message1 = message;

}

public void setMessage2(String message){

this.message2 = message;

}

public void setMessage3(String message){

this.message3 = message;

}

public void getMessage1(){

System.out.println("India Message1 : " + message1);

}

public void getMessage2(){

System.out.println("India Message2 : " + message2);

}

public void getMessage3(){

System.out.println("India Message3 : " + message3);

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

objA.getMessage1();

objA.getMessage2();

HelloIndia objB = (HelloIndia) context.getBean("helloIndia");

objB.getMessage1();

objB.getMessage2();

objB.getMessage3();

}

}

创建完成源代码和bean配置文件，让我们运行应用程序。如果一切顺利，这将打印以下信息：

World Message1 : Hello World!

World Message2 : Hello Second World!

India Message1 : Hello India!

India Message2 : Hello Second World!

India Message3 : Namaste India!

如果你在这里看到，我们没有通过message2同时创建“helloIndia”的bean，但它通过了，因为bean定义的继承。

## bean定义模板：

您可以创建可以在不会花太多功夫被其他子bean定义的bean定义模板。在定义bean定义模板，不应指定类属性，并应与真值指定如下所示的抽象属性：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="beanTeamplate" abstract="true">

<property name="message1" value="Hello World!"/>

<property name="message2" value="Hello Second World!"/>

<property name="message3" value="Namaste India!"/>

</bean>

<bean id="helloIndia" class="com.yiibai.HelloIndia"

parent="beanTeamplate">

<property name="message1" value="Hello India!"/>

<property name="message3" value="Namaste India!"/>

</bean>

</beans>

父bean不能被实例化它自己，因为它是不完整的，而且它也明确地标记为抽象。当一个定义是抽象的这个样子，它只是作为一个纯粹的模板bean定义，充当子定义的父定义使用。

# Spring 依赖注入

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_bean_definition_inheritance.html)[下一篇](http://www.yiibai.com/spring/constructor_based_dependency_injection.html)

每一个基于java的应用程序都有一个共同工作来展示给用户看到的内容作为工作的应用几个对象。当编写一个复杂的Java应用程序，应用程序类应该尽可能独立其他Java类来增加重复使用这些类，并独立于其他类别的测试它们，而这样做单元测试的可能性。依赖注入（或有时称为布线）有助于粘合这些类在一起，同时保持他们的独立。

考虑有其中有一个文本编辑器组件的应用程序，要提供拼写检查。标准的代码将看起来像这样：

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor() {

spellChecker = new SpellChecker();

}

}

我们在这里所做的就是创建文本编辑和拼写检查之间的依赖性。在控制方案中的反转，我们反而会做这样的事情：

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

在这里，文本编辑不应该担心拼写检查落实。拼写检查器将独立实施，将提供给文本编辑在文本编辑实例化的时候，这整个过程是由Spring框架的控制。

在这里，我们已经删除从文本编辑的全面控制，并保持它在其他地方（即XML配置文件）和依赖性（即类拼写检查）被注入到类文本编辑通过类构造函数。因此，流程控制已经“倒”通过依赖注入（DI），因为已经有效地委派依赖一些外部系统。

依赖注入的第二种方法是通过文本编辑类，我们将创建拼写检查实例的setter方法​​，该实例将被用来调用setter方法​​来初始化文本编辑的属性。

因此，DI主要有两种变体和下面的两个子章将涵盖两者结合实例：

|  |  |
| --- | --- |
| **S.N.** | **依赖注入类型及说明** |
| 1 | [基于构造函数的依赖注入](http://www.yiibai.com/spring/constructor_based_dependency_injection.html) 当容器调用类的构造函数有多个参数，每个代表在其他类中的构造函数依赖关系为基础的DI来完成。 |
| 2 | [基于setter方法的依赖注入](http://www.yiibai.com/spring/setter_based_dependency_injection.html) 基于setterDI由容器调用setter方法​​对你的bean调用无参构造器或无参static工厂方法实例化bean之后完成的。 |

你可以混合两种，构造型和基于setter方法DI，但它是拇指使用构造函数的参数进行强制依赖和setter可选依赖的一个很好的规则。

代码是清洁器与DI原理，当对象被提供有它们的依赖性的去耦效果更明显。对象不查询它的依赖，以及不知道的依赖性的位置或类，而一切都由Spring框架关照。

# Spring注入内部bean

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**1**

[上一篇](http://www.yiibai.com/spring/setter_based_dependency_injection.html)[下一篇](http://www.yiibai.com/spring/spring_injecting_collection.html)

正如你所知道的Java内部类是其他类的范围内定义的，同样，内部bean是被其他bean的范围内定义的bean。因此<property/>或<constructor-arg/>元素内<bean/>元件被称为内部bean和它如下所示。

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="outerBean" class="...">

<property name="target">

<bean id="innerBean" class="..."/>

</property>

</bean>

</beans>

## 例如：

我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *TextEditor*, *SpellChecker* and *MainApp* under the *com.yiibai*package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是TextEditor.java文件的内容：

package com.yiibai;

public class TextEditor {

private SpellChecker spellChecker;

// a setter method to inject the dependency.

public void setSpellChecker(SpellChecker spellChecker) {

System.out.println("Inside setSpellChecker." );

this.spellChecker = spellChecker;

}

// a getter method to return spellChecker

public SpellChecker getSpellChecker() {

return spellChecker;

}

public void spellCheck() {

spellChecker.checkSpelling();

}

}

下面是另外一个相关的类文件SpellChecker.java内容：

package com.yiibai;

public class SpellChecker {

public SpellChecker(){

System.out.println("Inside SpellChecker constructor." );

}

public void checkSpelling(){

System.out.println("Inside checkSpelling." );

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

TextEditor te = (TextEditor) context.getBean("textEditor");

te.spellCheck();

}

}

以下是配置文件beans.xml文件里面有配置为基于setter 注入，但使用内部bean：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Definition for textEditor bean using inner bean -->

<bean id="textEditor" class="com.yiibai.TextEditor">

<property name="spellChecker">

<bean id="spellChecker" class="com.yiibai.SpellChecker"/>

</property>

</bean>

</beans>

创建源代码和bean配置文件来完成，让我们运行应用程序。如果一切顺利，这将打印以下信息：

Inside SpellChecker constructor.

Inside setSpellChecker.

Inside checkSpelling.

# Spring注入集合

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

您已经看到了如何配置基本数据类型使用value属性和使用<property>标签的ref属性在你的bean配置文件中的对象引用。这两种情况下处理过单值到一个bean。

现在什么样，如果你想通过多元价值，如Java Collection类型List, Set, Map 及 Properties。要处理这种情况，Spring提供了四种类型的如下集合的配置元素：

|  |  |
| --- | --- |
| **元素** | **描述** |
| <list> | 这有助于注入值列表List配线，使重复。 |
| <set> | 这有助于在配线的一组值，但不能重复。 |
| <map> | 这可用于注入的名称 - 值对，其中名称和值可以是任何类型的集合。 |
| <props> | 这可以用来注入的名称 - 值对，其中名称和值都是字符串的集合。 |

可以使用<list> 或<set> 来连接任何实现java.util.Collection或数组。

会遇到两种情况（a）将收集的直接的值及（b）传递一个bean的引用作为集合的元素之一。

## 例子:

我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *JavaCollection*, and *MainApp* under the *com.yiibai* package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是JavaCollection.java文件的内容：

package com.yiibai;

import java.util.\*;

public class JavaCollection {

List addressList;

Set addressSet;

Map addressMap;

Properties addressProp;

// a setter method to set List

public void setAddressList(List addressList) {

this.addressList = addressList;

}

// prints and returns all the elements of the list.

public List getAddressList() {

System.out.println("List Elements :" + addressList);

return addressList;

}

// a setter method to set Set

public void setAddressSet(Set addressSet) {

this.addressSet = addressSet;

}

// prints and returns all the elements of the Set.

public Set getAddressSet() {

System.out.println("Set Elements :" + addressSet);

return addressSet;

}

// a setter method to set Map

public void setAddressMap(Map addressMap) {

this.addressMap = addressMap;

}

// prints and returns all the elements of the Map.

public Map getAddressMap() {

System.out.println("Map Elements :" + addressMap);

return addressMap;

}

// a setter method to set Property

public void setAddressProp(Properties addressProp) {

this.addressProp = addressProp;

}

// prints and returns all the elements of the Property.

public Properties getAddressProp() {

System.out.println("Property Elements :" + addressProp);

return addressProp;

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

JavaCollection jc=(JavaCollection)context.getBean("javaCollection");

jc.getAddressList();

jc.getAddressSet();

jc.getAddressMap();

jc.getAddressProp();

}

}

以下是配置文件beans.xml文件里面有配置的集合的所有类型：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Definition for javaCollection -->

<bean id="javaCollection" class="com.yiibai.JavaCollection">

<!-- results in a setAddressList(java.util.List) call -->

<property name="addressList">

<list>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>USA</value>

</list>

</property>

<!-- results in a setAddressSet(java.util.Set) call -->

<property name="addressSet">

<set>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>USA</value>

</set>

</property>

<!-- results in a setAddressMap(java.util.Map) call -->

<property name="addressMap">

<map>

<entry key="1" value="INDIA"/>

<entry key="2" value="Pakistan"/>

<entry key="3" value="USA"/>

<entry key="4" value="USA"/>

</map>

</property>

<!-- results in a setAddressProp(java.util.Properties) call -->

<property name="addressProp">

<props>

<prop key="one">INDIA</prop>

<prop key="two">Pakistan</prop>

<prop key="three">USA</prop>

<prop key="four">USA</prop>

</props>

</property>

</bean>

</beans>

创建源代码和bean配置文件完成后，让我们运行应用程序。如果应用程序一切顺利，这将打印以下信息：

List Elements :[INDIA, Pakistan, USA, USA]

Set Elements :[INDIA, Pakistan, USA]

Map Elements :{1=INDIA, 2=Pakistan, 3=USA, 4=USA}

Property Elements :{two=Pakistan, one=INDIA, three=USA, four=USA}

## 注入Bean引用：

下面bean定义将帮助您了解如何注入bean的引用作为集合的元素之一。甚至可以混合引用和值都在一起，如下图所示：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- Bean Definition to handle references and values -->

<bean id="..." class="...">

<!-- Passing bean reference for java.util.List -->

<property name="addressList">

<list>

<ref bean="address1"/>

<ref bean="address2"/>

<value>Pakistan</value>

</list>

</property>

<!-- Passing bean reference for java.util.Set -->

<property name="addressSet">

<set>

<ref bean="address1"/>

<ref bean="address2"/>

<value>Pakistan</value>

</set>

</property>

<!-- Passing bean reference for java.util.Map -->

<property name="addressMap">

<map>

<entry key="one" value="INDIA"/>

<entry key ="two" value-ref="address1"/>

<entry key ="three" value-ref="address2"/>

</map>

</property>

</bean>

</beans>

使用上面的bean定义，需要定义这样一种方式，他们应该能够处理的参考，以及setter方法。

## 注入null和空字符串的值

如果需要传递一个空字符串作为值，如下所示：

<bean id="..." class="exampleBean">

<property name="email" value=""/>

</bean>

前面的例子等同于Java代码： exampleBean.setEmail("")

如果需要传递一个null值，如下所示：

<bean id="..." class="exampleBean">

<property name="email"><null/></property>

</bean>

前面的例子等同于Java代码：exampleBean.setEmail(null)

# Spring Beans自动装配

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_injecting_collection.html)[下一篇](http://www.yiibai.com/spring/spring_autowiring_byname.html)

前面已经学会如何使用的<bean>元素来声明bean和注入<bean>，通过使用在XML配置文件<constructor-arg>和<property>元素。

Spring容器可以自动装配相互协作bean之间的关系，这有助于减少对XML配置，而无需编写一个大的基于Spring应用程序的较多的<constructor-arg>和<property>元素。

## 自动装配模式：

有下列自动装配模式，可用于指示Spring容器使用自动装配依赖注入。使用<bean/>元素的autowire属性为一个bean定义中指定自动装配模式。

|  |  |
| --- | --- |
| **模式** | **描述** |
| no | This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter. |
| [byName](http://www.yiibai.com/spring/spring_autowiring_byname.html) | Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byName* in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file. |
| [byType](http://www.yiibai.com/spring/spring_autowiring_bytype.html) | Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byType* in the XML configuration file. It then tries to match and wire a property if its **type**matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown. |
| [constructor](http://www.yiibai.com/spring/spring_autowiring_byconstructor.html) | Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised. |
| autodetect | Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*. |

可以使用类型和constructor自动装配模式来连接数组和其他类型化的集合。

## 自动装配的局限性：

自动装配最好效果是它始终在一个项目中使用。如果自动装配不一般的使用，它可能会被混淆为开发人员可以使用它来连接只有一个或两个bean定义。不过，自动装配可以显著减少需要指定属性或构造器参数，但你应该使用它们之前考虑自动装配的局限性和缺点。

|  |  |
| --- | --- |
| **限制** | **描述** |
| 压倒一切的可能性 | 可以使用<constructor-arg>和<property>设置总是覆盖自动装配还指定依赖关系。 |
| 原始数据类型 | 不能自动装配所谓的简单类型包括基本类型，字符串和类。 |
| 混乱的本质 | 自动装配比显式装配确切的少，所以如果可能的话可以使用显式的连接。 |

# Spring基于注解配置

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**1**

[上一篇](http://www.yiibai.com/spring/spring_autowiring_byconstructor.html)[下一篇](http://www.yiibai.com/spring/spring_required_annotation.html)

从Spring2.5开始就有可能使用注释来配置依赖注入。而是采用XML来描述一个bean接线，你可以使用注解的相关类，方法或字段声明将bean配置到组件类本身。

注释注入在XML注入之前进行，因此后者的配置将覆盖前者通过两种方式连接的属性。

注释接线默认情况下不开启在Spring容器。所以，我们才可以使用基于注解的接线，我们将需要启用它在我们的Spring配置文件。因此，考虑到已在下列情况下，配置文件要使用的任何注释在Spring应用程序。

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

当<context:annotation-config/>配置后，您就可以开始注释代码表明，Spring自动连线的值到属性，方法和构造函数。让我们来看看几个重要的注解，以了解它们是如何工作的：

|  |  |
| --- | --- |
| **S.N.** | **注释与说明** |
| 1 | [@Required](http://www.yiibai.com/spring/spring_required_annotation.html) @Required注释适用于bean属性的setter方法​​。 |
| 2 | [@Autowired](http://www.yiibai.com/spring/spring_autowired_annotation.html) @Autowired 注释可以应用到bean属性的setter方法​​，非setter方法​​，构造函数和属性。 |
| 3 | [@Qualifier](http://www.yiibai.com/spring/spring_qualifier_annotation.html) @ Autowired随着@ Qualifier注释可以用来通过指定确切的bean将有线，除去混乱。 |
| 4 | [JSR-250 Annotations](http://www.yiibai.com/spring/spring_jsr250_annotations.html) Spring支持JSR-250的基础的注解，其中包括了@Resource，@PostConstruct和@PreDestroy注解。 |

# Spring 基于Java配置

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_jsr250_annotations.html)[下一篇](http://www.yiibai.com/spring/event_handling_in_spring.html)

到目前为止，您已经了解了如何配置使用XML配置文件的Spring bean。如果习惯使用XML配置，那么会说，这不是真的需要学习如何进行基于Java的配置，因为你要使用的配置或者可达到相同的结果。

基于Java配置选项，可以编写大多数的Spring不用配置XML，但有几个基于Java的注释的帮助下解释。

## @Configuration & @Bean 注解:

注释类与@Configuration表示这个类可以使用Spring IoC容器为bean定义的来源。在@Bean 注解告诉Spring的注解为@Bean的一个方法将返回应注册为在Spring应用程序上下文中的bean对象。最简单可行的@Configuration类将如下所示：

package com.yiibai;

import org.springframework.context.annotation.\*;

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

上面的代码将等同于下面的XML配置：

<beans>

<bean id="helloWorld" class="com.yiibai.HelloWorld" />

</beans>

下面注解为@Bean的方法名称作为工作bean的id，它创建并返回实际的bean。配置类可以有声明多个@Bean。一旦配置类定义，可以加载和提供他们使用AnnotationConfigApplicationContext 如下，以Spring容器：

public static void main(String[] args) {

ApplicationContext ctx =

new AnnotationConfigApplicationContext(HelloWorldConfig.class);

HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

helloWorld.setMessage("Hello World!");

helloWorld.getMessage();

}

可以加载各种配置类别如下：

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx =

new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

### 例子:

让我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Because you are using Java-based annotations, so you also need to add *CGLIB.jar* from your Java installation directory and *ASM.jar* library which can be downloaded from *asm.ow2.org*. |
| 4 | Create Java classes *HelloWorldConfig*, *HelloWorld* and *MainApp* under the *com.yiibai*package. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是HelloWorldConfig.java文件的内容：

package com.yiibai;

import org.springframework.context.annotation.\*;

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

这里是HelloWorld.java的文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.\*;

public class MainApp {

public static void main(String[] args) {

ApplicationContext ctx =

new AnnotationConfigApplicationContext(HelloWorldConfig.class);

HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

helloWorld.setMessage("Hello World!");

helloWorld.getMessage();

}

}

创建所有的源文件并添加所需的额外（外部）的库，让我们运行应用程序。应该注意，不需要配置文件。如果一切顺利，这将打印以下信息：

Your Message : Hello World!

## 注入Bean的依赖关系：

当@Bean对彼此的依赖，表达这种依赖很简单，只要有一个Bean的方法调用另一个如下：

package com.yiibai;

import org.springframework.context.annotation.\*;

@Configuration

public class AppConfig {

@Bean

public Foo foo() {

return new Foo(bar());

}

@Bean

public Bar bar() {

return new Bar();

}

}

在这里，Bean接收基准通过构造函数注入。现在，让我们看到一个正常工作的例子：

### 例子:

让我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Because you are using Java-based annotations, so you also need to add *CGLIB.jar* from your Java installation directory and *ASM.jar* library which can be downloaded from *asm.ow2.org*. |
| 4 | Create Java classes *TextEditorConfig*, *TextEditor*, *SpellChecker* and *MainApp* under the*com.yiibai* package. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是TextEditorConfig.java文件的内容：

package com.yiibai;

import org.springframework.context.annotation.\*;

@Configuration

public class TextEditorConfig {

@Bean

public TextEditor textEditor(){

return new TextEditor( spellChecker() );

}

@Bean

public SpellChecker spellChecker(){

return new SpellChecker( );

}

}

这里是TextEditor.java文件的内容：

package com.yiibai;

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor(SpellChecker spellChecker){

System.out.println("Inside TextEditor constructor." );

this.spellChecker = spellChecker;

}

public void spellCheck(){

spellChecker.checkSpelling();

}

}

下面是另外一个相关的类文件SpellChecker.java内容：

package com.yiibai;

public class SpellChecker {

public SpellChecker(){

System.out.println("Inside SpellChecker constructor." );

}

public void checkSpelling(){

System.out.println("Inside checkSpelling." );

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.\*;

public class MainApp {

public static void main(String[] args) {

ApplicationContext ctx =

new AnnotationConfigApplicationContext(TextEditorConfig.class);

TextEditor te = ctx.getBean(TextEditor.class);

te.spellCheck();

}

}

创建所有的源文件并添加所需的额外的库完成，让我们运行应用程序。应该注意，不需要配置文件。如果一切顺利，这将打印以下信息：

Inside SpellChecker constructor.

Inside TextEditor constructor.

Inside checkSpelling.

## @Import 注解:

@ import的注解允许加载@Bean从另一个配置类定义。考虑一个配置类，如下所示：

@Configuration

public class ConfigA {

@Bean

public A a() {

return new A();

}

}

您可以在另一个bean声明导入上述bean声明如下：

@Configuration

@Import(ConfigA.class)

public class ConfigB {

@Bean

public B a() {

return new A();

}

}

现在，不需要实例化的前提下，当同时指定配置A.class和配置B.class，只有Config B类需要如下提供：

public static void main(String[] args) {

ApplicationContext ctx =

new AnnotationConfigApplicationContext(ConfigB.class);

// now both beans A and B will be available...

A a = ctx.getBean(A.class);

B b = ctx.getBean(B.class);

}

## 生命周期回调：

@Bean注解支持指定任意的初始化和销毁​​回调方法，就像Spring的XML的初始化方法和bean元素销毁方法的属性：

public class Foo {

public void init() {

// initialization logic

}

public void cleanup() {

// destruction logic

}

}

@Configuration

public class AppConfig {

@Bean(initMethod = "init", destroyMethod = "cleanup" )

public Foo foo() {

return new Foo();

}

}

## 指定Bean的适用范围：

默认范围是单例，但可以使用@Scope注解来覆盖此如下：

@Configuration

public class AppConfig {

@Bean

@Scope("prototype")

public Foo foo() {

return new Foo();

}

}

# Spring事件处理

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/spring_java_based_configuration.html)[下一篇](http://www.yiibai.com/spring/custom_events_in_spring.html)

你已经看到了所有的章节了Spring的核心是ApplicationContext，它管理bean的整个生命周期。ApplicationContext中加载bean时发布某些类型的事件。例如，一个ContextStartedEvent是当所述上下文被启动并发布，上下文被停止ContextStoppedEvent。

事件处理中ApplicationContext 通过了ApplicationEvent类和ApplicationListener接口提供。所以，如果一个bean实现了ApplicationListener，然后每anApplicationEvent被发布到ApplicationContext的时候通知bean。

Spring提供了以下标准的事件：

|  |  |
| --- | --- |
| **S.N.** | **Spring 内置活动及说明** |
| 1 | **ContextRefreshedEvent** This event is published when the *ApplicationContext* is either initialized or refreshed. This can also be raised using the refresh() method on the *ConfigurableApplicationContext* interface. |
| 2 | **ContextStartedEvent** This event is published when the *ApplicationContext* is started using the start() method on the*ConfigurableApplicationContext* interface. You can poll your database or you can re/start any stopped application after receiving this event. |
| 3 | **ContextStoppedEvent** This event is published when the *ApplicationContext* is stopped using the stop() method on the *ConfigurableApplicationContext* interface. You can do required housekeep work after receiving this event. |
| 4 | **ContextClosedEvent** This event is published when the *ApplicationContext* is closed using the close() method on the*ConfigurableApplicationContext* interface. A closed context reaches its end of life; it cannot be refreshed or restarted. |
| 5 | **RequestHandledEvent** This is a web-specific event telling all beans that an HTTP request has been serviced. |

Spring事件处理是单线程的，所以如果一个事件被发布，直至及除非所有的接收器得到的消息，该进程被阻塞并且流程将不会继续。因此，应注意在设计应用程序时，如果事件处理被使用。

## 监听上下文事件：

要监听上下文事件，一个bean应该实现ApplicationListener接口，只有一个方法onApplicationEvent()。因此，让我们写一个例子，看看如何传播的事件，以及如何可以把代码来执行基于某些事件所需的任务。

让我们使用Eclipse IDE，然后按照下面的步骤来创建一个Spring应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld*, *CStartEventHandler*, *CStopEventHandler* and *MainApp*under the *com.yiibai* package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是HelloWorld.java 文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

以下是CStartEventHandler.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationListener;

import org.springframework.context.event.ContextStartedEvent;

public class CStartEventHandler

implements ApplicationListener<ContextStartedEvent>{

public void onApplicationEvent(ContextStartedEvent event) {

System.out.println("ContextStartedEvent Received");

}

}

以下是CStopEventHandler.java文件的内容：

package com.yiibai;

import org.springframework.context.ApplicationListener;

import org.springframework.context.event.ContextStoppedEvent;

public class CStopEventHandler

implements ApplicationListener<ContextStoppedEvent>{

public void onApplicationEvent(ContextStoppedEvent event) {

System.out.println("ContextStoppedEvent Received");

}

}

以下是MainApp.java文件的内容：

package com.yiibai;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ConfigurableApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

// Let us raise a start event.

context.start();

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

// Let us raise a stop event.

context.stop();

}

}

以下是配置文件beans.xml文件：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld">

<property name="message" value="Hello World!"/>

</bean>

<bean id="cStartEventHandler"

class="com.yiibai.CStartEventHandler"/>

<bean id="cStopEventHandler"

class="com.yiibai.CStopEventHandler"/>

</beans>

创建源程序和bean配置文件完成，让我们运行应用程序。如果一切顺利，这将打印以下信息：

ContextStartedEvent Received

Your Message : Hello World!

ContextStoppedEvent Received

如果喜欢，你可以发布自己的自定义事件，以后你可以捕捉相同处理那些自定义事件动作。如果你有兴趣在编写自己的自定义事件，可以查看 [Spring自定义事件](http://www.yiibai.com/spring/custom_events_in_spring.html)

# Spring框架的AOP

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/custom_events_in_spring.html)[下一篇](http://www.yiibai.com/spring/schema_based_aop_appoach.html)

Spring框架的关键组件是面向方面编程（AOP）框架。面向方面的编程不仅打破程序逻辑分成不同的部分称为所谓的担忧。跨越多个点的应用程序的功能被称为横切关注点和这些横切关注点是从应用程序的业务逻辑概念上区分开来。还有像日志记录，审计，声明性事务，安全性和高速缓存等方面的各种常见的好例子

模块化的OOP中的关键单元是类，而在AOP中模块化的单元则是切面。依赖注入可以帮助你从对方解耦应用程序对象和AOP可以帮助你从他们影响的对象分离横切关注点。 AOP是一样的编程语言如[Perl](http://www.yiibai.com/perl/)，.NET，[Java](http://www.yiibai.com/java/)和其他触发器。

Spring AOP模块提供了拦截器拦截的应用程序，例如，执行一个方法时，可以之前或之后执行的方法添加额外的功能。

## AOP术语：

在我们开始使用AOP之前，先熟悉AOP的概念和术语。这些条款是不特定于Spring，问题都是有关AOP。

|  |  |
| --- | --- |
| **术语** | **描述** |
| Aspect | A module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. |
| Join point | This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework. |
| Advice | This is the actual action to be taken either before or after the method execution. This is actual piece of code that is invoked during program execution by Spring AOP framework. |
| Pointcut | This is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples. |
| Introduction | An introduction allows you to add new methods or attributes to existing classes. |
| Target object | The object being advised by one or more aspects, this object will always be a proxied object. Also referred to as the advised object. |
| Weaving | Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime. |

## 建议的类型

Spring方面可以用5种下面提到的建议：

|  |  |
| --- | --- |
| **Advice** | **描述** |
| before | 在一个方法执行之前运行的建议。 |
| after | 该方法的执行，无论其结果如何运行后建议。 |
| after-returning | 只有当方法成功地完成了一个方法执行后运行的建议。 |
| after-throwing | 在一个方法执行后运行建议只有当方法因抛出一个异常。 |
| around | 建议的方法之前和之后运行建议被调用。 |

## 自定义方面实现

Spring支持@AspectJ注解风格的方法，并实现自定义方面的架构为基础的方法。这两种方法已被详细地在下面的两个子章节解释

|  |  |
| --- | --- |
| **方法** | **描述** |
| [基于XML Schema](http://www.yiibai.com/spring/schema_based_aop_appoach.html) | 切面可以使用类以及基于XML的配置来实现。 |
| [基于@AspectJ](http://www.yiibai.com/spring/aspectj_based_aop_appoach.html) | @ AspectJ是指声明方面的风格注释的使用Java 5注释普通的Java类。 |

# Spring JDBC框架

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**3**

[上一篇](http://www.yiibai.com/spring/aspectj_based_aop_appoach.html)[下一篇](http://www.yiibai.com/spring/spring_jdbc_example.html)

同时与数据库使用普通的旧JDBC的工作，它变得繁琐写不必要的代码来处理异常，打开和关闭数据库连接等，但Spring的JDBC框架需要的所有低层次细节从打开连接，准备和执行SQL语句，过程异常，处理事务，最后关闭连接。

所以，你所要做的只是定义连接参数，并指定要执行的SQL语句，并做必要的工作，在每次迭代时从数据库中获取数据。

Spring JDBC提供了一些方法和相应不同的类与数据库进行交互。我要采取经典和最流行的做法，利用JdbcTemplateclass框架。这是管理的所有数据库的通信和异常处理中心框架类。

## JdbcTemplate 类

JdbcTemplate类执行SQL查询，更新语句和存储过程调用，在结果集和提取返回参数值进行迭代。它还捕捉JDBC的异常并将其转换为通用的，信息更丰富，除了在org.springframework.dao包中定义的层次结构。

JdbcTemplate类的实例是一次配置的线程。所以，你可以配置一个JdbcTemplate的一个实例，然后安全地注入这种共享引用到多个DAO。

使用JdbcTemplate类时，通常的做法是配置一个DataSource在Spring配置文件，然后依赖关系注入该共享数据源豆到DAO类，JdbcTemplate或者是在setter数据源创建。

## 配置数据源

让我们一起创造数据库test数据库表的 student 。假设使用MySQL数据库，如果使用其他数据库，那么可以相应地改变你的DDL和SQL查询。

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT,

NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL,

PRIMARY KEY (ID)

);

现在，我们需要提供一个数据源给JdbcTemplate类，因此它可以自行配置，以获得数据库访问。您可以配置数据源的XML文件中有一段代码，如下图所示：

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/TEST"/>

<property name="username" value="root"/>

<property name="password" value="password"/>

</bean>

## 数据访问对象 (DAO)

DAO表示这是通常用于数据库交互的数据访问对象。 DAO的存在是为了提供读取和写入数据到数据库中，他们应该通过该应用程序的其余部分将访问它们的接口公开此功能的一种手段。

在Spring的数据访问对象（DAO）的支持使得它很容易与如[JDBC](http://www.yiibai.com/jdbc/)，[Hibernate](http://www.yiibai.com/hibernate/)，JPA和JDO以一致的方式进行数据访问技术。

## 执行SQL语句

让我们来看看如何使用SQL和的JdbcTemplate对象数据库中的表执行CRUD（创建，读取，更新和删除）操作。

查询一个整数：

String SQL = "select count(\*) from Student";

int rowCount = jdbcTemplateObject.queryForInt( SQL );

查询长整数：

String SQL = "select count(\*) from Student";

long rowCount = jdbcTemplateObject.queryForLong( SQL );

使用绑定变量的简单查询：

String SQL = "select age from Student where id = ?";

int age = jdbcTemplateObject.queryForInt(SQL, new Object[]{10});

在查询字符串：

String SQL = "select name from Student where id = ?";

String name = jdbcTemplateObject.queryForObject(SQL, new Object[]{10}, String.class);

查询并返回一个对象：

String SQL = "select \* from Student where id = ?";

Student student = jdbcTemplateObject.queryForObject(SQL,

new Object[]{10}, new StudentMapper());

public class StudentMapper implements RowMapper<Student> {

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student student = new Student();

student.setID(rs.getInt("id"));

student.setName(rs.getString("name"));

student.setAge(rs.getInt("age"));

return student;

}

}

查询并返回多个对象：

String SQL = "select \* from Student";

List<Student> students = jdbcTemplateObject.query(SQL,

new StudentMapper());

public class StudentMapper implements RowMapper<Student> {

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student student = new Student();

student.setID(rs.getInt("id"));

student.setName(rs.getString("name"));

student.setAge(rs.getInt("age"));

return student;

}

}

插入一行到表：

String SQL = "insert into Student (name, age) values (?, ?)";

jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );

更新一行到表：

String SQL = "update Student set name = ? where id = ?";

jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );

从表中删除行：

String SQL = "delete Student where id = ?";

jdbcTemplateObject.update( SQL, new Object[]{20} );

## 执行DDL语句

您可以使用execute(...)方法的JdbcTemplate来执行任何SQL语句或DDL语句。下面是一个示例使用CREATE语句创建一个表：

String SQL = "CREATE TABLE Student( " +

"ID INT NOT NULL AUTO\_INCREMENT, " +

"NAME VARCHAR(20) NOT NULL, " +

"AGE INT NOT NULL, " +

"PRIMARY KEY (ID));"

jdbcTemplateObject.execute( SQL );

## Spring JDBC框架的例子：

基于上述理念，让我们检查，这将有助于您了解在Spring JDBC框架的使用几个重要的例子：

|  |  |
| --- | --- |
| **S.N.** | **例子& 描述** |
| 1 | [Spring JDBC 例子](http://www.yiibai.com/spring/spring_jdbc_example.html) 这个例子将说明如何编写一个简单的一个基于JDBC的Spring应用程序。 |
| 2 | [SpringSQL存储过程](http://www.yiibai.com/spring/calling_stored_procedure.html) 了解如何在Spring中使用JDBC调用SQL存储过程。 |

# Spring 事务管理

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**0**

[上一篇](http://www.yiibai.com/spring/calling_stored_procedure.html)[下一篇](http://www.yiibai.com/spring/programmatic_management.html)

数据库事务是被当作单个工作单元的操作序列。这些操作要么全部完成或全部不成功。事务管理是面向企业应用程序，以确保数据的完整性和一致性RDBMS中的重要组成部分。事务的概念可以用下面的描述为ACID四个关键属性来描述：

1. 原子性: 一个事务应该被视为单个操作单元表示的操作的任一整个序列是成功的或不成功的。
2. 一致性: 这代表了数据库的参照完整性，在桌等唯一主键的一致性
3. 隔离性: 可能有很多事务处理相同的数据集的同时，每个事务都应由他人隔离，以防止数据损坏。
4. 持久性: 一旦事务完成，本次事务的结果必须作出永久性的，不能从数据库中删除因系统故障。

一个真正的RDBMS数据库系统将保证所有的四个属性为每个事务。颁发给使用SQL数据库的事务的简单观点如下：

* 使用begin transaction命令开始事务。
* 使用SQL查询执行各种删除，更新或插入操作。
* 如果所有的操作都成功，那么执行提交，否则回滚所有操作。

Spring框架提供的不同的底层事务管理API之上的抽象层。在Spring的事务支持，旨在通过增加事务功能，的POJO提供EJB的替代品事务。 Spring支持两种编程式和声明式事务管理。需要的EJB应用程序服务器，但Spring事务管理，而不需要一个应用服务器来实现。

## 局部与全局事务

局部事务是针对像一个JDBC连接一个单一的事务性资源，而全局事务可以跨越像事务多个事务资源的分布式系统。

局部事务管理可以在一个集中式计算环境下的应用程序的组件和资源都位于一个单一的网站是有用的，而事务管理只涉及一个单独的机器上运行的本地数据管理。局部事务更容易实现。

全局事务管理，需要在分布在多个系统中的所有资源的分布式计算环境。在这种情况下，事务管理既需要在地方和全局层面的工作要做。一个分布式或全局事务在多个系统上执行，其执行需要全局事务管理系统和所有相关系统的所有局部数据管理人员之间的协调。

## 编程与声明

Spring支持两种类型的事务管理：

1. [编程式事务管理:](http://www.yiibai.com/spring/programmatic_management.html) Spring支持两种类型的事务管理：
2. [声明式事务管理:](http://www.yiibai.com/spring/declarative_management.html) 这意味着你的业务代码分开事务管理。你只用注释或基于XML 配置来管理事务。

声明式事务管理要优于编程式事务管理，虽然它比编程式事务管理，它允许你通过代码来控制事务不够灵活。但作为一种横切关注点的，声明式事务管理可以用模块化的AOP方法。Spring通过Spring AOP框架支持声明式事务管理。

## Spring事务抽象

关键Spring事务抽象的定义是org.springframework.transaction.PlatformTransactionManager接口，如下所示：

public interface PlatformTransactionManager {

TransactionStatus getTransaction(TransactionDefinition definition);

throws TransactionException;

void commit(TransactionStatus status) throws TransactionException;

void rollback(TransactionStatus status) throws TransactionException;

}

|  |  |
| --- | --- |
| **S.N.** | **方法 & 描述** |
| 1 | **TransactionStatus getTransaction(TransactionDefinition definition)** This method returns a currently active transaction or create a new one, according to the specified propagation behavior. |
| 2 | **void commit(TransactionStatus status)** This method commits the given transaction, with regard to its status. |
| 3 | **void rollback(TransactionStatus status)** This method performs a rollback of the given transaction. |

TransactionDefinition是在Spring中事务支持的核心接口，它的定义如下：

public interface TransactionDefinition {

int getPropagationBehavior();

int getIsolationLevel();

String getName();

int getTimeout();

boolean isReadOnly();

}

|  |  |
| --- | --- |
| **S.N.** | **方法 & 描述** |
| 1 | **int getPropagationBehavior()** This method returns the propagation behavior. Spring offers all of the transaction propagation options familiar from EJB CMT. |
| 2 | **int getIsolationLevel()** This method returns the degree to which this transaction is isolated from the work of other transactions. |
| 3 | **String getName()** This method returns the name of this transaction. |
| 4 | **int getTimeout()** This method returns the time in seconds in which the transaction must complete. |
| 5 | **boolean isReadOnly()** This method returns whether the transaction is read-only. |

下面是隔离级别可能的值：

|  |  |
| --- | --- |
| **S.N.** | **隔离& 描述** |
| 1 | **TransactionDefinition.ISOLATION\_DEFAULT** This is the default isolation level. |
| 2 | **TransactionDefinition.ISOLATION\_READ\_COMMITTED** Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur. |
| 3 | **TransactionDefinition.ISOLATION\_READ\_UNCOMMITTED** Indicates that dirty reads, non-repeatable reads and phantom reads can occur. |
| 4 | **TransactionDefinition.ISOLATION\_REPEATABLE\_READ** Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur. |
| 5 | **TransactionDefinition.ISOLATION\_SERIALIZABLE** Indicates that dirty reads, non-repeatable reads and phantom reads are prevented. |

以下是针对Propagation 类型可能的值：

|  |  |
| --- | --- |
| **S.N.** | **传输& 描述** |
| 1 | **TransactionDefinition.PROPAGATION\_MANDATORY** Support a current transaction; throw an exception if no current transaction exists. |
| 2 | **TransactionDefinition.PROPAGATION\_NESTED** Execute within a nested transaction if a current transaction exists. |
| 3 | **TransactionDefinition.PROPAGATION\_NEVER** Do not support a current transaction; throw an exception if a current transaction exists. |
| 4 | **TransactionDefinition.PROPAGATION\_NOT\_SUPPORTED** Do not support a current transaction; rather always execute non-transactionally. |
| 5 | **TransactionDefinition.PROPAGATION\_REQUIRED** Support a current transaction; create a new one if none exists. |
| 6 | **TransactionDefinition.PROPAGATION\_REQUIRES\_NEW** Create a new transaction, suspending the current transaction if one exists. |
| 7 | **TransactionDefinition.PROPAGATION\_SUPPORTS** Support a current transaction; execute non-transactionally if none exists. |
| 8 | **TransactionDefinition.TIMEOUT\_DEFAULT** Use the default timeout of the underlying transaction system, or none if timeouts are not supported. |

TransactionStatus接口为处理事务的代码来控制事务执行和查询事务状态的简单方法。

public interface TransactionStatus extends SavepointManager {

boolean isNewTransaction();

boolean hasSavepoint();

void setRollbackOnly();

boolean isRollbackOnly();

boolean isCompleted();

}

|  |  |
| --- | --- |
| **S.N.** | **方法 & 描述** |
| 1 | **boolean hasSavepoint()** This method returns whether this transaction internally carries a savepoint, that is, has been created as nested transaction based on a savepoint. |
| 2 | **boolean isCompleted()** This method returns whether this transaction is completed, that is, whether it has already been committed or rolled back. |
| 3 | **boolean isNewTransaction()** This method returns true in case the present transaction is new. |
| 4 | **boolean isRollbackOnly()** This method returns whether the transaction has been marked as rollback-only. |
| 5 | **void setRollbackOnly()** This method sets the transaction rollback-only. |

# Spring MVC框架教程

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**10**

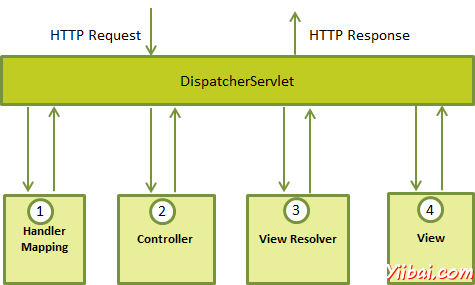
[上一篇](http://www.yiibai.com/spring/declarative_management.html)[下一篇](http://www.yiibai.com/spring/spring_mvc_hello_world_example.html)

在Spring的Web MVC框架提供了模型 - 视图 - 控制器架构以及可用于开发灵活，松散耦合的Web应用程序准备的组件。 MVC模式会导致分离的应用程序（输入逻辑，业务逻辑和UI逻辑）的不同方面，同时提供这些元素之间的松耦合。

* 模型（Model ）封装了应用程序的数据和一般他们会组成的POJO。
* 视图（View）是负责呈现模型数据和一般它生成的HTML输出，客户端的浏览器能够解释。
* 控制器（Controller ）负责处理用户的请求，并建立适当的模型，并把它传递给视图渲染。

## DispatcherServlet

Spring的web模型 - 视图 - 控制器（MVC）框架是围绕着处理所有的HTTP请求和响应的DispatcherServlet的设计。 Spring的Web MVC框架的DispatcherServlet的请求处理流程说明如下图：



下面是对应于传入的HTTP请求到DispatcherServlet的事件序列：

1. 接收HTTP请求后，DispatcherServlet 咨询 HandlerMapping 来调用相应的控制器。
2. 该控制器接受请求并调用基于使用GET或POST方法相应的服务方法。服务方法将基于定义的业务逻辑设置模型数据，并返回视图名到DispatcherServlet。
3. DispatcherServlet将需要帮助的ViewResolver从拾取到该请求所定义的视图。
4. 一旦视图被敲定，DispatcherServlet会传递模型数据是在浏览器上最终呈现的视图。

所有上述部件，即HandlerMapping，控制器和视图解析器WebApplicationContext 部分是纯的 ApplicationContext 必要的 Web应用程序的一些额外的功能扩展。

## 所需配置

你需要映射所需的DispatcherServlet处理，通过在web.xml文件中使用URL映射请求。下面是一个例子，说明声明和映射 HelloWeb DispatcherServlet 的例子：

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>Spring MVC Application</display-name>

<servlet>

<servlet-name>HelloWeb</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>HelloWeb</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

</web-app>

web.xml 文件将被保存在您的Web应用程序的 WebContent/ WEB-INF 目录。 在 DispatcherServlet 的 HelloWeb 初始化，该框架将尝试从一个名为 [servlet-name]-servlet.xml位于应用程序 WebContent/WEB-INF 目录文件加载应用程序上下文。在这种情况下我们的文件将是HelloWeb-servlet.xml。

接下来，<servlet-mapping>标记指示URL会被DispatcherServlet处理。这里全部用。jsp结束HTTP请求将由DispatcherServlet的HelloWeb处理。

如果不想用默认文件名为[servlet-name]-servlet.xml和默认位置的WebContent/WEB-INF，可以通过添加servlet 的 listenerContextLoaderListener 在 web.xml 文件中定义该文件的名称和位置如下：

<web-app...>

<!-------- *DispatcherServlet* definition goes here----->

....

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

现在，让我们检查所需配置的HelloWeb-servlet.xml文件，放置在 Web应用程序的WebContent/WEB-INF目录：

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="com.yiibai" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

</beans>

下面是有关 HelloWeb-servlet.xml 文件的要点：

* [servlet-name]-servlet.xml 文件将被用于创建定义的Bean，会覆盖在全局范围里名字相同的Bean的定义。
* <context:component-scan...>标签将使用启动Spring MVC的注解扫描功能，允许做出像 @Controller和使用@RequestMapping注解等使用。
* 使用InternalResourceViewResolver将有定义来解析视图名的规则。按照上述定义的规则，命名为你好的逻辑视图被委托给一个视图实现位于/WEB-INF/jsp/hello.jsp。

下一节将学习如何创建实际组件IE浏览器。控制器，模型和视图。

## 定义控制器

DispatcherServlet 委托请求发送到控制器，以执行特定于它的功能。注解@Controller表示一个特定的类提供一个控制器的角色。注解@RequestMapping 用于将URL映射到任何一个类或者一个特定的处理方法。

@Controller

@RequestMapping("/hello")

public class HelloController{

@RequestMapping(method = RequestMethod.GET)

public String printHello(ModelMap model) {

model.addAttribute("message", "Hello Spring MVC Framework!");

return "hello";

}

}

注解@Controller类定义为一个Spring MVC控制器。在这里，使用@RequestMapping第一次使用表明，该控制器上的所有处理方法是相对于/hello 路径。下一步标注使用@RequestMapping（方法= RequestMethod.GET）用于声明printHello() 方法作为控制器的默认服务方法来处理HTTP GET请求。可以定义另一种方法来处理同一URL的POST请求。

可以写在上面的另一种形式，可以使用@RequestMapping添加额外的属性如下控制：

@Controller

public class HelloController{

@RequestMapping(value = "/hello", method = RequestMethod.GET)

public String printHello(ModelMap model) {

model.addAttribute("message", "Hello Spring MVC Framework!");

return "hello";

}

}

value属性指示该处理方法所映射到的URL和method属性定义了服务的方法来处理HTTP GET请求。有以下重要点要注意关于上述定义的控制器：

* 将定义所需的业务逻辑内部的服务方法。可以调用此方法在另一个方法按要求。
* 基于定义的业务逻辑，将在此方法中创建一个模型。您可以设定器不同的模型的属性和这些属性将被视图访问提出的最终结果。此示例创建一个具有自己的属性“message”的模型。
* 定义的服务方法可以返回一个字符串，其中包含要用于渲染模型视图的名称。这个例子返回“hello”作为逻辑视图名。

## 创建JSP视图

Spring MVC支持多种类型的视图不同的演示技术。这些措施包括：JSP，HTML，PDF，Excel，XML，Velocity，XSLT，JSON，Atom和RSS订阅，JasperReports等，但最常用JSTL JSP模板。因此，我们编写一个/WEB-INF/hello/hello.jsp简单的hello 视图：

<html>

<head>

<title>Hello Spring MVC</title>

</head>

<body>

<h2>${message}</h2>

</body>

</html>

在这里， ${message}是我们已经建立了控制器内部的属性。你可以在视图中显示多个属性。

## Spring Web MVC框架的例子：

基于上述理念，让我们检查，这将有助于建立Spring Web 应用程序几个重要的例子：

|  |  |
| --- | --- |
| **S.N.** | **实例& 描述** |
| 1 | [Spring MVC Hello World Example](http://www.yiibai.com/spring/spring_mvc_hello_world_example.html) 这个例子将说明如何编写一个简单的Spring Web Hello World应用程序。 |
| 2 | [Spring MVC Form Handling Example](http://www.yiibai.com/spring/spring_mvc_form_handling_example.html) 这个例子将说明如何使用HTML表单将数据提交到控制器和显示回处理结果写一个Spring Web应用程序。 |
| 3 | [Spring Page Redirection Example](http://www.yiibai.com/spring/spring_page_redirection_example.html) 了解如何在Spring MVC框架中使用页面重定向功能。 |
| 4 | [Spring Static Pages Example](http://www.yiibai.com/spring/spring_static_pages_example.html) 了解如何访问静态网页与动态网页在Spring MVC框架。 |
| 5 | [Spring Exception Handling Example](http://www.yiibai.com/spring/spring_exception_handling_example.html) 了解如何处理Spring MVC框架异常。 |

# Spring 日志 Log4J

 来源：翻译整理    **注：**由于水平有限，欢迎指正。转载请务必注明出处。

**2**

[上一篇](http://www.yiibai.com/spring/spring_exception_handling_example.html)

Spring应用是非常容易使用Log4J里面的功能。下面的例子将带你通过简单的步骤来解释[Log4j](http://www.yiibai.com/log4j/)和Spring之间的简单集成。

我假设你已经有你的机器上安装log4J日志，如果你没有，那么你可以从 <http://logging.apache.org/> 下载并简单地提取任意文件夹的压缩文件。我们将使用唯一的log4j-x.y.z.jar在项目中。

接下来，使用Eclipse IDE，并按照以下步骤使用Spring Web框架开发动态表单的Web应用程序：

|  |  |
| --- | --- |
| **步骤** | **描述** |
| 1 | Create a project with a name *SpringExample* and create a package *com.yiibai* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Add log4j library *log4j-x.y.z.jar* as well in your project using using *Add External JARs*. |
| 4 | Create Java classes *HelloWorld* and *MainApp* under the *com.yiibai* package. |
| 5 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 6 | Create log4J configuration file *log4j.properties* under the **src** folder. |
| 7 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

这里是HelloWorld.java的文件的内容：

package com.yiibai;

public class HelloWorld {

private String message;

public void setMessage(String message){

this.message = message;

}

public void getMessage(){

System.out.println("Your Message : " + message);

}

}

下面是第二个文件MainApp.java的内容：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.log4j.Logger;

public class MainApp {

static Logger log = Logger.getLogger(MainApp.class.getName());

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

log.info("Going to create HelloWord Obj");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

log.info("Exiting the program");

}

}

可以生成，因为我们已经产生信息的消息调试和错误消息类似的方式。现在，让我们来看看 beans.xml 文件的内容：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="helloWorld" class="com.yiibai.HelloWorld">

<property name="message" value="Hello World!"/>

</bean>

</beans>

下面是log4j.properties 文件的内容定义，产生的日志消息所需的 Log4J 的标准规则：

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

# Set the name of the file

log4j.appender.FILE.File=C:\\log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite

log4j.appender.FILE.Append=false

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

创建源代码和bean配置文件完成后，让我们运行应用程序。如果您的应用程序一切顺利，这将打印以下信息在Eclipse的控制台：

Your Message : Hello World!

有时，如果查看 C:\\ drive，那么应该找到日志文件log.out各种日志消息，内容如下：

<!-- initialization log messages -->

Going to create HelloWord Obj

Returning cached instance of singleton bean 'helloWorld'

Exiting the program

## Jakarta Commons记录（JCL）API

另外，您可以使用Jakarta Commons Logging (JCL) API来在Spring应用程序生成的日志。 JCL可以从<http://jakarta.apache.org/commons/logging/>下载。我们在技术上需要此软件包的唯一文件是commons-logging-x.y.z.jar文件，该文件需要被放置在classpath类似的方式，已经把 log4j-x.y.z.jar 放入在上面的例子。

要使用需要一个org.apache.commons.logging.Log对象的日志记录功能，然后可以调用下面的方法之一根据您的需求：

* fatal(Object message)
* error(Object message)
* warn(Object message)
* info(Object message)
* debug(Object message)
* trace(Object message)

下面是MainApp.java的取代，这使得使用JCL API：

package com.yiibai;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.apache.commons.logging. Log;

import org.apache.commons.logging. LogFactory;

public class MainApp {

static Log log = LogFactory.getLog(MainApp.class.getName());

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

log.info("Going to create HelloWord Obj");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

log.info("Exiting the program");

}

}

你已经确定你包括编译和运行程序之前在项目中共享记录 - xyzjar文件。

现在，保持配置和上面的例子中其它的内容不变，如果编译并运行应用程序会得到类似的结果。