



Spring教程

极客学院出版

前言

Spring 框架是一个开源的 Java 平台，它为容易而快速的开发出耐用的 Java 应用程序提供了全面的基础设施。

Spring 框架最初是由 Rod Johnson 编写的，并且 2003 年 6 月首次在 Apache 2.0 许可下发布。

本教程是基于在 2015 年 3 月发布的 Spring 框架 4.1.6 版本编写的。

适用人群

本教程是为需要详细了解 Spring 框架的体系结构和实际应用的 Java 程序员设计的。本教程将带你达到中级的专业知识水平，而你可以将自己提升至更高层次的专业知识水平。

学习前提

在进行本教程之前，你应该对 Java 编程语言有一个很好的了解。对 Eclipse IDE 的基本了解也是必须的，因为所有的示例都是使用 Eclipse IDE 进行编译的。

更新日期	更新内容
2015-06-18	Spring 教程

目录

前言	1
第 1 章 概述	4
第 1 章 依赖注入 (DI)	6
第 2 章 体系结构	8
第 3 章 环境设置	11
第 4 章 Hello World 实例	16
第 5 章 IoC 容器	24
Spring 的 BeanFactory 容器	27
Spring ApplicationContext 容器	29
第 6 章 Bean 定义	32
第 7 章 Bean 的作用域	35
第 8 章 Bean 的生命周期	40
第 9 章 Spring——Bean 后置处理器	45
第 10 章 Bean 定义继承	49
第 11 章 依赖注入	54
Spring 基于构造函数的依赖注入	57
Spring 基于设值函数的依赖注入	60
第 12 章 注入内部 Beans	64
第 13 章 注入集合	68
第 14 章 Beans 自动装配	74
Spring 自动装配 ‘byName’	76
Spring 自动装配 ‘byType’	79

	Spring 由构造函数自动装配	82
第 15 章	基于注解的配置	85
	Spring @Required 注释	88
	Spring @Autowired 注释	91
	Spring @Qualifier 注释	96
	Spring JSR-250 注释	99
第 16 章	基于 Java 的配置	102
第 17 章	Spring 中的事件处理	110
第 18 章	Spring 中的自定义事件	115
第 19 章	Spring 框架的 AOP	119
	Spring 中基于 AOP 的 XML 架构	122
	Spring 中基于 AOP 的 @AspectJ	129
第 20 章	JDBC 框架概述	135
	Spring JDBC 示例	140
	Spring 中 SQL 的存储过程	146
第 21 章	事务管理	152
	Spring 编程式事务管理	159
	Spring 声明式事务管理	165
第 22 章	MVC 框架教程	171
	Spring MVC Hello World 例子	178
	Spring MVC 表单处理例子	181
	Spring 页面重定向例子	186
	Spring 静态页面例子	190
	Spring 异常处理例子	194
第 23 章	使用 Log4J 记录日志	201



T



1

概述



Spring 是最受欢迎的企业级 Java 应用程序开发框架。数以百万的来自世界各地的开发人员使用 Spring 框架来创建好性能、易于测试、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 编写的，并且 2003 年 6 月首次在 Apache 2.0 许可下发布。

当谈论到大小和透明度时，Spring 是轻量级的。Spring 框架的基础版本是在 2 MB 左右的。

Spring 框架的核心特性可以用于开发任何 Java 应用程序，但是在 Java EE 平台上构建 web 应用程序是需要扩展的。Spring 框架的目标是使 J2EE 开发变得更容易使用，通过启用基于 POJO 编程模型来促进良好的编程实践。

使用 Spring 框架的好处

下面列出的是使用 Spring 框架主要的好处：

- Spring 可以使开发人员使用 POJOs 开发企业级的应用程序。只使用 POJOs 的好处是你不需要一个 EJB 容器产品，比如一个应用程序服务器，但是你可以选择使用一个健壮的 servlet 容器，比如 Tomcat 或者一些商业产品。
- Spring 在一个单元模式中是有组织的。即使包和类的数量非常大，你必须并且只需要但是你需要的，而忽略剩余的那部分。
- Spring 不会让你白费力气坐重复工作，它真正的利用了一些现有的技术，像几个 ORM 框架、日志框架、JEE、Quartz 和 JDK 计时器，其他视图技术。
- 测试一个用 Spring 编写的应用程序很容易，因为 environment-dependent 代码被放进了这个框架中。此外，通过使用 JavaBean-style POJOs，它在使用依赖注入注入测试数据时变得更容易。
- Spring 的 web 框架是一个设计良好的 web MVC 框架，它为 web 框架，比如 Struts 或者其他工程上的或者很少受欢迎的 web 框架，提供了一个很好的供替代的选择。
- 为将特定技术的异常（例如，由 JDBC、Hibernate，或者 JDO 抛出的异常）翻译成一致的，Spring 提供了一个方便的 API，而这些都是未经检验的异常。
- 轻量级的 IOC 容器往往是轻量级的，例如，特别是当与 EJB 容器相比的时候。这有利于在内存和 CPU 资源有限的计算机上开发和部署应用程序。
- Spring 提供了一个一致的事务管理界面，该界面可以缩小成一个本地事务（例如，使用一个单一的数据库）和扩展成一个全局事务（例如，使用 JTA）。



第1章 依赖注入 (DI)



Spring 最认同的技术是控制反转的依赖注入 (DI) 模式。控制反转 (IoC) 是一个通用的概念，它可以用许多不同的方式去表达，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 应用程序时，应用程序类应该尽可能的独立于其他的 Java 类来增加这些类可重用额可能性，当进行单元测试时，可以使它们独立于其他类进行测试。依赖注入（或者有时被称为配线）有助于将这些类粘合在一起，并且在同一时间让它们保持独立。

到底什么是依赖注入？让我们将这两个词分开来看一看。这里将依赖关系部分转化为两个类之间的关联。例如，类 A 依赖于类 B。现在，让我们看一看第二部分，注入。所有这一切都意味着类 B 将通过 IoC 被注入到类 A 中。

依赖注入可以以向构造函数传递参数的方式发生，或者通过使用 setter 方法 post-construction。由于依赖注入是 Spring 框架的核心部分，所以我将在一个单独的章节中利用很好的例子去解释这一概念。

面向方面的程序设计 (AOP) :

Spring 框架的一个关键组件是面向方面的程序设计 (AOP) 框架。一个程序中跨越多个点的功能被称为横切关注点，这些横切关注点在概念上独立于应用程序的业务逻辑。有各种各样常见的很好的关于方面的例子，比如日志记录、声明性事务、安全性，和缓存等等。

在 OOP 中模块化的关键单元是类，而在 AOP 中模块化的关键单元是方面。AOP 帮助你横切关注点从它们所影响的对象中分离出来，然而依赖注入帮助你应用程序对象从彼此中分离出来。

Spring 框架的 AOP 模块提供了面向方面的程序设计实现，允许你定义拦截器方法和切入点，可以实现将应该被分开的代码干净的分开功能。我将在一个独立的章节中讨论更多关于 Spring AOP 的概念。

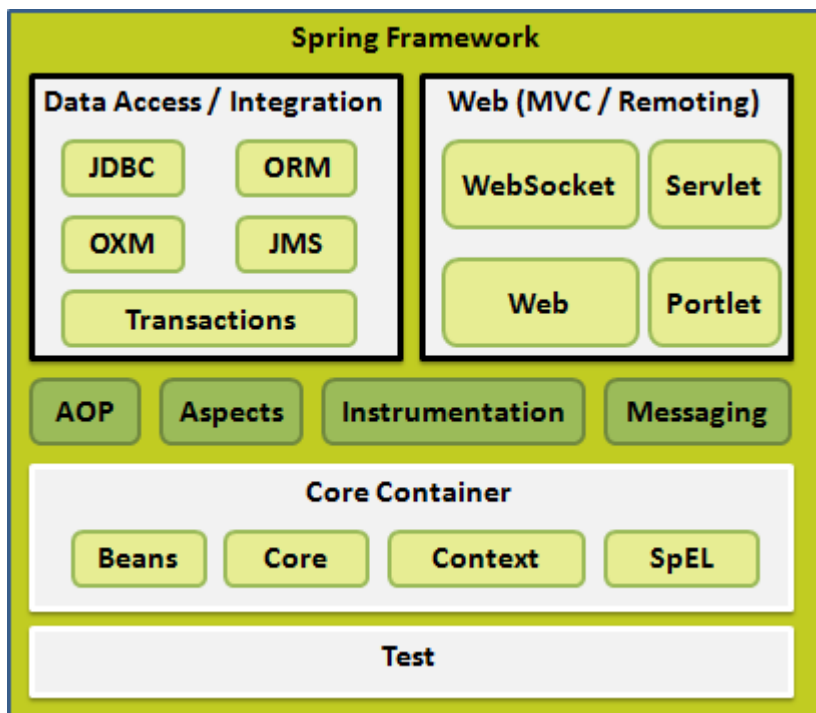


体系结构



Spring 有可能成为所有企业应用程序的一站式服务点，然而，Spring 是模块化的，允许你挑选和选择适用于你的模块，不必要把剩余部分也引入。下面的部分对在 Spring 框架中所有可用的模块给出了详细的介绍。

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

核心容器由核心，Bean，上下文和表达式语言模块组成，它们的细节如下：

- 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，它是一个工厂模式的复杂实现。
- 上下文模块建立在由核心和 Bean 模块提供的坚实基础上，它是访问定义和配置的任何对象的媒介。ApplicationContext 接口是上下文模块的重点。
- 表达式语言模块在运行时提供了查询和操作一个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，ORM，OXM，JMS 和事务处理模块，它们的细节如下：

- JDBC 模块提供了删除冗余的 JDBC 相关编码的 JDBC 抽象层。

- ORM 模块为流行的对象关系映射 API，包括 JPA，JDO，Hibernate 和 iBatis，提供了集成层。
- OXM 模块提供了抽象层，它支持对 JAXB，Castor，XMLBeans，JiBX 和 XStream 的对象/XML 映射实现。
- Java 消息服务 JMS 模块包含生产和消费的信息的功能。
- 事务模块为实现特殊接口的类及所有的 POJO 支持编程式和声明式事务管理。

Web

Web 层由 Web，Web-MVC，Web-Socket 和 Web-Portlet 组成，它们的细节如下：

- Web 模块提供了基本的面向 web 的集成功能，例如多个文件上传的功能和使用 servlet 监听器和面向 web 应用程序的上下文来初始化 IoC 容器。
- Web-MVC 模块包含 Spring 的模型-视图-控制器（MVC），实现了 web 应用程序。
- Web-Socket 模块为 WebSocket-based 提供了支持，而且在 web 应用程序中提供了客户端和服务器端之间通信的两种方式。
- Web-Portlet 模块提供了在 portlet 环境中实现 MVC，并且反映了 Web-Servlet 模块的功能。

其他

还有其他一些重要的模块，像 AOP，Aspects，Instrumentation，Web 和测试模块，它们的细节如下：

- AOP 模块提供了面向方面的编程实现，允许你定义方法拦截器和切入点代码进行干净地解耦，它实现了该分离的功能。
- Aspects 模块提供了与 AspectJ 的集成，这是一个功能强大且成熟的面向切面编程（AOP）框架。
- Instrumentation 模块在一定的应用服务器中提供了类 instrumentation 的支持和类加载器的实现。
- Messaging 模块为 STOMP 提供了支持作为在应用程序中 WebSocket 子协议的使用。它也支持一个注解编程模型，它是为了选路和处理来自 WebSocket 客户端的 STOMP 信息。
- 测试模块支持对具有 JUnit 或 TestNG 框架的 Spring 组件的测试。



环境设置



本教程将指导你如何准备开发环境来使用 Spring 框架开始你的工作。本教程还将教你在安装 Spring 框架之前如何在你的机器上安装 JDK, Tomcat 和 Eclipse。

第 1 步：安装 Java 开发工具包（JDK）

你可以从 Oracle 的 Java 网站 [Java SE Downloads \(http://www.oracle.com/technetwork/java/javase/downloads/index.html\)](http://www.oracle.com/technetwork/java/javase/downloads/index.html) 下载 SDK 的最新版本。你会在下载的文件中找到教你如何安装 JDK 的说明，按照给出的说明安装和配置 JDK 的设置。最后，设置 PATH 和 JAVA_HOME 环境变量，引入包含 java 和 javac 的目录，通常分别为 java _ install _ dir/bin 和 java _ install _ dir。

如果你运行的是 Windows，并在 C:\jdk1.6.0_15 上安装了 JDK，你就可以把下面这行写入 C:\autoexec.bat 文件中。

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

或者，在 Windows XP/7/8 中，你也可以右键单击“我的电脑”，选择“属性”，然后是“高级”，然后是“环境变量”。接下来，你将更新 PATH 值，并且按下 OK 按钮。

在 Unix(Solaris、Linux 等等)上，如果在 /usr/local/jdk1.6.0_15 上安装 SDK，并且使用 C shell 命令，你将把下面的内容添加到 .cshrc 文件中。

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

或者，如果你使用集成开发环境（IDE），如 Borland JBuilder, Eclipse, IntelliJ IDEA 或者 Sun ONE Studio，编译和运行一个简单的程序，用来确认 IDE 知道你安装了 Java，否则应该根据 IDE 给定的文档做正确的设置。

第 2 步：安装 Apache Commons Logging API

你可以从 <http://commons.apache.org/logging/> 下载 Apache Commons Logging API 的最新版本。一旦你下载完安装包，并且解压二进制的发行版本到一个方便的位置。例如在 windows 上的 C:\commons-logging-1.1.1 中，或在 Linux/Unix 上的 /usr/local/commons-logging-1.1.1 中。该目录将有如下的 jar 文件和其他支持的文件等。

Name	Date modified	Type	Size
site	11/22/2007 12:28 ...	File folder	
commons-logging-1.1.1	11/22/2007 12:28 ...	WinRAR archive	60 KB
commons-logging-1.1.1-javadoc	11/22/2007 12:28 ...	WinRAR archive	139 KB
commons-logging-1.1.1-sources	11/22/2007 12:28 ...	WinRAR archive	74 KB
commons-logging-adapters-1.1.1	11/22/2007 12:28 ...	WinRAR archive	26 KB
commons-logging-api-1.1.1	11/22/2007 12:28 ...	WinRAR archive	52 KB
commons-logging-tests	11/22/2007 12:28 ...	WinRAR archive	109 KB
LICENSE	11/22/2007 12:27 ...	Text Document	12 KB
NOTICE	11/22/2007 12:27 ...	Text Document	1 KB
RELEASE-NOTES	11/22/2007 12:27 ...	Text Document	8 KB

确保你在这个目录上正确的设置 CLASSPATH 变量，否则你将会在运行应用程序时遇到问题。

第 3 步：安装 Eclipse IDE

本教程中的所有例子使用 Eclipse IDE 编写。所以我建议你应该在你的机器上安装 Eclipse 的最新版本。

为了安装 Eclipse IDE，从 <http://www.eclipse.org/downloads/> 上下载最新的 Eclipse 二进制文件。一旦你下载完安装包，并且解压二进制的发行版本到一个方便的位置。例如在 windows 上的 C:\eclipse 中，或在 Linux/Unix 上的 /usr/local/eclipse 中，最后恰当的设置 PATH 变量。

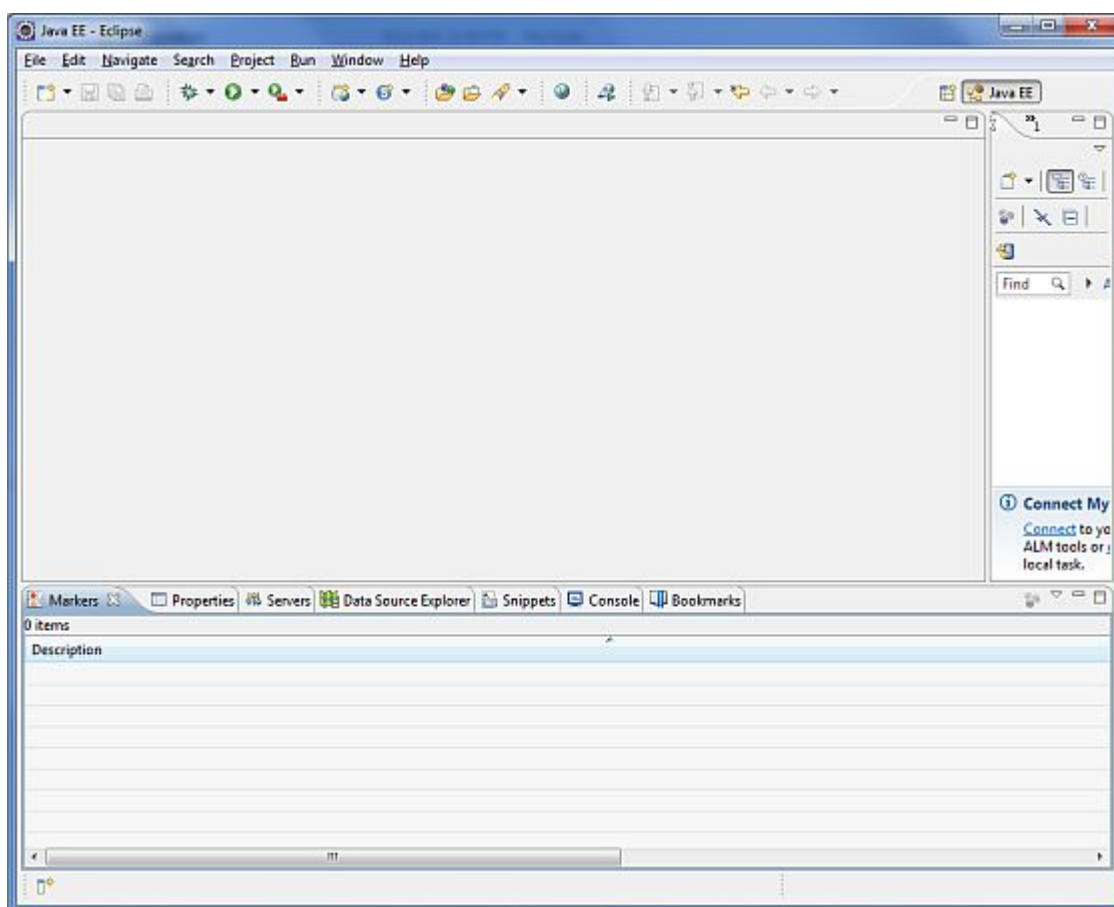
在 Windows 机器上，可以通过执行以下命令启动 Eclipse，或者可以简单地双击 eclipse.exe。

```
%C:\eclipse\eclipse.exe
```

在 Unix (Solaris 和 Linux 等) 上，可以通过执行下面的命令启动 Eclipse：

```
$/usr/local/eclipse/eclipse
```







启动成功后，如果一切正常，它应该显示下面的结果：



第4步：安装 Spring 框架库

现在如果一切正常，你就可以继续设置你的 Spring 框架。下面是在你的机器上下载并安装框架的简单步骤。

- 选择是要在 Windows 还是在 UNIX 上安装 Spring，然后继续进行下一个步骤，在 Windows 上下载 .zip 文件,而在 Unix 上下载 .tar 文件。
- 从 <http://repo.spring.io/release/org/springframework/spring> 下载最新版本的 Spring 框架的二进制文件。
- 在写本教程的时候，我在我的 Windows 机器上下载了 `spring-framework-4.1.6.RELEASE-dist.zip`，当你解压缩下载的文件时，它内置的目录结构为 `E:\spring`，如下所示。

Name	Date modified	Type	Size
 docs	4/22/2015 2:44 PM	File folder	
 libs	4/22/2015 2:45 PM	File folder	
 schema	4/22/2015 2:45 PM	File folder	
 license	4/22/2015 2:42 PM	Text Document	15 KB
 notice	4/22/2015 2:42 PM	Text Document	1 KB
 readme	4/22/2015 2:42 PM	Text Document	1 KB

你会在目录 `E:\spring\libs` 中发现所有的 Spring 库。确保你在这个目录上正确的设置 `CLASSPATH` 变量，否则你将会在运行应用程序时遇到问题。如果使用的是 Eclipse，就不需要设置 `CLASSPATH`，因为所有的设置将通过 Eclipse 完成。

一旦你完成了最后一步后，你就可以继续你的第一个 Spring 例子，你将会在下一章中看到。



4

Hello World 实例

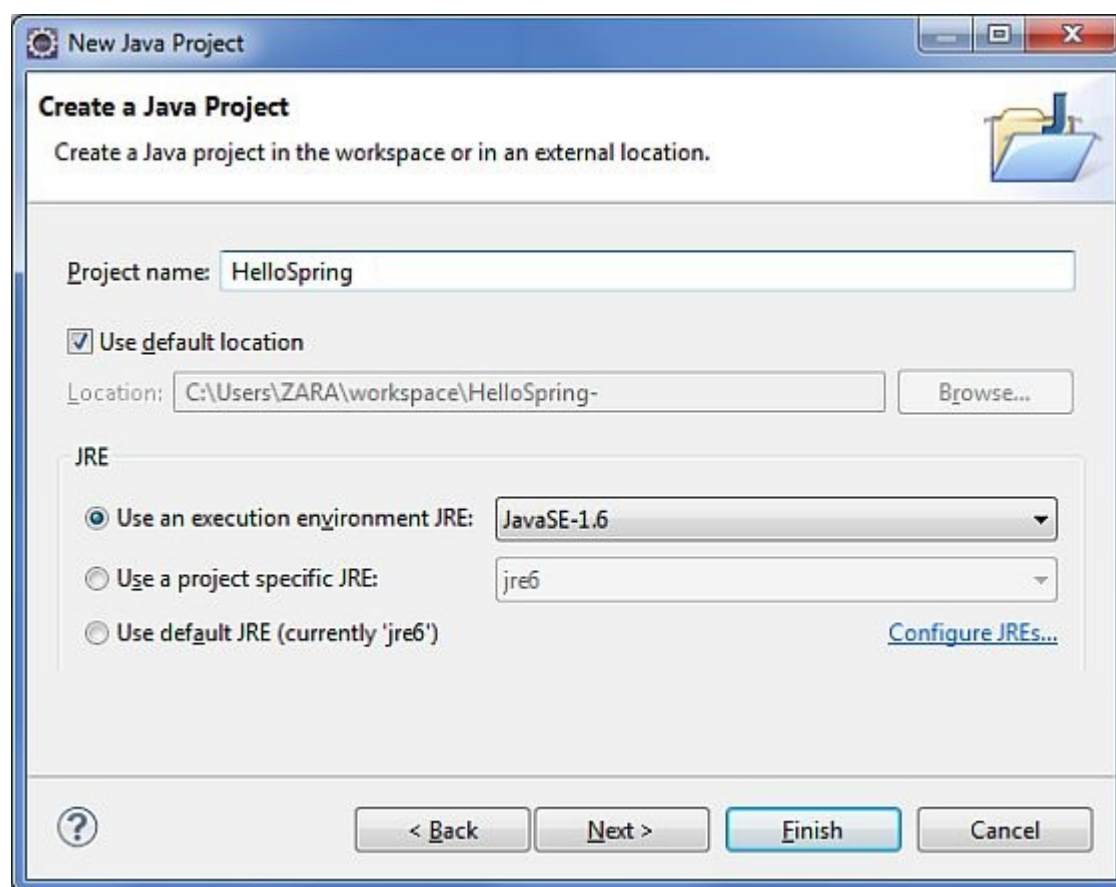


让我们使用 Spring 框架开始实际的编程。在你开始使用 Spring 框架编写第一个例子之前，你必须确保已经正确地设置了 Spring 环境，正如在 [Spring——环境设置 \(\)](#) 教程中如所说的。假设你有了解一些有关 Eclipse IDE 工作的知识。

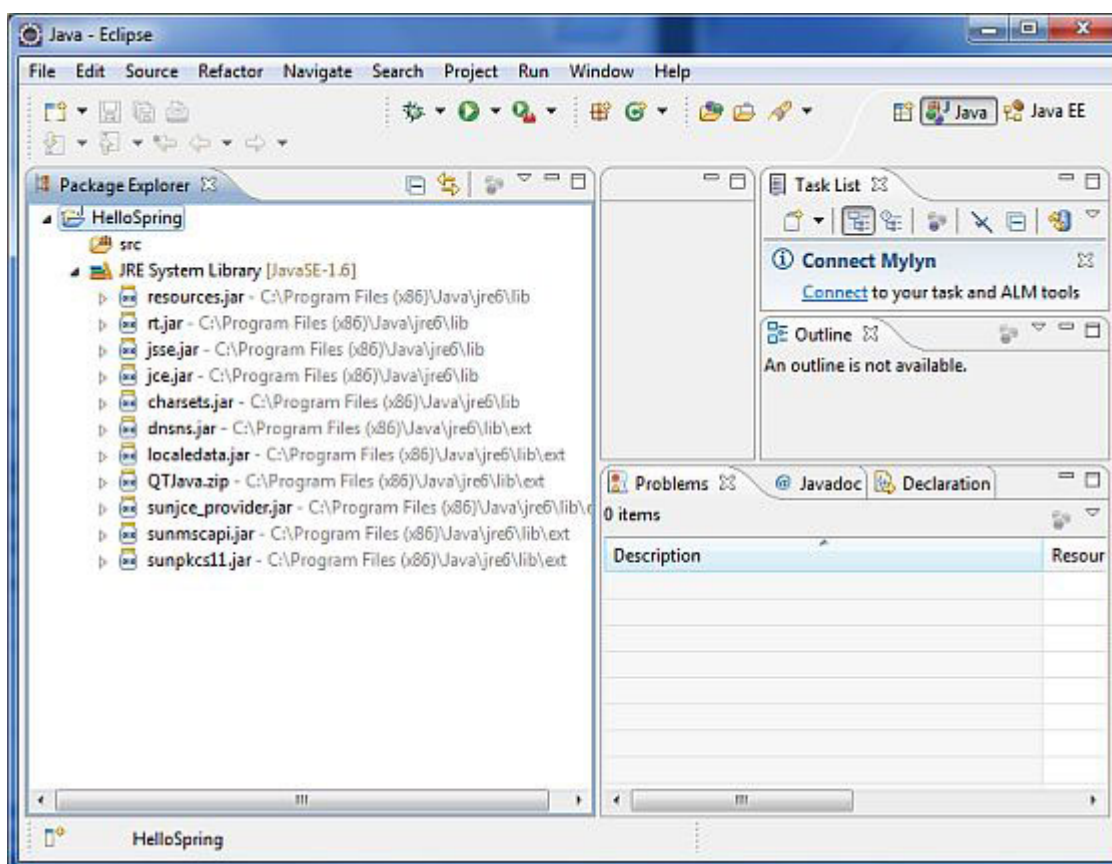
因此，让我们继续编写一个简单的 Spring 应用程序，它将根据在 Spring Beans 配置文件中配置的信息输出“Hello World!”或其他信息。

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。按照选项 **File -> New -> Project**，最后从向导列表中选择 Java Project 向导。现在，使用向导窗口将你的项目命名为 HelloSpring，如下所示：

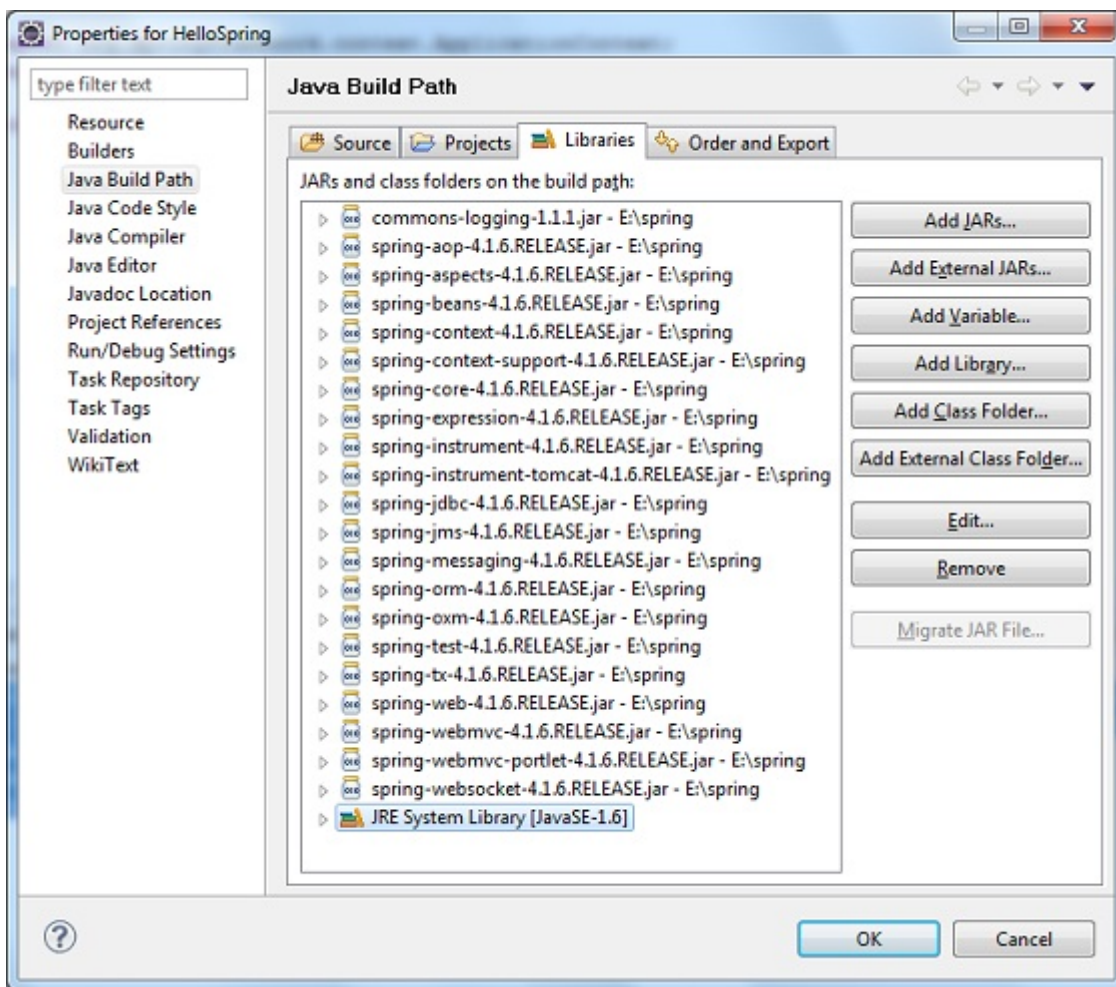


一旦你的项目创建成功后，将在 Project Explorer 看到下面的内容：



第 2 步：添加必需的库

第二步让我们添加 Spring 框架和通用的日志 API 库到我们的项目中。为了做到这个，在你的项目名称 HelloSpring 上单击右键，然后在快捷菜单上按照下面可用的选项：Build Path -> Configure Build Path 显示 Java 构建路径窗口，如下所示：



现在，在 Libraries 标签中使用可用的 Add External JARs 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：

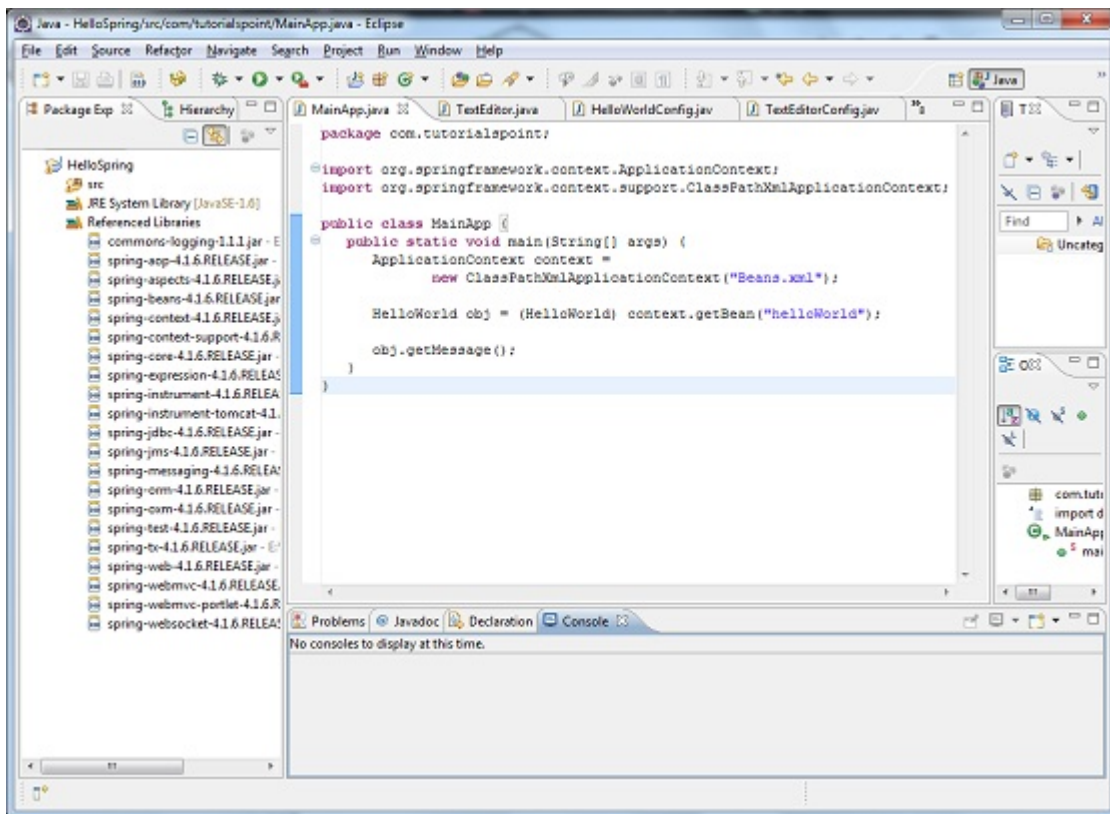
- commons-logging-1.1.1
- spring-aop-4.1.6.RELEASE
- spring-aspects-4.1.6.RELEASE
- spring-beans-4.1.6.RELEASE
- spring-context-4.1.6.RELEASE
- spring-context-support-4.1.6.RELEASE
- spring-core-4.1.6.RELEASE
- spring-expression-4.1.6.RELEASE
- spring-instrument-4.1.6.RELEASE

- spring-instrument-tomcat-4.1.6.RELEASE
- spring-jdbc-4.1.6.RELEASE
- spring-jms-4.1.6.RELEASE
- spring-messaging-4.1.6.RELEASE
- spring-orm-4.1.6.RELEASE
- spring-oxm-4.1.6.RELEASE
- spring-test-4.1.6.RELEASE
- spring-tx-4.1.6.RELEASE
- spring-web-4.1.6.RELEASE
- spring-webmvc-4.1.6.RELEASE
- spring-webmvc-portlet-4.1.6.RELEASE
- spring-websocket-4.1.6.RELEASE

第 3 步：创建源文件

现在让我们在 `HelloSpring` 项目下创建实际的源文件。首先，我们需要创建一个名为 `com.tutorialspoint` 的包。为了做到这个，在 package explore 区域中的 `src` 上点击右键，并按照选项：New -> Package。

接下来，我们在包 `com.tutorialspoint` 下创建 `HelloWorld.java` 和 `MainApp.java` 文件。



这里是 HelloWorld.java 文件的内容：

```
package com.tutorialspoint;

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.tutorialspoint;
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();
    }
}
```



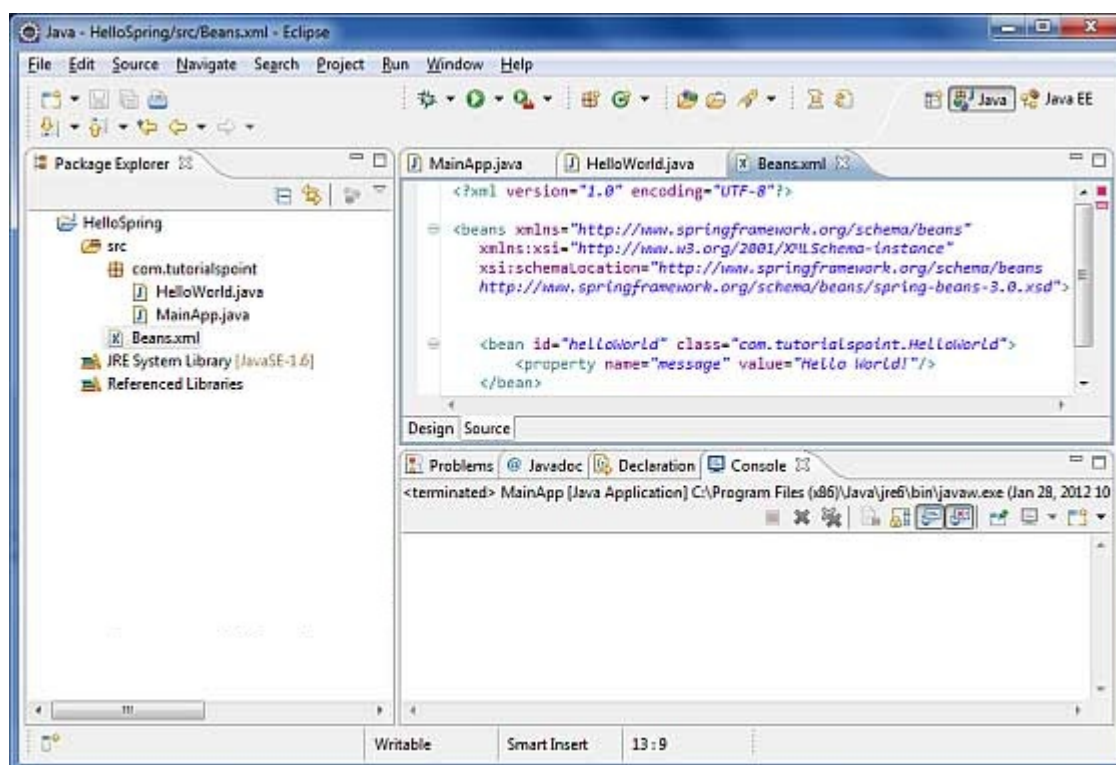
```
}
}
```

关于主要程序有以下两个要点需要注意：

- 第一步是我们使用框架 API `ClassPathXmlApplicationContext()` 来创建应用程序的上下文。这个 API 加载 beans 的配置文件并最终基于所提供的 API，它处理创建并初始化所有的对象，即在配置文件中提到的 beans。
- 第二步是使用已创建的上下文的 `**getBean()` 方法来获得所需的 bean。这个方法使用 bean 的 ID 返回一个最终可以转换为实际对象的通用对象。一旦有了对象，你就可以使用这个对象调用任何类的方法。

第 4 步：创建 bean 的配置文件

你需要创建一个 Bean 的配置文件，该文件是一个 XML 文件，并且作为粘合 bean 的粘合剂即类。这个文件需要在 src 目录下创建，如下图所示：



通常开发人员保存该文件的名称为 Beans.xml 文件，但是你可以单独选择你喜欢的任何名称。你必须确保这个文件在 CLASSPATH 中是可用的，并在主应用程序中使用相同的名称，而在 MainApp.java 文件中创建应用程序的上下文，如图所示。

Beans.xml 用于给不同的 bean 分配唯一的 ID，并且控制不同值的对象的创建，而不会影响 Spring 的任何源文件。例如，使用下面的文件，你可以为 “message” 变量传递任何值，因此你就可以输出信息的不同值，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.tutorialspoint.HelloWorld">
    <property name="message" value="Hello World!"/>
  </bean>

</beans>
```

当 Spring 应用程序被加载到内存中时，框架利用了上面的配置文件来创建所有已经定义的 beans，并且按照标签的定义为它们分配一个唯一的 ID。你可以使用 标签来传递在创建对象时使用不同变量的值。

第 5 步：运行程序

一旦你完成了创建源代码和 bean 的配置文件后，准备好下一步编译和运行你的程序。为了做到这个，请保持 MainApp.java 文件标签是有效的，并且在 Eclipse IDE 中使用可用的 Run 选项，或使用 Ctrl + F11 编译并运行你的应用程序 MainApp。如果你的应用程序一切都正常，将在 Eclipse IDE 控制台打印以下信息：

```
Your Message : Hello World!
```

祝贺，你已经成功地创建了你的第一个 Spring 应用程序。通过更改 “message” 属性的值并且保持两个源文件不变，你可以看到上述 Spring 应用程序的灵活性。下一步，我们开始在接下来的几个章节中做一些更有趣的事情。

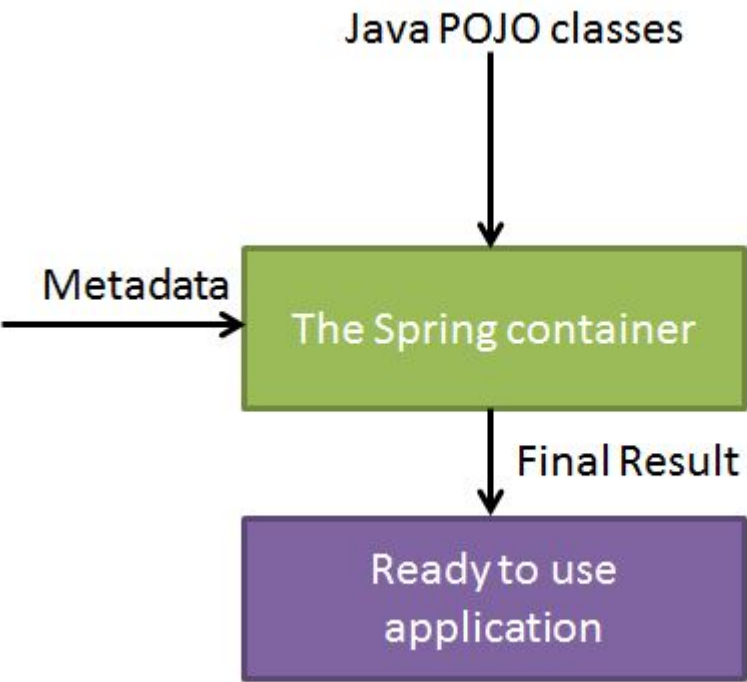


IoC 容器



Spring 容器是 Spring 框架的核心。容器将创建对象，把它们连接在一起，配置它们，并管理他们的整个生命周期从创建到销毁。Spring 容器使用依赖注入（DI）来管理组成一个应用程序的组件。这些对象被称为 Spring Beans，我们将在下一章中进行讨论。

通过阅读配置元数据提供的指令，容器知道对哪些对象进行实例化，配置和组装。配置元数据可以通过 XML，Java 注释或 Java 代码来表示。下图是 Spring 如何工作的高级视图。Spring IoC 容器利用 Java 的 POJO 类和配置元数据来生成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。

序号	容器 & 描述
1	Spring BeanFactory 容器 (ioc-container/spring-bean-factory-container.md) 它是最简单的容器，给 DI 提供了基本的支持，它用 <code>org.springframework.beans.factory.BeanFactory</code> 接口来定义。BeanFactory 或者相关的接口，如 BeanFactory Aware, InitializingBean, DisposableBean, 在 Spring 中仍然存在具有大量的与 Spring 整合的第三方框架的反向兼容性的目的。
2	Spring ApplicationContext 容器 (ioc-container/spring-application-context-container.md) 该容器添加了更多的企业特定的功能，例如从一个属性文件中解析文本信息的能力，发布应用程序事件给感兴趣的事件监听器的能力。该容器是由 <code>org.springframework.context.ApplicationContext</code> 接口定义。

ApplicationContext 容器包括 *BeanFactory* 容器的所有功能，所以通常建议超过 *BeanFactory*。*BeanFactory* 仍然可以用于轻量级的应用程序，如移动设备或基于 applet 的应用程序，其中它的数据量和速度是显著。

Spring 的 BeanFactory 容器

这是一个最简单的容器，它主要的功能是为依赖注入（DI）提供支持，这个容器接口在 `org.springframework.k.beans.factory.BeanFactory` 中被定义。BeanFactory 和相关的接口，比如，BeanFactoryAware、DisposableBean、InitializingBean，仍旧保留在 Spring 中，主要目的是向后兼容已经存在的和那些 Spring 整合在一起的第三方框架。

在 Spring 中，有大量对 BeanFactory 接口的实现。其中，最常被使用的是 XmlBeanFactory 类。这个容器从一个 XML 文件中读取配置元数据，由这些元数据来生成一个被配置化的系统或者应用。

在资源宝贵的移动设备或者基于 applet 的应用当中，BeanFactory 会被优先选择。否则，一般使用的是 ApplicationContext，除非你有更好的理由选择 BeanFactory。

例子：

假设我们已经安装 Eclipse IDE，按照下面的步骤，我们可以创建一个 Spring 应用程序。

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的工程并在 <i>src</i> 文件夹下新建一个名为 <i>com.tutorialspoint</i> 文件夹。
2	点击右键，选择 <i>Add External JARs</i> 选项，导入 Spring 的库文件，正如我们在 <i>Spring Hello World Example</i> 章节中提到的导入方式。
3	在 <i>com.tutorialspoint</i> 文件夹下创建 <i>HelloWorld.java</i> 和 <i>MainApp.java</i> 两个类文件。
4	在 <i>src</i> 文件夹下创建 Bean 的配置文件 <i>Beans.xml</i>
5	最后的步骤是创建所有 Java 文件和 Bean 的配置文件的内容，按照如下所示步骤运行应用程序。

下面是文件 HelloWorld.java 的内容：

```
package com.tutorialspoint;
public class HelloWorld {
    private String message;
    public void setMessage(String message){
        this.message = message;
    }
    public void getMessage(){
        System.out.println("Your Message : " + message);
    }
}
```

下面是文件 MainApp.java 的内容：

```

package com.tutorialspoint;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class MainApp {
    public static void main(String[] args) {
        XmlBeanFactory factory = new XmlBeanFactory
            (new ClassPathResource("Beans.xml"));

        HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
        obj.getMessage();
    }
}

```

在主程序当中，我们需要注意以下两点：

- 第一步利用框架提供的 `XmlBeanFactory()` API 去生成工厂 bean 以及利用 `ClassPathResource()` API 去加载在路径 `CLASSPATH` 下可用的 bean 配置文件。`XmlBeanFactory()` API 负责创建并初始化所有的对象，即在配置文件中提到的 bean。
- 第二步利用第一步生成的 bean 工厂对象的 `getBean()` 方法得到所需要的 bean。这个方法通过配置文件中的 bean ID 来返回一个真正的对象，该对象最后可以用于实际的对象。一旦得到这个对象，就可以利用这个对象来调用任何方法。

下面是配置文件 `Beans.xml` 中的内容：

```

<?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.tutorialspoint.HelloWorld">
        <property name="message" value="Hello World!"/>
    </bean>

</beans>

```

如果你已经完成上面的内容，接下来，让我们运行这个应用程序。如果程序没有错误，你将从控制台看到以下信息：

```
Your Message : Hello World!
```

Spring ApplicationContext 容器

Application Context 是 spring 中较高级的容器。和 *BeanFactory* 类似，它可以加载配置文件中定义的 bean，把所有的 bean 集中在一起，当有请求的时候分配 bean。另外，它增加了企业所需要的功能，比如，从属性文件从解析文本信息和将事件传递给所指定的监听器。这个容器在 *org.springframework.context.ApplicationContext interface* 接口中定义。

ApplicationContext 包含 *BeanFactory* 所有的功能，一般情况下，相对于 *BeanFactory*，*ApplicationContext* 会被推荐使用。*BeanFactory* 仍然可以在轻量级应用中使用，比如移动设备或者基于 applet 的应用程序。

最常被使用的 *ApplicationContext* 接口实现：

- *FileSystemXmlApplicationContext*：该容器从 XML 文件中加载已被定义的 bean。在这里，你需要提供给构造器 XML 文件的完整路径
- *ClassPathXmlApplicationContext*：该容器从 XML 文件中加载已被定义的 bean。在这里，你不需要提供 XML 文件的完整路径，只需正确配置 CLASSPATH 环境变量即可，因为，容器会从 CLASSPATH 中搜索 bean 配置文件。
- *WebXmlApplicationContext*：该容器会在一个 web 应用程序的范围内加载在 XML 文件中已被定义的 bean。

我们已经在 *Spring Hello World Example* 章节中看到过 *ClassPathXmlApplicationContext* 容器，并且，在基于 spring 的 web 应用程序这个独立的章节中，我们讨论了很多关于 *XmlWebApplicationContext*。所以，接下来，让我们看一个关于 *FileSystemXmlApplicationContext* 的例子。

例子：

假设我们已经安装 Eclipse IDE，按照下面的步骤，我们可以创建一个 Spring 应用程序。

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的工程，在 <i>src</i> 下新建一个名为 <i>com.tutorialspoint</i> 的文件夹
2	点击右键，选择 <i>Add External JARs</i> 选项，导入 Spring 的库文件，正如我们在 <i>Spring Hello World Example</i> 章节中提到的导入方式。
3	在 <i>com.tutorialspoint</i> 文件夹下创建 <i>HelloWorld.java</i> 和 <i>MainApp.java</i> 两个类文件。
4	文件夹下创建 Bean 的配置文件 <i>Beans.xml</i> 。
5	最后的步骤是编辑所有 JAVA 文件的内容和 Bean 的配置文件,按照以前我们讲的那样去运行应用程序。

下面是文件 HelloWorld.java 的内容：

```
package com.tutorialspoint;
public class HelloWorld {
    private String message;
    public void setMessage(String message){
        this.message = message;
    }
    public void getMessage(){
        System.out.println("Your Message : " + message);
    }
}
```

下面是文件 MainApp.java 的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new FileSystemXmlApplicationContext
            ("C:/Users/ZARA/workspace/HelloSpring/src/Beans.xml");
        HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
        obj.getMessage();
    }
}
```

在主程序当中，我们需要注意以下两点：

- 第一步生成工厂对象。加载完指定路径下 bean 配置文件后，利用框架提供的 `FileSystemXmlApplicationContext` API 去生成工厂 bean。`FileSystemXmlApplicationContext` 负责生成和初始化所有的对象，比如，所有在 XML bean 配置文件中的 bean。
- 第二步利用第一步生成的上下文中的 `getBean()` 方法得到所需要的 bean。这个方法通过配置文件中的 bean ID 来返回一个真正的对象。一旦得到这个对象，就可以利用这个对象来调用任何方法。

下面是配置文件 Beans.xml 中的内容：

```
<?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.tutorialspoint.HelloWorld">
        <property name="message" value="Hello World!"/>
    </bean>

</beans>
```

如果你已经完成上面的内容，接下来，让我们运行这个应用程序。如果程序没有错误，你将从控制台看到以下信息：

```
Your Message : Hello World!
```




Bean 定义



被称作 bean 的对象是构成应用程序的支柱也是由 Spring IoC 容器管理的。bean 是一个被实例化，组装，并通过 Spring IoC 容器所管理的对象。这些 bean 是由用容器提供的配置元数据创建的，例如，已经在先前章节看到的，在 XML 的表单中的定义。

bean 定义包含称为配置元数据的信息，下述容器也需要知道配置元数据：

- 如何创建一个 bean
- bean 的生命周期的详细信息
- bean 的依赖关系

上述所有的配置元数据转换成一组构成每个 bean 定义的下列属性。

属性	描述
<u>class</u>	<u>这个属性是强制性的，并且指定用来创建 bean 的 bean 类。</u>
<u>name</u>	<u>这个属性指定唯一的 bean 标识符。在基于 XML 的配置元数据中，你可以使用 ID 和/或 name 属性来指定 bean 标识符。</u>
<u>scope</u>	<u>这个属性指定由特定的 bean 定义创建的对象的作用域，它将会在 bean 作用域的章节中进行讨论。</u>
constructor-arg	它是用来注入依赖关系的，并会在接下来的章节中进行讨论。
properties	它是用来注入依赖关系的，并会在接下来的章节中进行讨论。
autowiring mode	它是用来注入依赖关系的，并会在接下来的章节中进行讨论。
<u>lazy-initialization mode</u>	<u>延迟初始化的 bean 告诉 IoC 容器在它第一次被请求时，而不是在启动时去创建一个 bean 实例。</u>
<u>initialization 方法</u>	<u>在 bean 的所有必需的属性被容器设置之后，调用回调方法。它将会在 bean 的生命周期章节中进行讨论。</u>
<u>destruction 方法</u>	<u>当包含该 bean 的容器被销毁时，使用回调方法。它将会在 bean 的生命周期章节中进行讨论。</u>

Spring 配置元数据

Spring IoC 容器完全由实际编写的配置元数据的格式解耦。有下面三个重要的方法把配置元数据提供给 Spring 容器：

- 基于 XML 的配置文件。
- 基于注解的配置
- 基于 Java 的配置

你已经看到了如何把基于 XML 的配置元数据提供给容器，但是让我们看看另一个基于 XML 配置文件的例子，这个配置文件中有不同的 bean 定义，包括延迟初始化，初始化方法和销毁方法的：

```
<?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">

  <!-- 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>

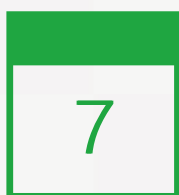
  <!-- A bean definition with destruction method -->
  <bean id="..." class="..." destroy-method="...">
    <!-- collaborators and configuration for this bean go here -->
  </bean>

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

</beans>
```

你可以查看 [Spring Hello World 实例 \(\)](#) 来理解如何定义，配置和创建 Spring Beans。

关于基于注解的配置将在一个单独的章节中进行讨论。刻意把它保留在一个单独的章节，是因为我想让你在开始使用注解和 Spring 依赖注入编程之前，能掌握一些其他重要的 Spring 概念。



Bean 的作用域



当在 Spring 中定义一个时，你必须声明该 bean 的作用域的选项。例如，为了强制 Spring 在每次需要时都产生一个新的 bean 实例，你应该声明 bean 的作用域的属性为 `prototype`。同理，如果你想让 Spring 在每次需要时都返回同一个 bean 实例，你应该声明 bean 的作用域的属性为 `singleton`。

Spring 框架支持以下五个作用域，如果你使用 web-aware `ApplicationContext` 时，其中三个是可用的。

作用域	描述
<code>singleton</code>	该作用域将 bean 的定义的限制在每一个 Spring IoC 容器中的一个单一实例(默认)。
<code>prototype</code>	该作用域将单一 bean 的定义限制在任意数量的对象实例。
<code>request</code>	该作用域将 bean 的定义限制为 HTTP 请求。只在 web-aware Spring <code>ApplicationContext</code> 的上下文中有效。
<code>session</code>	该作用域将 bean 的定义限制为 HTTP 会话。只在 web-aware Spring <code>ApplicationContext</code> 的上下文中有效。
<code>global-session</code>	该作用域将 bean 的定义限制为全局 HTTP 会话。只在 web-aware Spring <code>ApplicationContext</code> 的上下文中有效。

本章将讨论前两个范围，当我们将讨论有关 web-aware `Spring ApplicationContext` 时，其余三个将被讨论。

singleton 作用域:

如果作用域设置为 `singleton`，那么 Spring IoC 容器刚好创建一个由该 bean 定义的对象实例。该单一实例将存储在这种单例 bean 的高速缓存中，以及针对该 bean 的所有后续的请求和引用都返回缓存对象。

默认作用域是始终是 `singleton`，但是当仅仅需要 bean 的一个实例时，你可以在 bean 的配置文件中设置作用域的属性为 `singleton`，如下所示：

```
<!-- A bean definition with singleton scope -->
<bean id="..." class="..." scope="singleton">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```

例子

我们在适当的位置使用 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	创建一个名称为 <code>SpringExample</code> 的项目，并且在创建项目的 <code>src</code> 文件夹中创建一个包 <code>com.tutorialspoint</code> 。
2	使用 <code>Add External JARs</code> 选项，添加所需的 Spring 库，在 <code>Spring Hello World Example</code> 章节解释。
3	在 <code>com.tutorialspoint</code> 包中创建 Java 类 <code>HelloWorld</code> 和 <code>MainApp</code> 。

- | | |
|---|---|
| 4 | 在 <code>src</code> 文件夹中创建 Beans 配置文件 <code>Beans.xml</code> 。 |
| 5 | 最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下。 |

这里是 `HelloWorld.java` 文件的内容：

```
package com.tutorialspoint;
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 `MainApp.java` 文件的内容：

```
package com.tutorialspoint;
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();
    }
}
```

下面是 singleton 作用域必需的配置文件 `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.tutorialspoint.HelloWorld"
        scope="singleton">
    </bean>

</beans>
```

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```
Your Message : I'm object A
Your Message : I'm object A
```

prototype 作用域

如果作用域设置为 prototype，那么每次特定的 bean 发出请求时 Spring IoC 容器就创建对象的新的 Bean 实例。一般说来，满状态的 bean 使用 prototype 作用域和没有状态的 bean 使用 singleton 作用域。

为了定义 prototype 作用域，你可以在 bean 的配置文件中设置作用域的属性为 prototype，如下所示：

```
<!-- A bean definition with singleton scope -->
<bean id="..." class="..." scope="prototype">
    <!-- collaborators and configuration for this bean go here -->
</bean>
```

例子

我们在适当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>HelloWorld</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹中创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 *HelloWorld.java* 文件的内容：

```
package com.tutorialspoint;

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 MainApp.java 文件的内容：

```

package com.tutorialspoint;
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();
    }
}

```

下面是 **prototype** 作用域必需的配置文件 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.tutorialspoint.HelloWorld"
        scope="prototype">
    </bean>

</beans>

```

一旦你创建源代码和 Bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```

Your Message : I'm object A
Your Message : null

```




T



8

Bean 的生命周期



理解 Spring bean 的生命周期很容易。当一个 bean 被实例化时，它可能需要执行一些初始化使它转换成可用状态。同样，当 bean 不再需要，并且从容器中移除时，可能需要做一些清除工作。

尽管还有一些在 Bean 实例化和销毁之间发生的活动，但是本章将只讨论两个重要的生命周期回调方法，它们在 bean 的初始化和销毁的时候是必需的。

为了定义安装和拆卸一个 bean，我们只要声明带有 `init-method` 和/或 `destroy-method` 参数的。 `init-method` 属性指定一个方法，在实例化 bean 时，立即调用该方法。同样，`destroy-method` 指定一个方法，只有从容器中移除 bean 之后，才能调用该方法。

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，你可以简单地实现上述接口和初始化工作可以在 `afterPropertiesSet()` 方法中执行，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，你可以使用 `init-method` 属性来指定带有 void 无参数方法的名称。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并且结束工作可以在 `destroy()` 方法中执行，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，你可以使用 `destroy-method` 属性来指定带有 `void` 无参数方法的名称。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
    }
}
```

如果你在非 web 应用程序环境中使用 Spring 的 IoC 容器；例如在丰富的客户端桌面环境中；那么在 JVM 中你要注册关闭 hook。这样做可以确保正常关闭，为了让所有的资源都被释放，可以在单个 beans 上调用 `destroy` 方法。

建议你不要使用 `InitializingBean` 或者 `DisposableBean` 的回调方法，因为 XML 配置在命名方法上提供了极大的灵活性。

例子

我们在适当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <code>src</code> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>HelloWorld</i> 和 <i>MainApp</i> 。
4	在 <code>src</code> 文件夹中创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 `HelloWorld.java` 的文件的内容：

```

package com.tutorialspoint;

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.");
    }
}

```

下面是 MainApp.java 文件的内容。在这里，你需要注册一个在 AbstractApplicationContext 类中声明的关闭 hook 的 registerShutdownHook() 方法。它将确保正常关闭，并且调用相关的 destroy 方法。

```

package com.tutorialspoint;
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 方法必需的配置文件 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.tutorialspoint.HelloWorld"
        init-method="init" destroy-method="destroy">

```

```

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

</beans>

```

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```

Bean is going through init.
Your Message : Hello World!
Bean will destroy now.

```

默认的初始化和销毁方法

如果你有太多具有相同名称的初始化或者销毁方法的 Bean，那么你不需要在每一个 bean 上声明初始化方法和销毁方法。框架使用元素中的 `default-init-method` 和 `default-destroy-method` 属性提供了灵活地配置这种情况，如下所示：

```

<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>

```



9



Spring——Bean 后置处理器



BeanPostProcessor 接口定义回调方法，你可以实现该方法来提供自己的实例化逻辑，依赖解析逻辑等。你也可以在 Spring 容器通过插入一个或多个 **BeanPostProcessor** 的实现来完成实例化，配置和初始化一个 bean 之后实现一些自定义逻辑回调方法。

你可以配置多个 **BeanPostProcessor** 接口，通过设置 **BeanPostProcessor** 实现的 **Ordered** 接口提供的 **order** 属性来控制这些 **BeanPostProcessor** 接口的执行顺序。

BeanPostProcessor 可以对 bean（或对象）实例进行操作，这意味着 Spring IoC 容器实例化一个 bean 实例，然后 **BeanPostProcessor** 接口进行它们的工作。

ApplicationContext 会自动检测由 **BeanPostProcessor** 接口的实现定义的 bean，注册这些 bean 为后置处理器，然后通过容器中创建 bean，在适当的时候调用它。

例子：

下面的例子显示如何在 **ApplicationContext** 的上下文中编写，注册和使用 **BeanPostProcessor**。

我们在适当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 src 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>HelloWorld</i> 、 <i>InitHelloWorld</i> 和 <i>MainApp</i> 。
4	在 src 文件夹中创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 ****HelloWorld.java**** 文件的内容：

```
package com.tutorialspoint;
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` 的非常简单的例子，它在任何 bean 的初始化的之前和之后输入该 bean 的名称。你可以在初始化 bean 的之前和之后实现更复杂的逻辑，因为你有两个访问内置 bean 对象的后置处理程序的方法。

这里是 `InitHelloWorld.java` 文件的内容：

```
package com.tutorialspoint;
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
    }
}
```

下面是 `MainApp.java` 文件的内容。在这里，你需要注册一个在 `AbstractApplicationContext` 类中声明的关闭 hook 的 `registerShutdownHook()` 方法。它将确保正常关闭，并且调用相关的 `destroy` 方法。

```
package com.tutorialspoint;
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.tutorialspoint.HelloWorld"
    init-method="init" destroy-method="destroy">
    <property name="message" value="Hello World!"/>
</bean>

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

</beans>
```

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```
BeforeInitialization : helloWorld
Bean is going through init.
AfterInitialization : helloWorld
Your Message : Hello World!
Bean will destroy now.
```



10

Bean 定义继承



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，容器的具体信息例如初始化方法，静态工厂方法名，等等。

子 bean 的定义继承父定义的配置数据。子定义可以根据需要重写一些值，或者添加其他值。

Spring Bean 定义的继承与 Java 类的继承无关，但是继承的概念是一样的。你可以定义一个父 bean 的定义作为模板和其他子 bean 就可以从父 bean 中继承所需的配置。

当你使用基于 XML 的配置元数据时，通过使用父属性，指定父 bean 作为该属性的值来表明子 bean 的定义。

例子

我们在适当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>HelloWorld</i> 、 <i>HelloIndia</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹中创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

下面是配置文件 *Beans.xml*，在该配置文件中我们定义有两个属性 *message1* 和 *message2* 的 “helloWorld” bean。然后，使用 *parent* 属性把 “helloIndia” bean 定义为 “helloWorld” bean 的孩子。这个子 bean 继承 *message2* 的属性，重写 *message1* 的属性，并且引入一个属性 *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.tutorialspoint.HelloWorld">
    <property name="message1" value="Hello World!"/>
    <property name="message2" value="Hello Second World!"/>
  </bean>

  <bean id="helloIndia" class="com.tutorialspoint.HelloIndia" parent="helloWorld">
    <property name="message1" value="Hello India!"/>
    <property name="message3" value="Namaste India!"/>
  </bean>
</beans>
```

```
</bean>

</beans>
```

这里是 HelloWorld.java 文件的内容：

```
package com.tutorialspoint;
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.tutorialspoint;

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);
}
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;

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();
    }
}

```

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```

World Message1 : Hello World!
World Message2 : Hello Second World!
India Message1 : Hello India!
India Message2 : Hello Second World!
India Message3 : Namaste India!

```

在这里你可以观察到，我们创建 “helloIndia” bean 的同时并没有传递 message2，但是由于 Bean 定义的继承，所以它传递了 message2。

Bean 定义模板

你可以创建一个 Bean 定义模板，不需要花太多功夫它就可以被其他子 bean 定义使用。在定义一个 Bean 定义模板时，你不应该指定类的属性，而应该指定带 true 值的抽象属性，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="beanTemplate" 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.tutorialspoint.HelloIndia" parent="beanTemplate">
    <property name="message1" value="Hello India!"/>
    <property name="message3" value="Namaste India!"/>
  </bean>

</beans>
```

父 bean 自身不能被实例化，因为它是不完整的，而且它也被明确地标记为抽象的。当一个定义是抽象的，它仅作为一个纯粹的模板 bean 定义来使用的，充当子定义的父定义使用。



依赖注入



每个基于应用程序的 java 都有几个对象，这些对象一起工作来呈现出终端用户所看到的工作的应用程序。当编写一个复杂的 Java 应用程序时，应用程序类应该尽可能独立于其他 Java 类来增加这些类重用的可能性，并且在做单元测试时，测试独立于其他类的独立性。依赖注入（或有时称为布线）有助于把这些类粘合在一起，同时保持他们独立。

假设你有一个包含文本编辑器组件的应用程序，并且你想要提供拼写检查。标准代码看起来是这样的：

```
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor() {
        spellChecker = new SpellChecker();
    }
}
```

在这里我们所做的就是创建一个 TextEditor 和 SpellChecker 之间的依赖关系。在控制反转的场景中，我们反而会做这样的事情：

```
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor() {
        spellChecker = new SpellChecker();
    }
}
```

在这里，TextEditor 不应该担心 SpellChecker 的实现。SpellChecker 将会独立实现，并且在 TextEditor 实例化的时候将提供给 TextEditor，整个过程是由 Spring 框架的控制。

在这里，我们已经从 TextEditor 中删除了全面控制，并且把它保存到其他地方（即 XML 配置文件），且依赖关系（即 SpellChecker 类）通过类构造函数被注入到 TextEditor 类中。因此，控制流通过依赖注入（DI）已经“反转”，因为你已经有效地委托依赖关系到一些外部系统。

依赖注入的第二种方法是通过 TextEditor 类的 Setter 方法，我们将创建 SpellChecker 实例，该实例将被用于调用 setter 方法来初始化 TextEditor 的属性。

因此，DI 主要有两种变体和下面的两个子章将结合实例涵盖它们：

序号 依赖注入类型 & 描述

- 1 [Constructor-based dependency injection \(dependency-injection/spring-constructor-based-dependency-injection.md\)](#)

当容器调用带有多个参数的构造函数类时，实现基于构造函数的 DI，每个代表在其他类中的一个依赖关系。

2 [Setter-based dependency injection \(dependency-injection/spring-setter-based-dependency-injection.md\)](#)

基于 setter 方法的 DI 是通过在调用无参数的构造函数或无参数的静态工厂方法实例化 bean 之后容器调用 beans 的 setter 方法来实现的。

你可以混合这两种方法，基于构造函数和基于 setter 方法的 DI，然而使用有强制性依存关系的构造函数和有可选依赖关系的 setter 是一个好的做法。

代码是 DI 原理的清洗机，当对象与它们的依赖关系被提供时，解耦效果更明显。对象不查找它的依赖关系，也不知道依赖关系的位置或类，而这一切都由 Spring 框架控制的。

Spring 基于构造函数的依赖注入

当容器调用带有一组参数的类构造函数时，基于构造函数的 DI 就完成了，其中每个参数代表一个对其他类的依赖。

示例：

下面的例子显示了一个类 `TextEditor`，只能用构造函数注入来实现依赖注入。

让我们用 Eclipse IDE 适当地工作，并按照以下步骤创建一个 Spring 应用程序。

步骤	描述
1	创建一个名为 <code>SpringExample</code> 的项目，并在创建的项目中的 <code>src</code> 文件夹下创建包 <code>com.tutorialspoint</code> 。
2	使用 <code>Add External JARs</code> 选项添加必需的 Spring 库，解释见 <code>Spring Hello World Example chapter</code> 。
3	在 <code>com.tutorialspoint</code> 包下创建 Java 类 <code>TextEditor</code> 、 <code>SpellChecker</code> 和 <code>MainApp</code> 。
4	在 <code>src</code> 文件夹下创建 Beans 的配置文件 <code>Beans.xml</code> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容并按照如下所示的方法运行应用程序。

这是 `TextEditor.java` 文件的内容：

```
package com.tutorialspoint;
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor(SpellChecker spellChecker) {
        System.out.println("Inside TextEditor constructor.");
        this.spellChecker = spellChecker;
    }
    public void spellCheck() {
        spellChecker.checkSpelling();
    }
}
```

下面是另一个依赖类文件 `SpellChecker.java` 的内容：

```
package com.tutorialspoint;
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.tutorialspoint;
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();
    }
}
```

下面是配置文件 Beans.xml 的内容，它有基于构造函数注入的配置：

```
<?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 -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <constructor-arg ref="spellChecker"/>
    </bean>

    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
    </bean>

</beans>
```

当你完成了创建源和 bean 配置文件后，让我们开始运行应用程序。如果你的应用程序运行顺利的话，那么将会输出下述所示消息：

```
Inside SpellChecker constructor.
Inside TextEditor constructor.
Inside checkSpelling.
```

构造函数参数解析：

如果存在不止一个参数时，当把参数传递给构造函数时，可能会存在歧义。要解决这个问题，那么构造函数的参数在 bean 定义中的顺序就是把这些参数提供给适当的构造函数的顺序就可以了。考虑下面的类：

```
package x.y;
public class Foo {
    public Foo(Bar bar, Baz baz) {
        // ...
    }
}
```

下述配置文件工作顺利：

```
<beans>
    <bean id="foo" class="x.y.Foo">
        <constructor-arg ref="bar"/>
        <constructor-arg ref="baz"/>
    </bean>

    <bean id="bar" class="x.y.Bar"/>
    <bean id="baz" class="x.y.Baz"/>
</beans>
```

让我们再检查一下我们传递给构造函数不同类型的位置。考虑下面的类：

```
package x.y;
public class Foo {
    public Foo(int year, String name) {
        // ...
    }
}
```

如果你使用 `type` 属性显式的指定了构造函数参数的类型，容器也可以使用与简单类型匹配的类型。例如：

```
<beans>

    <bean id="exampleBean" class="examples.ExampleBean">
        <constructor-arg type="int" value="2001"/>
        <constructor-arg type="java.lang.String" value="Zara"/>
    </bean>

</beans>
```

最后并且也是最好的传递构造函数参数的方式，使用 `index` 属性来显式的指定构造函数参数的索引。下面是基于索引为 0 的例子，如下所示：

```
<beans>

    <bean id="exampleBean" class="examples.ExampleBean">
        <constructor-arg index="0" value="2001"/>
        <constructor-arg index="1" value="Zara"/>
    </bean>

</beans>
```

最后，如果你想要向一个对象传递一个引用，你需要使用 标签的 `ref` 属性，如果你想要直接传递值，那么你应该使用如上所示的 `value` 属性。

Spring 基于设值函数的依赖注入

当容器调用一个无参的构造函数或一个无参的静态 `factory` 方法来初始化你的 `bean` 后，通过容器在你的 `bean` 上调用设值函数，基于设值函数的 DI 就完成了。

示例：

下述例子显示了一个类 `TextEditor`，它只能使用纯粹的基于设值函数的注入来实现依赖注入。

让我们用 Eclipse IDE 适当地工作，并按照以下步骤创建一个 Spring 应用程序。

步骤	描述
1	创建一个名为 <code>SpringExample</code> 的项目，并在创建的项目中的 <code>src</code> 文件夹下创建包 <code>com.tutorialspoint</code> 。
2	使用 <code>Add External JARs</code> 选项添加必需的 Spring 库，解释见 <code>Spring Hello World Example chapter</code> 。
3	在 <code>com.tutorialspoint</code> 包下创建 Java 类 <code>TextEditor</code> ， <code>SpellChecker</code> 和 <code>MainApp</code> 。
4	在 <code>src</code> 文件夹下创建 Beans 的配置文件 <code>Beans.xml</code> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容并按照如下所示的方法运行应用程序。

下面是 `TextEditor.java` 文件的内容：

```
package com.tutorialspoint;

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();
    }
}
```

在这里，你需要检查设值函数方法的名称转换。要设置一个变量 `spellChecker`，我们使用 `setSpellChecker()` 方法，该方法与 Java POJO 类非常相似。让我们创建另一个依赖类文件 `SpellChecker.java` 的内容：

```
package com.tutorialspoint;
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.tutorialspoint;
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();
    }
}
```

下面是配置文件 Beans.xml 的内容，该文件有基于设值函数注入的配置：

```
<?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 -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <property name="spellChecker" ref="spellChecker"/>
    </bean>

    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
    </bean>

</beans>
```

你应该注意定义在基于构造函数注入和基于设值函数注入中的 Beans.xml 文件的区别。唯一的区别就是在基于构造函数注入中，我们使用的是 `<constructor-arg>` 元素，而在基于设值函数的注入中，我们使用的是 `<property>` 元素。

第二个你需要注意的点是，如果你要把一个引用传递给一个对象，那么你需要使用 `ref` 属性，而如果你要直接传递一个值，那么你应该使用 `value` 属性。

当你完成了创建源和 bean 配置文件后，让我们开始运行应用程序。如果你的应用程序运行顺利的话，那么将会输出下述所示消息：

```
Inside SpellChecker constructor.
Inside setSpellChecker.
Inside checkSpelling.
```

使用 p-namespace 实现 XML 配置：

如果你有许多的设值函数方法，那么在 XML 配置文件中使用 **p-namespace** 是非常方便的。让我们查看一下区别：

以带有 标签的标准 XML 配置文件为例：

```
<?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">

  <bean id="john-classic" class="com.example.Person">
    <property name="name" value="John Doe"/>
    <property name="spouse" ref="jane"/>
  </bean>

  <bean name="jane" class="com.example.Person">
    <property name="name" value="John Doe"/>
  </bean>

</beans>
```

上述 XML 配置文件可以使用 **p-namespace** 以一种更简洁的方式重写，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"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="john-classic" class="com.example.Person"
    p:name="John Doe"
    p:spouse-ref="jane"/>
  </bean>

  <bean name="jane" class="com.example.Person"
    p:name="John Doe"/>
  </bean>

</beans>
```

在这里，你不应该区别指定原始值和带有 p-namespace 的对象引用。`-ref` 部分表明这不是一个直接的值，而是对另一个 bean 的引用。



12

注入内部 Beans



正如你所知道的 Java 内部类是在其他类的范围内被定义的，同理，**inner beans** 是在其他 bean 的范围内定义的 bean。因此在 `<bean>` 元素内 元素被称为内部bean，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>
```

例子

我们在适当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。option as explained in the chapter.
3	在 <i>com.tutorialspoint</i> 包中创建Java类 <i>TextEditor</i> 、 <i>SpellChecker</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹中创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有Java文件和Bean配置文件的内容，并运行应用程序，解释如下所示。

这里是 *TextEditor.java* 文件的内容：

```
package com.tutorialspoint;

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();
}
}

```

下面是另一个依赖的类文件 `SpellChecker.java` 内容：

```

package com.tutorialspoint;
public class SpellChecker {
    public SpellChecker(){
        System.out.println("Inside SpellChecker constructor." );
    }
    public void checkSpelling(){
        System.out.println("Inside checkSpelling." );
    }
}

```

下面是 `MainApp.java` 文件的内容：

```

package com.tutorialspoint;
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();
    }
}

```

下面是使用内部 bean 为基于 setter 注入进行配置的配置文件 `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">

    <!-- Definition for textEditor bean using inner bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <property name="spellChecker">
            <bean id="spellChecker" class="com.tutorialspoint.SpellChecker"/>
        </property>
    </bean>

```

```
</property>  
</bean>  
  
</beans>
```

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```
Inside SpellChecker constructor.  
Inside setSpellChecker.  
Inside checkSpelling.
```



注入集合



你已经看到了如何使用 `value` 属性来配置基本数据类型和在你的 bean 配置文件中使用 标签的 `ref` 属性来配置对象引用。这两种情况下处理奇异值传递给一个 bean。

现在如果你想传递多个值，如 Java Collection 类型 List、Set、Map 和 Properties，应该怎么做呢。为了处理这种情况，Spring 提供了四种类型的集合的配置元素，如下所示：

元素	描述
<list>	它有助于连线，如注入一系列值，允许重复。
<set>	它有助于连线一组值，但不能重复。
<map>	它可以用来注入名称-值对的集合，其中名称和值可以是任何类型。
<props>	它可以用来注入名称-值对的集合，其中名称和值都是字符串类型。

你可以使用 或 来连接任何 `java.util.Collection` 的实现或数组。

你会遇到两种情况（a）传递集合中直接的值（b）传递一个 bean 的引用作为集合的元素。

例子

我们在适当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <code>src</code> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。option as explained in the chapter.
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>TextEditor</i> 、 <i>SpellChecker</i> 和 <i>MainApp</i> 。
4	在 <code>src</code> 文件夹中创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 `JavaCollection.java` 文件的内容：

```
package com.tutorialspoint;
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;
}
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;
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();
}
}

```

下面是配置所有类型的集合的配置文件 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">

    <!-- Definition for javaCollection -->
    <bean id="javaCollection" class="com.tutorialspoint.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>
    </bean>

```



```

<!-- 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>

```

一旦你创建源代码和 bean 配置文件完成后，我们就可以运行该应用程序。你应该注意这里不需要配置文件。如果你的应用程序一切都正常，将输出以下信息：

```

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

下面的 Bean 定义将帮助你理解如何注入 bean 的引用作为集合的元素。甚至你可以将引用和值混合在一起，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 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>
  </bean>
</beans>

```

```

        </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>

```

为了使用上面的 bean 定义，你需要定义 setter 方法，它们应该也能够是用这种方式来处理引用。

注入 null 和空字符串的值

如果你需要传递一个空字符串作为值，那么你可以传递它，如下所示：

```

<bean id="..." class="exampleBean">
    <property name="email" value=""/>
</bean>

```

前面的例子相当于 Java 代码：exampleBean.setEmail("")。

如果你需要传递一个 NULL 值，那么你可以传递它，如下所示：

```

<bean id="..." class="exampleBean">
    <property name="email"><null/></property>
</bean>

```

前面的例子相当于 Java 代码：exampleBean.setEmail(null)。



14

Beans 自动装配



你已经学会如何使用 `<bean>` 元素来声明 bean 和通过使用 XML 配置文件中的 `<import>` 和 `<context:component-scan>` 元素来注入。

Spring 容器可以在不使用 `<bean>` 元素的情况下自动装配相互协作的 bean 之间的关系，这有助于减少编写一个大的基于 Spring 的应用程序的 XML 配置的数量。

自动装配模式

下列自动装配模式，它们可用于指示 Spring 容器为来使用自动装配进行依赖注入。你可以使用 `<bean>` 元素的 `autowire` 属性为一个 bean 定义指定自动装配模式。

模式	描述
no	这是默认的设置，它意味着没有自动装配，你应该使用显式的bean引用来连线。你不用为了连线做特殊的事。在依赖注入章节你已经看到这个了。
byName (beans-auto-wiring/spring-autowiring-byname.md)	由属性名自动装配。Spring 容器看到在 XML 配置文件中 bean 的 <i>自动装配</i> 的属性设置为 <i>byName</i> 。然后尝试匹配，并且将它的属性与在配置文件中被定义为相同名称的 beans 的属性进行连接。
byType (beans-auto-wiring/spring-autowiring-byType.md)	由属性数据类型自动装配。Spring 容器看到在 XML 配置文件中 bean 的 <i>自动装配</i> 的属性设置为 <i>byType</i> 。然后如果它的类型匹配配置文件中的一个确切的 bean 名称，它将尝试匹配和连接属性的类型。如果存在不止一个这样的 bean，则一个致命的异常将会被抛出。
constructor (beans-auto-wiring/spring-autowiring-by-Constructor.md)	类似于 <i>byType</i> ，但该类型适用于构造函数参数类型。如果在容器中没有一个构造函数参数类型的 bean，则一个致命错误将会发生。
autodetect	Spring首先尝试通过 <i>constructor</i> 使用自动装配来连接，如果它不执行，Spring 尝试通过 <i>byType</i> 来自动装配。

可以使用 *byType* 或者 *constructor* 自动装配模式来连接数组和其他类型的集合。

自动装配的局限性

当自动装配始终在同一个项目中使用，它的效果最好。如果通常不使用自动装配，它可能会使开发人员混淆的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的属性或构造器参数，但你应该在使用它们之前考虑到自动装配的局限性和缺点。

限制	描述
重写的可能性	你可以使用总是重写自动装配的 <code><constructor-arg></code> 和 <code><property></code> 设置来指定依赖关系。
原始数据类型	你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	自动装配不如显式装配精确，所以如果可能的话尽可能使用显式装配。

Spring 自动装配 ‘byName’

这种模式由属性名称指定自动装配。Spring 容器看作 beans，在 XML 配置文件中 beans 的 *auto-wire* 属性设置为 *byName*。然后，它尝试将它的属性与配置文件中定义为相同名称的 beans 进行匹配和连接。如果找到匹配项，它将注入这些 beans，否则，它将抛出异常。

例如，在配置文件中，如果一个 bean 定义设置为自动装配 *byName*，并且它包含 *spellChecker* 属性（即，它有一个 *setSpellChecker(...)* 方法），那么 Spring 就会查找定义名为 *spellChecker* 的 bean，并且用它来设置这个属性。你仍然可以使用 `<property>` 标签连接其余的属性。下面的例子将说明这个概念。

让我们在恰当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，在 <i>Spring Hello World Example</i> 章节中已说明。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>TextEditor</i> ， <i>SpellChecker</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹中创建 Beans 的配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并运行该应用程序，正如下面解释的一样。

这里是 *TextEditor.java* 文件的内容：

```
package com.tutorialspoint;
public class TextEditor {
    private SpellChecker spellChecker;
    private String name;
    public void setSpellChecker( SpellChecker spellChecker ){
        this.spellChecker = spellChecker;
    }
    public SpellChecker getSpellChecker() {
        return spellChecker;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void spellCheck() {
        spellChecker.checkSpelling();
    }
}
```

```
}
}
```

下面是另一个依赖类文件 `SpellChecker.java` 的内容：

```
package com.tutorialspoint;
public class SpellChecker {
    public SpellChecker() {
        System.out.println("Inside SpellChecker constructor." );
    }
    public void checkSpelling() {
        System.out.println("Inside checkSpelling." );
    }
}
```

下面是 `MainApp.java` 文件的内容：

```
package com.tutorialspoint;
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();
    }
}
```

下面是在正常情况下的配置文件 `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">

    <!-- Definition for textEditor bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <property name="spellChecker" ref="spellChecker" />
        <property name="name" value="Generic Text Editor" />
    </bean>

    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
    </bean>

</beans>
```

但是，如果你要使用自动装配 “byName”，那么你的 XML 配置文件将成为如下：

```
<?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 -->
  <bean id="textEditor" class="com.tutorialspoint.TextEditor"
    autowire="byName">
    <property name="name" value="Generic Text Editor" />
  </bean>

  <!-- Definition for spellChecker bean -->
  <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
  </bean>

</beans>
```

一旦你完成了创建源代码和 bean 的配置文件，我们就可以运行该应用程序。如果你的应用程序一切都正常，它将打印下面的消息：

```
Inside SpellChecker constructor.
Inside checkSpelling.
```

Spring 自动装配 ‘byType’

这种模式由属性类型指定自动装配。Spring 容器看作 beans，在 XML 配置文件中 beans 的 *autowire* 属性设置为 *byType*。然后，如果它的 *type* 恰好与配置文件中 beans 名称中的一个相匹配，它将尝试匹配和连接它的属性。如果找到匹配项，它将注入这些 beans，否则，它将抛出异常。

例如，在配置文件中，如果一个 bean 定义设置为自动装配 *byType*，并且它包含 *SpellChecker* 类型的 *spellChecker* 属性，那么 Spring 就会查找定义名为 *SpellChecker* 的 bean，并且用它来设置这个属性。你仍然可以使用 `<property>` 标签连接其余属性。下面的例子将说明这个概念，你会发现和上面的例子没有什么区别，除了 XML 配置文件已经被改变。

让我们在恰当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，在 <i>Spring Hello World Example</i> 章节中已说明。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>TextEditor</i> ， <i>SpellChecker</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹中创建 Beans 的配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并运行该应用程序，正如下面解释的一样。

这里是 *TextEditor.java* 文件的内容：

```
package com.tutorialspoint;

public class TextEditor {
    private SpellChecker spellChecker;
    private String name;
    public void setSpellChecker( SpellChecker spellChecker ) {
        this.spellChecker = spellChecker;
    }
    public SpellChecker getSpellChecker() {
        return spellChecker;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```



```

public void spellCheck() {
    spellChecker.checkSpelling();
}
}

```

下面是另一个依赖类文件 **SpellChecker.java** 的内容：

```

package com.tutorialspoint;
public class SpellChecker {
    public SpellChecker(){
        System.out.println("Inside SpellChecker constructor." );
    }
    public void checkSpelling() {
        System.out.println("Inside checkSpelling." );
    }
}

```

下面是 **MainApp.java** 文件的内容：

```

package com.tutorialspoint;
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();
    }
}

```

下面是在正常情况下的配置文件 **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">

    <!-- Definition for textEditor bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <property name="spellChecker" ref="spellChecker" />
        <property name="name" value="Generic Text Editor" />
    </bean>

```

```

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>

</beans>

```

但是，如果你要使用自动装配 “byType”，那么你的 XML 配置文件将成为如下：

```

<?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 -->
  <bean id="textEditor" class="com.tutorialspoint.TextEditor"
    autowire="byType">
    <property name="name" value="Generic Text Editor" />
  </bean>

  <!-- Definition for spellChecker bean -->
  <bean id="SpellChecker" class="com.tutorialspoint.SpellChecker">
  </bean>

</beans>

```

一旦你完成了创建源代码和 bean 的配置文件，我们就可以运行该应用程序。如果你的应用程序一切都正常，它将打印下面的消息：

```

Inside SpellChecker constructor.
Inside checkSpelling.

```

Spring 由构造函数自动装配

这种模式与 *byType* 非常相似，但它应用于构造器参数。Spring 容器看作 beans，在 XML 配置文件中 beans 的 *autowire* 属性设置为 *constructor*。然后，它尝试把它的构造函数的参数与配置文件中 beans 名称中的一个进行匹配和连线。如果找到匹配项，它会注入这些 bean，否则，它会抛出异常。

例如，在配置文件中，如果一个 bean 定义设置为通过构造函数自动装配，而且它有一个带有 *SpellChecker* 类型的参数之一的构造函数，那么 Spring 就会查找定义名为 *SpellChecker* 的 bean，并用它来设置构造函数的参数。你仍然可以使用 `<constructor-arg>` 标签连接其余属性。下面的例子将说明这个概念。

让我们在恰当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，在 <i>Spring Hello World Example</i> 章节中已说明。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>TextEditor</i> ， <i>SpellChecker</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹中创建 Beans 的配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并运行该应用程序，正如下面解释的一样。

这里是 *TextEditor.java* 文件的内容：

```
package com.tutorialspoint;
public class TextEditor {
    private SpellChecker spellChecker;
    private String name;
    public TextEditor( SpellChecker spellChecker, String name ) {
        this.spellChecker = spellChecker;
        this.name = name;
    }
    public SpellChecker getSpellChecker() {
        return spellChecker;
    }
    public String getName() {
        return name;
    }
    public void spellCheck() {
        spellChecker.checkSpelling();
    }
}
```

下面是另一个依赖类文件 `SpellChecker.java` 的内容：

```
package com.tutorialspoint;
public class SpellChecker {
    public SpellChecker(){
        System.out.println("Inside SpellChecker constructor." );
    }
    public void checkSpelling()
    {
        System.out.println("Inside checkSpelling." );
    }
}
```

下面是 `MainApp.java` 文件的内容：

```
package com.tutorialspoint;
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();
    }
}
```

下面是在正常情况下的配置文件 `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">

    <!-- Definition for textEditor bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
        <constructor-arg ref="spellChecker" />
        <constructor-arg value="Generic Text Editor"/>
    </bean>

    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
    </bean>

</beans>
```

但是，如果你要使用自动装配 “by constructor”，那么你的 XML 配置文件将成为如下：

```
<?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 -->
  <bean id="textEditor" class="com.tutorialspoint.TextEditor"
    autowire="constructor">
    <constructor-arg value="Generic Text Editor"/>
  </bean>

  <!-- Definition for spellChecker bean -->
  <bean id="SpellChecker" class="com.tutorialspoint.SpellChecker">
  </bean>

</beans>
```

一旦你完成了创建源代码和 bean 的配置文件，我们就可以运行该应用程序。如果你的应用程序一切都正常，它将打印下面的消息：

```
Inside SpellChecker constructor.
Inside checkSpelling.
```



15

基于注解的配置



从 Spring 2.5 开始就可以使用注解来配置依赖注入。而不是采用 XML 来描述一个 bean 连线，你可以使用相关类，方法或字段声明的注解，将 bean 配置移动到组件类本身。

在 XML 注入之前进行注解注入，因此后者的配置将通过两种方式的属性连线被前者重写。

注解连线在默认情况下在 Spring 容器中不打开。因此，在可以使用基于注解的连线之前，我们将需要在我们的 Spring 配置文件中启用它。所以如果你想在 Spring 应用程序中使用的任何注解，可以考虑到下面的配置文件。

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

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  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>
```

一旦 被配置后，你就可以开始注解你的代码，表明 Spring 应该自动连接值到属性，方法和构造函数。让我们来看看几个重要的注解，并且了解它们是如何工作的：

序号	注解 & 描述
1	@Required (annotation-based-configuration/spring-required-annotation.md) @Required 注解应用于 bean 属性的 setter 方法。
2	@Autowired (annotation-based-configuration/spring-autowired-annotation.md) @Autowired 注解可以应用到 bean 属性的 setter 方法，非 setter 方法，构造函数和属性。
3	@Qualifier (annotation-based-configuration/spring-qualifier-annotation.md) 通过指定确切的将被连线的 bean，@Autowired 和 @Qualifier 注解可以用来删除混乱。
4	JSR-250 Annotations (annotation-based-configuration/spring-jsr250-annotation.md)

Spring 支持 JSR-250 的基础的注解，其中包括了 `@Resource`，`@PostConstruct` 和 `@PreDestroy` 注解。

Spring @Required 注释

@Required 注释应用于 bean 属性的 setter 方法，它表明受影响的 bean 属性在配置时必须放在 XML 配置文件中，否则容器就会抛出一个 BeanInitializationException 异常。下面显示的是一个使用 @Required 注释的示例。

示例：

让我们使 Eclipse IDE 处于工作状态，请按照下列步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 src 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。
3	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>Student</i> 和 <i>MainApp</i> 。
4	在 src 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

下面是 Student.java 文件的内容：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Required;
public class Student {
    private Integer age;
    private String name;
    @Required
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    @Required
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

```
}
}
```

下面是 MainApp.java 文件的内容：

```
package com.tutorialspoint;
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");
        Student student = (Student) context.getBean("student");
        System.out.println("Name : " + student.getName() );
        System.out.println("Age : " + student.getAge() );
    }
}
```

下面是配置文件 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"
    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/>

    <!-- Definition for student bean -->
    <bean id="student" class="com.tutorialspoint.Student">
        <property name="name" value="Zara" />

        <!-- try without passing age and check the result -->
        <!-- property name="age" value="11"-->
    </bean>

</beans>
```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将引起 *BeanInitializationException* 异常，并且会输出一下错误信息和其他日志消息：

```
Property 'age' is required for bean 'student'
```

下一步，在你按照如下所示从 “age” 属性中删除了注释，你可以尝试运行上面的示例：

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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/>

<!-- Definition for student bean -->
<bean id="student" class="com.tutorialspoint.Student">
  <property name="name" value="Zara" />
  <property name="age" value="11"/>
</bean>

</beans>
```

现在上面的示例将产生如下结果：

```
Name : Zara
Age : 11
```

Spring @Autowired 注释

@Autowired 注释对在哪里和如何完成自动连接提供了更多的细微的控制。

@Autowired 注释可以在 setter 方法中被用于自动连接 bean，就像 @Autowired 注释，容器，一个属性或者任意命名的可能带有多个参数的方法。

Setter 方法中的 @Autowired

你可以在 XML 文件中的 setter 方法中使用 @Autowired 注释来除去 元素。当 Spring 遇到一个在 setter 方法中使用的 @Autowired 注释，它会在方法中视图执行 byType 自动连接。

示例

让我们使 Eclipse IDE 处于工作状态，然后按照如下步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 <i>src</i> 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。
3	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>TextEditor</i> , <i>SpellChecker</i> 和 <i>MainApp</i> 。
4	在 <i>src</i> 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

这里是 *TextEditor.java* 文件的内容：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
    private SpellChecker spellChecker;
    @Autowired
    public void setSpellChecker( SpellChecker spellChecker ){
        this.spellChecker = spellChecker;
    }
    public SpellChecker getSpellChecker( ) {
        return spellChecker;
    }
}
```

```

public void spellCheck() {
    spellChecker.checkSpelling();
}
}

```

下面是另一个依赖的类文件 `SpellChecker.java` 的内容：

```

package com.tutorialspoint;
public class SpellChecker {
    public SpellChecker(){
        System.out.println("Inside SpellChecker constructor." );
    }
    public void checkSpelling(){
        System.out.println("Inside checkSpelling." );
    }
}

```

下面是 `MainApp.java` 文件的内容：

```

package com.tutorialspoint;
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();
    }
}

```

下面是配置文件 `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"
    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/>

    <!-- Definition for textEditor bean without constructor-arg -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
    </bean>

    <!-- Definition for spellChecker bean -->
    <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
    </bean>

```

```
</beans>
```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Inside SpellChecker constructor.
Inside checkSpelling.
```

属性中的 @Autowired

你可以在属性中使用 @Autowired 注释来除去 setter 方法。当时使用 为自动连接属性传递的时候，Spring 会将这些传递过来的值或者引用自动分配给那些属性。所以利用在属性中 @Autowired 的用法，你的 `** TextEditor.java**` 文件将变成如下所示：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
    @Autowired
    private SpellChecker spellChecker;
    public TextEditor() {
        System.out.println("Inside TextEditor constructor.");
    }
    public SpellChecker getSpellChecker() {
        return spellChecker;
    }
    public void spellCheck(){
        spellChecker.checkSpelling();
    }
}
```

下面是配置文件 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"
    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/>

    <!-- Definition for textEditor bean -->
    <bean id="textEditor" class="com.tutorialspoint.TextEditor">
    </bean>
```

```

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>

</beans>

```

一旦你在源文件和 bean 配置文件中完成了上面两处改变，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```

Inside TextEditor constructor.
Inside SpellChecker constructor.
Inside checkSpelling.

```

构造函数中的 @Autowired

你也可以在构造函数中使用 @Autowired。一个构造函数 @Autowired 说明当创建 bean 时，即使在 XML 文件中没有使用 元素配置 bean，构造函数也会被自动连接。让我们检查一下下面的示例。

这里是 TextEditor.java 文件的内容：

```

package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
    private SpellChecker spellChecker;
    @Autowired
    public TextEditor(SpellChecker spellChecker){
        System.out.println("Inside TextEditor constructor.");
        this.spellChecker = spellChecker;
    }
    public void spellCheck(){
        spellChecker.checkSpelling();
    }
}

```

下面是配置文件 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"
    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/>

    <!-- Definition for textEditor bean without constructor-arg -->

```

```
<bean id="textEditor" class="com.tutorialspoint.TextEditor">
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>

</beans>
```

一旦你在源文件和 bean 配置文件中完成了上面两处改变，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Inside TextEditor constructor.
Inside SpellChecker constructor.
Inside checkSpelling.
```

@Autowired 的 (required=false) 选项

默认情况下，@Autowired 注释意味着依赖是必须的，它类似于 @Required 注释，然而，你可以使用 @Autowired 的 (required=false) 选项关闭默认行为。

即使你不为 age 属性传递任何参数，下面的示例也会成功运行，但是对于 name 属性则需要一个参数。你可以自己尝试一下这个示例，因为除了只有 Student.java 文件被修改以外，它和 @Required 注释示例是相似的。

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class Student {
    private Integer age;
    private String name;
    @Autowired(required=false)
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    @Autowired
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```


Spring @Qualifier 注释

可能会有这样一种情况，当你创建多个具有相同类型的 bean 时，并且想要用一个属性只为它们其中的一个进行装配，在这种情况下，你可以使用 @Qualifier 注释和 @Autowired 注释通过指定哪一个真正的 bean 将会被装配来消除混乱。下面显示的是使用 @Qualifier 注释的一个示例。

示例

让我们使 Eclipse IDE 处于工作状态，请按照下列步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 src 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。
3	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>Student</i> , <i>Profile</i> 和 <i>MainApp</i> 。
4	在 src 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

这里是 *Student.java* 文件的内容：

```
package com.tutorialspoint;
public class Student {
    private Integer age;
    private String name;
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

这里是 *Profile.java* 文件的内容：

```

package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class Profile {
    @Autowired
    @Qualifier("student1")
    private Student student;
    public Profile(){
        System.out.println("Inside Profile constructor." );
    }
    public void printAge() {
        System.out.println("Age : " + student.getAge() );
    }
    public void printName() {
        System.out.println("Name : " + student.getName() );
    }
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;
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");
        Profile profile = (Profile) context.getBean("profile");
        profile.printAge();
        profile.printName();
    }
}

```

考虑下面配置文件 Beans.xml 的示例：

```

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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/>

    <!-- Definition for profile bean -->
    <bean id="profile" class="com.tutorialspoint.Profile">
    </bean>

    <!-- Definition for student1 bean -->

```

```
<bean id="student1" class="com.tutorialspoint.Student">
  <property name="name" value="Zara" />
  <property name="age" value="11"/>
</bean>

<!-- Definition for student2 bean -->
<bean id="student2" class="com.tutorialspoint.Student">
  <property name="name" value="Nuha" />
  <property name="age" value="2"/>
</bean>

</beans>
```

一旦你在源文件和 bean 配置文件中完成了上面两处改变，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Inside Profile constructor.
Age : 11
Name : Zara
```

Spring JSR-250 注释

Spring 还使用基于 JSR-250 注释，它包括 `@PostConstruct`，`@PreDestroy` 和 `@Resource` 注释。因为你已经有了其他的选择，尽管这些注释并不是真正所需要的，但是关于它们仍然让我给出一个简短的介绍。

@PostConstruct 和 @PreDestroy 注释：

为了定义一个 bean 的安装和卸载，我们使用 `init-method` 和/或 `destroy-method` 参数简单的声明一下。`init-method` 属性指定了一个方法，该方法在 bean 的实例化阶段会立即被调用。同样地，`destroy-method` 指定了一个方法，该方法只在一个 bean 从容器中删除之前被调用。

你可以使用 `@PostConstruct` 注释作为初始化回调函数的一个替代，`@PreDestroy` 注释作为销毁回调函数的一个替代，其解释如下示例所示。

示例

让我们使 Eclipse IDE 处于工作状态，请按照下列步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 <code>src</code> 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。
3	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>HelloWorld</i> 和 <i>MainApp</i> 。
4	在 <code>src</code> 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

这里是 `HelloWorld.java` 文件的内容：

```
package com.tutorialspoint;
import javax.annotation.*;
public class HelloWorld {
    private String message;
    public void setMessage(String message){
        this.message = message;
    }
    public String getMessage(){
```

```

    System.out.println("Your Message : " + message);
    return message;
}
@PostConstruct
public void init(){
    System.out.println("Bean is going through init.");
}
@PreDestroy
public void destroy(){
    System.out.println("Bean will destroy now.");
}
}

```

下面是 `MainApp.java` 文件的内容。这里你需要注册一个关闭钩 `registerShutdownHook()` 方法，该方法在 `AbstractApplicationContext` 类中被声明。这将确保一个完美的关闭并调用相关的销毁方法。

```

package com.tutorialspoint;
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();
    }
}

```

下面是配置文件 `Beans.xml`，该文件在初始化和销毁方法中需要使用。

```

<?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 id="helloWorld"
        class="com.tutorialspoint.HelloWorld"
        init-method="init" destroy-method="destroy">
        <property name="message" value="Hello World!"/>
    </bean>

</beans>

```

一旦你在源文件和 bean 配置文件中完成了上面两处改变，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Bean is going through init.  
Your Message : Hello World!  
Bean will destroy now.
```

@Resource 注释：

你可以在字段中或者 setter 方法中使用 **@Resource** 注释，它和在 Java EE 5 中的运作是一样的。**@Resource** 注释使用一个 ‘name’ 属性，该属性以一个 bean 名称的形式被注入。你可以说，它遵循 by-name 自动连接语义，如下面的示例所示：

```
package com.tutorialspoint;  
import javax.annotation.Resource;  
public class TextEditor {  
    private SpellChecker spellChecker;  
    @Resource(name= "spellChecker")  
    public void setSpellChecker( SpellChecker spellChecker ){  
        this.spellChecker = spellChecker;  
    }  
    public SpellChecker getSpellChecker(){  
        return spellChecker;  
    }  
    public void spellCheck(){  
        spellChecker.checkSpelling();  
    }  
}
```

如果没有明确地指定一个 ‘name’，默认名称源于字段名或者 setter 方法。在字段的情况下，它使用的是字段名；在一个 setter 方法情况下，它使用的是 bean 属性名称。



16

基于 Java 的配置



到目前为止，你已经看到如何使用 XML 配置文件来配置 Spring bean。如果你熟悉使用 XML 配置，那么我会说，不需要再学习如何进行基于 Java 的配置是，因为你要达到相同的结果，可以使用其他可用的配置。

基于 Java 的配置选项，可以使你在不用配置 XML 的情况下编写大多数的 Spring，但是一些有帮助的基于 Java 的注解，解释如下：

@Configuration 和 @Bean 注解

带有 @Configuration 的注解类表示这个类可以使用 Spring IoC 容器作为 bean 定义的来源。@Bean 注解告诉 Spring，一个带有 @Bean 的注解方法将返回一个对象，该对象应该被注册为在 Spring 应用程序上下文中的 bean。最简单可行的 @Configuration 类如下所示：

```
package com.tutorialspoint;
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.tutorialspoint.HelloWorld" />
</beans>
```

在这里，带有 @Bean 注解的方法名称作为 bean 的 ID，它创建并返回实际的 bean。你的配置类可以声明多个 @Bean。一旦定义了配置类，你就可以使用 *AnnotationConfigApplicationContext* 来加载并把他们提供给 Spring 容器，如下所示：

```
public static void main(String[] args) {
    ApplicationContext ctx =
        new AnnotationConfigApplicationContext(HelloWorldConfig.class);
    HelloWorld helloWorld = ctx.getBean(HelloWorld.class);
    helloWorld.setMessage("Hello World!");
    helloWorld.getMessage();
}
```

你可以加载各种配置类，如下所示：


```

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

让我们在恰当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库,解释见 <i>Spring Hello World Example</i> 章节。
3	因为你是使用基于 java 的注解，所以你还需要添加来自 Java 安装目录的 <i>CGLIB.jar</i> 和可以从 <i>asm.ow 2.org</i> 中下载的 <i>ASM.jar</i> 库。
4	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>HelloWorldConfig</i> 、 <i>HelloWorld</i> 和 <i>MainApp</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 *HelloWorldConfig.java* 文件的内容：

```

package com.tutorialspoint;
import org.springframework.context.annotation.*;
@Configuration
public class HelloWorldConfig {
    @Bean
    public HelloWorld helloWorld(){
        return new HelloWorld();
    }
}

```

这里是 *HelloWorld.java* 文件的内容：

```

package com.tutorialspoint;

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 MainApp.java 文件的内容：

```

package com.tutorialspoint;

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();
    }
}

```

一旦你完成了创建所有的源文件并添加所需的额外的库后，我们就可以运行该应用程序。你应该注意这里不需要配置文件。如果你的应用程序一切都正常，将输出以下信息：

```
Your Message : Hello World!
```

注入 Bean 的依赖性

当 @Beans 依赖对方时，表达这种依赖性非常简单，只要有一个 bean 方法调用另一个，如下所示：

```

package com.tutorialspoint;
import org.springframework.context.annotation.*;
@Configuration
public class AppConfig {
    @Bean
    public Foo foo() {
        return new Foo(bar());
    }
}

```

```
@Bean
public Bar bar() {
    return new Bar();
}
}
```

这里，foo Bean 通过构造函数注入来接收参考基准。现在，让我们看到一个正在执行的例子：

例子:

让我们在恰当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库,解释见 <i>Spring Hello World Example</i> 章节。
3	因为你是使用基于 java 的注解，所以你还需要添加来自 Java 安装目录的 <i>CGLIB.jar</i> 和可以从 <i>asm.ow2.org</i> 中下载的 <i>ASM.jar</i> 库。
4	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>TextEditorConfig</i> 、 <i>TextEditor</i> 、 <i>SpellChecker</i> 和 <i>MainApp</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 *TextEditorConfig.java* 文件的内容：

```
package com.tutorialspoint;
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.tutorialspoint;
public class TextEditor {
    private SpellChecker spellChecker;
```

```

public TextEditor(SpellChecker spellChecker){
    System.out.println("Inside TextEditor constructor." );
    this.spellChecker = spellChecker;
}
public void spellCheck(){
    spellChecker.checkSpelling();
}
}

```

下面是另一个依赖的类文件 `SpellChecker.java` 的内容：

```

package com.tutorialspoint;
public class SpellChecker {
    public SpellChecker(){
        System.out.println("Inside SpellChecker constructor." );
    }
    public void checkSpelling(){
        System.out.println("Inside checkSpelling." );
    }
}

```

下面是 `MainApp.java` 文件的内容：

```

package com.tutorialspoint;

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();
    }
}

```

一旦你完成了创建所有的源文件并添加所需的额外的库后，我们就可以运行该应用程序。你应该注意这里不需要配置文件。如果你的应用程序一切都正常，将输出以下信息：

```

Inside SpellChecker constructor.
Inside TextEditor constructor.
Inside checkSpelling.

```

@Import 注解:

@import 注解允许从另一个配置类中加载 @Bean 定义。考虑 ConfigA 类，如下所示：

```
@Configuration
public class ConfigA {
    @Bean
    public A a() {
        return new A();
    }
}
```

你可以在另一个 Bean 声明中导入上述 Bean 声明，如下所示：

```
@Configuration
@Import(ConfigA.class)
public class ConfigB {
    @Bean
    public B a() {
        return new A();
    }
}
```

现在，当实例化上下文时，不需要同时指定 ConfigA.class 和 ConfigB.class，只有 ConfigB 类需要提供，如下所示：

```
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的回调方法，就像在 bean 元素中 Spring 的 XML 的初始化方法和销毁方法的属性：

```
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 的范围：

默认范围是单实例，但是你可以重写带有 `@Scope` 注解的该方法，如下所示：

```
@Configuration  
public class AppConfig {  
    @Bean  
    @Scope("prototype")  
    public Foo foo() {  
        return new Foo();  
    }  
}
```



T

17



Spring 中的事件处理



你已经看到了在所有章节中 Spring 的核心是 `ApplicationContext`，它负责管理 beans 的完整生命周期。当加载 beans 时，`ApplicationContext` 发布某些类型的事件。例如，当上下文启动时，`ContextStartedEvent` 发布，当上下文停止时，`ContextStoppedEvent` 发布。

通过 `ApplicationEvent` 类和 `ApplicationListener` 接口来提供在 `ApplicationContext` 中处理事件。如果一个 bean 实现 `ApplicationListener`，那么每次 `ApplicationEvent` 被发布到 `ApplicationContext` 上，那个 bean 会被通知。

Spring 提供了以下的标准事件：

序号	Spring 内置事件 & 描述
1	<div>ContextRefreshedEvent</div> <div><i>ApplicationContext</i> 被初始化或刷新时，该事件被发布。这也可以在 <i>ConfigurableApplicationContext</i> 接口中使用 <code>refresh()</code> 方法来发生。</div>
2	<div>ContextStartedEvent</div> <div>当使用 <i>ConfigurableApplicationContext</i> 接口中的 <code>start()</code> 方法启动 <i>ApplicationContext</i> 时，该事件被发布。你可以调查你的数据库，或者你可以在接受到这个事件后重启任何停止的应用程序。</div>
3	<div>ContextStoppedEvent</div> <div>当使用 <i>ConfigurableApplicationContext</i> 接口中的 <code>stop()</code> 方法停止 <i>ApplicationContext</i> 时，发布这个事件。你可以在接受到这个事件后做必要的清理的工作。</div>
4	<div>ContextClosedEvent</div> <div>当使用 <i>ConfigurableApplicationContext</i> 接口中的 <code>close()</code> 方法关闭 <i>ApplicationContext</i> 时，该事件被发布。一个已关闭的上下文到达生命周期末端；它不能被刷新或重启。</div>
5	<div>RequestHandledEvent</div> <div>这是一个 web-specific 事件，告诉所有 bean HTTP 请求已经被服务。</div>

由于 Spring 的事件处理是单线程的，所以如果一个事件被发布，直至并且除非所有的接收者得到的该消息，该进程被阻塞并且流程将不会继续。因此，如果事件处理被使用，在设计应用程序时应注意。

监听上下文事件

为了监听上下文事件，一个 bean 应该实现只有一个方法 `onApplicationEvent()` 的 `ApplicationListener` 接口。因此，我们写一个例子来看看事件是如何传播的，以及如何可以用代码来执行基于某些事件所需的任务。

让我们在恰当的位置使用 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	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <code>src</code> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	在 <i>com.tutorialspoint</i> 包中创建 Java 类 <i>HelloWorld</i> 、 <i>CStartEventHandler</i> 、 <i>CStopEventHandler</i> 和 <i>MainApp</i> 。
4	在 <code>src</code> 文件夹中创建 Bean 的配置文件 <i>Beans.xml</i> 。
5	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这里是 `HelloWorld.java` 文件的内容：

```
package com.tutorialspoint;
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 `CStartEventHandler.java` 文件的内容：

```
package com.tutorialspoint;
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");
    }
}
```

```

    }
}

```

下面是 CStopEventHandler.java 文件的内容：

```

package com.tutorialspoint;
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");
    }
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;

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();
    }
}

```

下面是配置文件 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.tutorialspoint.HelloWorld">
  <property name="message" value="Hello World!"/>
</bean>

<bean id="cStartEventHandler"
  class="com.tutorialspoint.CStartEventHandler"/>

<bean id="cStopEventHandler"
  class="com.tutorialspoint.CStopEventHandler"/>

</beans>
```

一旦你完成了创建源和 bean 的配置文件，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下消息：

```
ContextStartedEvent Received
Your Message : Hello World!
ContextStoppedEvent Received
```



18

Spring 中的自定义事件



编写和发布自己的自定义事件有许多步骤。按照在这一章给出的说明来编写，发布和处理自定义 Spring 事件。

步骤	描述
1	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	通过扩展 <i>ApplicationEvent</i> ，创建一个事件类 <i>CustomEvent</i> 。这个类必须定义一个默认的构造函数，它应该从 <i>ApplicationEvent</i> 类中继承的构造函数。
4	一旦定义事件类，你可以从任何类中发布它，假定 <i>EventClassPublisher</i> 实现了 <i>ApplicationEventPublisherAware</i> 。你还需要在 XML 配置文件中声明这个类作为一个 bean，之所以容器可以识别 bean 作为事件发布者，是因为它实现了 <i>ApplicationEventPublisherAware</i> 接口。
5	发布的事件可以在一个类中被处理，假定 <i>EventClassHandler</i> 实现了 <i>ApplicationListener</i> 接口，而且实现了自定义事件的 <i>onApplicationEvent</i> 方法。
6	在 <i>src</i> 文件夹中创建 bean 的配置文件 <i>Beans.xml</i> 和 <i>MainApp</i> 类，它可以作为一个 Spring 应用程序来运行。
7	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这个是 *CustomEvent.java* 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationEvent;
public class CustomEvent extends ApplicationEvent{
    public CustomEvent(Object source) {
        super(source);
    }
    public String toString(){
        return "My Custom Event";
    }
}
```

下面是 *CustomEventPublisher.java* 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.ApplicationEventPublisherAware;
public class CustomEventPublisher
    implements ApplicationEventPublisherAware {
    private ApplicationEventPublisher publisher;
    public void setApplicationEventPublisher
        (ApplicationEventPublisher publisher){
        this.publisher = publisher;
    }
    public void publish() {
        CustomEvent ce = new CustomEvent(this);
```

```

    publisher.publishEvent(ce);
}
}

```

下面是 CustomEventHandler.java 文件的内容：

```

package com.tutorialspoint;
import org.springframework.context.ApplicationListener;
public class CustomEventHandler
    implements ApplicationListener<CustomEvent>{
    public void onApplicationEvent(CustomEvent event) {
        System.out.println(event.toString());
    }
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;
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");
        CustomEventPublisher cvp =
            (CustomEventPublisher) context.getBean("customEventPublisher");
        cvp.publish();
        cvp.publish();
    }
}

```

下面是配置文件 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="customEventHandler"
        class="com.tutorialspoint.CustomEventHandler"/>

    <bean id="customEventPublisher"
        class="com.tutorialspoint.CustomEventPublisher"/>

```

```
</beans>
```

一旦你完成了创建源和 bean 的配置文件后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```
My Custom Event  
My Custom Event
```



19

Spring 框架的 AOP



Spring 框架的一个关键组件是面向方面的编程(AOP)框架。面向方面的编程需要把程序逻辑分解成不同的部分称为所谓的关注点。跨一个应用程序的多个点的功能被称为横切关注点，这些横切关注点在概念上独立于应用程序的业务逻辑。有各种各样的常见的很好的方面的例子，如日志记录、审计、声明式事务、安全性和缓存等。

在 OOP 中，关键单元模块度是类，而在 AOP 中单元模块度是方面。依赖注入帮助你对应用程序对象相互解耦和 AOP 可以帮助你从它们所影响的对象中对横切关注点解耦。AOP 是像编程语言的触发物，如 Perl，.NET，Java 或者其他。

Spring AOP 模块提供拦截器来拦截一个应用程序，例如，当执行一个方法时，你可以在方法执行之前或之后添加额外的功能。

AOP 术语

在我们开始使用 AOP 工作之前，让我们熟悉一下 AOP 概念和术语。这些术语并不特定于 Spring，而是与 AOP 有关的。

项	描述
Aspect	一个模块具有一组提供横切需求的 APIs。例如，一个日志模块为了记录日志将被 AOP 方面调用。应用程序可以拥有任意数量的方面，这取决于需求。
Join point	在你的应用程序中它代表一个点，你可以在插件 AOP 方面。你也能说，它是在实际的应用程序中，其中一个操作将使用 Spring AOP 框架。
Advice	这是实际行动之前或之后执行的方法。这是在程序执行期间通过 Spring AOP 框架实际被调用的代码。
Pointcut	这是一组一个或多个连接点，通知应该被执行。你可以使用表达式或模式指定切入点正如我们将在 AOP 的例子中看到的。
Introduction	引用允许你添加新方法或属性到现有的类中。
Target object	被一个或者多个方面所通知的对象，这个对象永远是一个被代理对象。也称为被通知对象。
Weaving	Weaving 把方面连接到其它的应用程序类型或者对象上，并创建一个被通知的对象。这些可以在编译时，类加载时和运行时完成。

通知的类型

Spring 方面可以使用下面提到的五种通知工作：

通知	描述
前置通知	在一个方法执行之前，执行通知。
后置通知	在一个方法执行之后，不考虑其结果，执行通知。
返回后通知	在一个方法执行之后，只有在方法成功完成时，才能执行通知。

抛出异常后通知	在一个方法执行之后，只有在方法退出抛出异常时，才能执行通知。
环绕通知	在建议方法调用之前和之后，执行通知。

实现自定义方面

Spring 支持 `@AspectJ annotation style` 的方法和基于模式的方法来实现自定义方面。这两种方法已经在下面两个子节进行了详细解释。

方法	描述
XML Schema based (aop-with-spring-frameswork/xml-schema-based-aop-with-spring.md)	方面是使用常规类以及基于配置的 XML 来实现的。
@AspectJ based (aop-with-spring-framework/aspectj-based-aop-with-spring.md)	@AspectJ 引用一种声明方面的风格作为带有 Java 5 注释的常规 Java 类注释。

Spring 中基于 AOP 的 XML 架构

为了在本节的描述中使用 aop 命名空间标签，你需要导入 spring-aop 架构，如下所述：

```
<?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:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

  <!-- bean definition & AOP specific configuration -->

</beans>
```

你还需要在你的应用程序的 CLASSPATH 中使用以下 AspectJ 库文件。这些库文件在一个 AspectJ 装置的 ‘lib’ 目录中是可用的，否则你可以在 Internet 中下载它们。

- aspectjrt.jar
- aspectjweaver.jar
- aspectj.jar
- aopalliance.jar

声明一个 aspect

一个 aspect 是使用 元素声明的，支持的 bean 是使用 ref 属性引用的，如下所示：

```
<aop:config>
  <aop:aspect id="myAspect" ref="aBean">
    ...
  </aop:aspect>
</aop:config>
<bean id="aBean" class="...">
  ...
</bean>
```

这里，“aBean” 将被配置和依赖注入，就像前面的章节中你看到的其他的 Spring bean 一样。

声明一个切入点

一个切入点有助于确定使用不同建议执行的感兴趣的连接点（即方法）。在处理基于配置的 XML 架构时，切入点将会按照如下所示定义：

```
<aop:config>
  <aop:aspect id="myAspect" ref="aBean">
    <aop:pointcut id="businessService"
      expression="execution(* com.xyz.myapp.service.*(..))"/>
    ...
  </aop:aspect>
</aop:config>
<bean id="aBean" class="...">
...
</bean>
```

下面的示例定义了一个名为 “businessService” 的切入点，该切入点将与 com.tutorialspoint 包下的 Student 类中的 getName() 方法相匹配：

```
<aop:config>
  <aop:aspect id="myAspect" ref="aBean">
    <aop:pointcut id="businessService"
      expression="execution(* com.tutorialspoint.Student.getName(..))"/>
    ...
  </aop:aspect>
</aop:config>
<bean id="aBean" class="...">
...
</bean>
```

声明建议

你可以使用 <aop:{ADVICE NAME}> 元素在一个 中声明五个建议中的任何一个，如下所示：

```
<aop:config>
  <aop:aspect id="myAspect" ref="aBean">
    <aop:pointcut id="businessService"
      expression="execution(* com.xyz.myapp.service.*(..))"/>
    <!-- a before advice definition -->
    <aop:before pointcut-ref="businessService"
      method="doRequiredTask"/>
    <!-- an after advice definition -->
```

```

<aop:after pointcut-ref="businessService"
    method="doRequiredTask"/>
<!-- an after-returning advice definition -->
<!--The doRequiredTask method must have parameter named retVal -->
<aop:after-returning pointcut-ref="businessService"
    returning="retVal"
    method="doRequiredTask"/>
<!-- an after-throwing advice definition -->
<!--The doRequiredTask method must have parameter named ex -->
<aop:after-throwing pointcut-ref="businessService"
    throwing="ex"
    method="doRequiredTask"/>
<!-- an around advice definition -->
<aop:around pointcut-ref="businessService"
    method="doRequiredTask"/>
...
</aop:aspect>
</aop:config>
<bean id="aBean" class="...">
...
</bean>

```

你可以对不同的建议使用相同的 `doRequiredTask` 或者不同的方法。这些方法将会作为 `aspect` 模块的一部分来定义。

基于 AOP 的 XML 架构的示例

为了理解上面提到的基于 AOP 的 XML 架构的概念，让我们编写一个示例，可以实现几个建议。为了在我们的示例中使用几个建议，让我们使 Eclipse IDE 处于工作状态，然后按照如下步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 <code>src</code> 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。
3	在项目中添加 Spring AOP 指定的库文件 <i>aspectjrt.jar</i> ， <i>aspectjweaver.jar</i> 和 <i>aspectj.jar</i> 。
4	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>Logging</i> ， <i>Student</i> 和 <i>MainApp</i> 。
5	在 <code>src</code> 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
6	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

这里是 *Logging.java* 文件的内容。这实际上是 `aspect` 模块的一个示例，它定义了在各个点调用的方法。

```

package com.tutorialspoint;
public class Logging {
    /**
     * This is the method which I would like to execute
     * before a selected method execution.
     */
    public void beforeAdvice(){
        System.out.println("Going to setup student profile.");
    }
    /**
     * This is the method which I would like to execute
     * after a selected method execution.
     */
    public void afterAdvice(){
        System.out.println("Student profile has been setup.");
    }
    /**
     * This is the method which I would like to execute
     * when any method returns.
     */
    public void afterReturningAdvice(Object retVal){
        System.out.println("Returning:" + retVal.toString() );
    }
    /**
     * This is the method which I would like to execute
     * if there is an exception raised.
     */
    public void AfterThrowingAdvice(IllegalArgumentException ex){
        System.out.println("There has been an exception: " + ex.toString());
    }
}

```

下面是 Student.java 文件的内容：

```

package com.tutorialspoint;
public class Student {
    private Integer age;
    private String name;
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        System.out.println("Age : " + age );
        return age;
    }
    public void setName(String name) {

```

```

    this.name = name;
}
public String getName() {
    System.out.println("Name : " + name );
    return name;
}
public void printThrowException(){
    System.out.println("Exception raised");
    throw new IllegalArgumentException();
}
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;
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");
        Student student = (Student) context.getBean("student");
        student.getName();
        student.getAge();
        student.printThrowException();
    }
}

```

下面是配置文件 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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <aop:config>
        <aop:aspect id="log" ref="logging">
            <aop:pointcut id="selectAll"
                expression="execution(* com.tutorialspoint.*(..))"/>
            <aop:before pointcut-ref="selectAll" method="beforeAdvice"/>
            <aop:after pointcut-ref="selectAll" method="afterAdvice"/>
            <aop:after-returning pointcut-ref="selectAll"
                returning="retVal"
                method="afterReturningAdvice"/>
            <aop:after-throwing pointcut-ref="selectAll"
                throwing="ex"
                method="AfterThrowingAdvice"/>
        </aop:aspect>
    
```

```

</aop:config>

<!-- Definition for student bean -->
<bean id="student" class="com.tutorialspoint.Student">
  <property name="name" value="Zara" />
  <property name="age" value="11"/>
</bean>

<!-- Definition for logging aspect -->
<bean id="logging" class="com.tutorialspoint.Logging"/>

</beans>

```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```

Going to setup student profile.
Name : Zara
Student profile has been setup.
Returning:Zara
Going to setup student profile.
Age : 11
Student profile has been setup.
Returning:11
Going to setup student profile.
Exception raised
Student profile has been setup.
There has been an exception: java.lang.IllegalArgumentException
.....
other exception content

```

让我们来解释一下上面定义的在 com.tutorialspoint 中 选择所有方法的 。让我们假设一下，你想要在一个特殊的方法之前或者之后执行你的建议，你可以通过替换使用真实类和方法名称的切入点定义中的星号（*）来定义你的切入点来缩短你的执行。

```

<?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:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

  <aop:config>
    <aop:aspect id="log" ref="logging">
      <aop:pointcut id="selectAll"
        expression="execution(* com.tutorialspoint.Student.getName(..))"/>
      <aop:before pointcut-ref="selectAll" method="beforeAdvice"/>
      <aop:after pointcut-ref="selectAll" method="afterAdvice"/>
    </aop:aspect>
  </aop:config>

  <!-- Definition for student bean -->

```



```
<bean id="student" class="com.tutorialspoint.Student">
  <property name="name" value="Zara" />
  <property name="age" value="11"/>
</bean>

<!-- Definition for logging aspect -->
<bean id="logging" class="com.tutorialspoint.Logging"/>

</beans>
```

如果你想要执行通过这些更改之后的示例应用程序，这将会输出以下消息：

```
Going to setup student profile.
Name : Zara
Student profile has been setup.
Age : 11
Exception raised
.....
other exception content
```

Spring 中基于 AOP 的 @AspectJ

@AspectJ 作为通过 Java 5 注释注释的普通的 Java 类，它指的是声明 aspects 的一种风格。通过在你的基于架构的 XML 配置文件中包含以下元素，@AspectJ 支持是可用的。

```
<aop:aspectj-autoproxy/>
```

你还需要在你的应用程序的 CLASSPATH 中使用以下 AspectJ 库文件。这些库文件在一个 AspectJ 装置的 ‘lib’ 目录中是可用的，否则你可以在 Internet 中下载它们。

- aspectjrt.jar
- aspectjweaver.jar
- aspectj.jar
- aopalliance.jar

声明一个 aspect

Aspects 类和其他任何正常的 bean 一样，除了它们将会用 @AspectJ 注释之外，它和其他类一样可能有方法和字段，如下所示：

```
package org.xyz;  
import org.aspectj.lang.annotation.Aspect;  
@Aspect  
public class AspectModule {  
}
```

它们将在 XML 中按照如下进行配置，就和其他任何 bean 一样：

```
<bean id="myAspect" class="org.xyz.AspectModule">  
  <!-- configure properties of aspect here as normal -->  
</bean>
```

声明一个切入点

一个切入点有助于确定使用不同建议执行的感兴趣的连接点（即方法）。在处理基于配置的 XML 架构时，切入点的声明有两个部分：

- 一个切入点表达式决定了我们感兴趣的哪个方法会真正被执行。
- 一个切入点标签包含一个名称和任意数量的参数。方法的真正内容是不相干的，并且实际上它应该是空的。

下面的示例中定义了一个名为 ‘businessService’ 的切入点，该切入点将与 com.tutorialspoint 包下的类中可用的每一个方法相匹配：

```
import org.aspectj.lang.annotation.Pointcut;
@Pointcut("execution(* com.xyz.myapp.service.*(..))") // expression
private void businessService() {} // signature
```

下面的示例中定义了一个名为 ‘getName’ 的切入点，该切入点将与 com.tutorialspoint 包下的 Student 类中的 getName() 方法相匹配：

```
import org.aspectj.lang.annotation.Pointcut;
@Pointcut("execution(* com.tutorialspoint.Student.getName(..))")
private void getName() {}
```

声明建议

你可以使用 @{ADVICE-NAME} 注释声明五个建议中的任意一个，如下所示。这假设你已经定义了一个切入点标签方法 businessService()：

```
@Before("businessService()")
public void doBeforeTask(){
    ...
}
@After("businessService()")
public void doAfterTask(){
    ...
}
@AfterReturning(pointcut = "businessService()", returning="retVal")
public void doAfterReturningTask(Object retVal){
    // you can intercept retVal here.
    ...
}
@AfterThrowing(pointcut = "businessService()", throwing="ex")
public void doAfterThrowingTask(Exception ex){
    // you can intercept thrown exception here.
    ...
}
@Around("businessService()")
public void doAroundTask(){
```

```
...
}
```

你可以为任意一个建议定义你的切入点内联。下面是在建议之前定义内联切入点的一个示例：

```
@Before("execution(* com.xyz.myapp.service.*(..))")
public doBeforeTask(){
    ...
}
```

基于 AOP 的 @AspectJ 示例

为了解上面提到的关于基于 AOP 的 @AspectJ 的概念，让我们编写一个示例，可以实现几个建议。为了在我们的示例中使用几个建议，让我们使 Eclipse IDE 处于工作状态，然后按照如下步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 <code>src</code> 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。
3	在项目中添加 Spring AOP 指定的库文件 <i>aspectjrt.jar</i> ， <i>aspectjweaver.jar</i> 和 <i>aspectj.jar</i> 。
4	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>Logging</i> ， <i>Student</i> 和 <i>MainApp</i> 。
5	在 <code>src</code> 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
6	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

这里是 *Logging.java* 文件的内容。这实际上是 aspect 模块的一个示例，它定义了在各个点调用的方法。

```
package com.tutorialspoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Around;
@Aspect
public class Logging {
    /** Following is the definition for a pointcut to select
     * all the methods available. So advice will be called
     * for all the methods.
     */
}
```

```

@Pointcut("execution(* com.tutorialspoint.*.*(..))")
private void selectAll(){}
/**
 * This is the method which I would like to execute
 * before a selected method execution.
 */
@Before("selectAll()")
public void beforeAdvice(){
    System.out.println("Going to setup student profile.");
}
/**
 * This is the method which I would like to execute
 * after a selected method execution.
 */
@After("selectAll()")
public void afterAdvice(){
    System.out.println("Student profile has been setup.");
}
/**
 * This is the method which I would like to execute
 * when any method returns.
 */
@AfterReturning(pointcut = "selectAll()", returning="retVal")
public void afterReturningAdvice(Object retVal){
    System.out.println("Returning:" + retVal.toString() );
}
/**
 * This is the method which I would like to execute
 * if there is an exception raised by any method.
 */
@AfterThrowing(pointcut = "selectAll()", throwing = "ex")
public void AfterThrowingAdvice(IllegalArgumentException ex){
    System.out.println("There has been an exception: " + ex.toString());
}
}

```

下面是 Student.java 文件的内容：

```

package com.tutorialspoint;
public class Student {
    private Integer age;
    private String name;
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {

```

```

    System.out.println("Age : " + age );
    return age;
}
public void setName(String name) {
    this.name = name;
}
public String getName() {
    System.out.println("Name : " + name );
    return name;
}
public void printThrowException(){
    System.out.println("Exception raised");
    throw new IllegalArgumentException();
}
}

```

下面是 MainApp.java 文件的内容：

```

package com.tutorialspoint;
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");
        Student student = (Student) context.getBean("student");
        student.getName();
        student.getAge();
        student.printThrowException();
    }
}

```

下面是配置文件 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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

    <aop:aspectj-autoproxy/>

    <!-- Definition for student bean -->
    <bean id="student" class="com.tutorialspoint.Student">
        <property name="name" value="Zara" />
        <property name="age" value="11"/>
    </bean>

```

```
<!-- Definition for logging aspect -->  
<bean id="logging" class="com.tutorialspoint.Logging"/>  
  
</beans>
```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Going to setup student profile.  
Name : Zara  
Student profile has been setup.  
Returning:Zara  
Going to setup student profile.  
Age : 11  
Student profile has been setup.  
Returning:11  
Going to setup student profile.  
Exception raised  
Student profile has been setup.  
There has been an exception: java.lang.IllegalArgumentException  
.....  
other exception content
```



20

JDBC 框架概述



在使用普通的 JDBC 数据库时，就会很麻烦的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 JDBC 框架负责所有的低层细节，从开始打开连接，准备和执行 SQL 语句，处理异常，处理事务，到最后关闭连接。

所以当从数据库中获取数据时，你所做的是定义连接参数，指定要执行的 SQL 语句，每次迭代完成所需的工作。

Spring JDBC 提供几种方法和数据库中相应的不同的类与接口。我将给出使用 `JdbcTemplate` 类框架的经典和最受欢迎的方法。这是管理所有数据库通信和异常处理的中央框架类。

JdbcTemplate 类

`JdbcTemplate` 类执行 SQL 查询、更新语句和存储过程调用，执行迭代结果集和提取返回参数值。它也捕获 JDBC 异常并转换它们到 `org.springframework.dao` 包中定义的通用类、更多的信息、异常层次结构。

`JdbcTemplate` 类的实例是线程安全配置的。所以你可以配置 `JdbcTemplate` 的单个实例，然后将这个共享的引用安全地注入到多个 DAOs 中。

使用 `JdbcTemplate` 类时常见的做法是在你的 Spring 配置文件中配置数据源，然后共享数据源 bean 依赖注入到 DAO 类中，并在数据源的设值函数中创建了 `JdbcTemplate`。

配置数据源

我们在数据库 TEST 中创建一个数据库表 `Student`。假设你正在使用 MySQL 数据库，如果你使用其他数据库，那么你可以改变你的 DDL 和相应的 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到 `JdbcTemplate` 中，所以它可以配置本身来获得数据库访问。你可以在 XML 文件中配置数据源，其中一段代码如下所示：

```
<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 代表常用的数据库交互的数据访问对象。DAOs 提供一种方法来读取数据并将数据写入到数据库中，它们应该通过一个接口显示此功能，应用程序的其余部分将访问它们。

在 Spring 中，数据访问对象(DAO)支持很容易用统一的方法使用数据访问技术，如 JDBC、Hibernate、JPA 或者 JDO。

执行 SQL 语句

我们看看如何使用 SQL 和 jdbcTemplate 对象在数据库表中执行 CRUD(创建、读取、更新和删除)操作。

查询一个整数类型：

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL );
```

查询一个 long 类型：

```
String SQL = "select count(*) from Student";
long rowCount = jdbcTemplateObject.queryForLong( SQL );
```

一个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});
```

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;
    }
}

```

在表中插入一行：

```

String SQL = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );

```

更新表中的一行：

```

String SQL = "update Student set name = ? where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );

```

从表中删除一行：

```

String SQL = "delete Student where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{20} );

```

执行 DDL 语句

你可以使用 *jdbcTemplate* 中的 `execute(..)` 方法来执行任何 SQL 语句或 DDL 语句。下面是一个使用 CREATE 语句创建一个表的示例：

```

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 框架例子

基于上述概念，让我们看看一些重要的例子来帮助你理解在 Spring 中使用 JDBC 框架：

序号	例子 & 描述
1	Spring JDBC Example (jdbc-framework-overview/spring-jdbc-example.md) 这个例子将解释如何编写一个简单的基于 Spring 应用程序的 JDBC。
2	SQL Stored Procedure in Spring (jdbc-framework-overview/sql-stored-procedure-in-spring.md) 学习在使用 Spring 中的 JDBC 时如何调用 SQL 存储过程。

Spring JDBC 示例

想要理解带有 jdbc 模板类的 Spring JDBC 框架的相关概念，让我们编写一个简单的示例，来实现下述 Student 表的所有 CRUD 操作。

```
CREATE TABLE Student(  
  ID INT NOT NULL AUTO_INCREMENT,  
  NAME VARCHAR(20) NOT NULL,  
  AGE INT NOT NULL,  
  PRIMARY KEY (ID)  
);
```

在继续之前，让我们适当地使用 Eclipse IDE 并按照如下所示的步骤创建一个 Spring 应用程序：

步 骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并在创建的项目中的 <i>src</i> 文件夹下创建包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项添加必需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	在项目中添加 Spring JDBC 指定的最新的库 <i>mysql-connector-java.jar</i> ， <i>org.springframework.jdbc.jar</i> 和 <i>org.springframework.transaction.jar</i> 。如果这些库不存在，你可以下载它们。
4	创建 DAO 接口 <i>StudentDAO</i> 并列所有必需的方法。尽管这一步不是必需的而且你可以直接编写 <i>StudentJDBCTemplate</i> 类，但是作为一个好的实践，我们最好还是做这一步。
5	在 <i>com.tutorialspoint</i> 包下创建其他的必需的 Java 类 <i>Student</i> ， <i>StudentMapper</i> ， <i>StudentJDBCTemplate</i> 和 <i>MainApp</i> 。
6	确保你已经在 TEST 数据库中创建了 <i>Student</i> 表。并确保你的 MySQL 服务器运行正常，且你可以使用给出的用户名和密码读/写访问数据库。
7	在 <i>src</i> 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。
8	最后一步是创建所有的 Java 文件和 Bean 配置文件的内容并按照如下所示的方法运行应用程序。

以下是数据访问对象接口文件 *StudentDAO.java* 的内容：

```
package com.tutorialspoint;  
import java.util.List;  
import javax.sql.DataSource;  
public interface StudentDAO {  
  /**  
   * This is the method to be used to initialize  
   * database resources ie. connection.  
   */  
  public void setDataSource(DataSource ds);  
  /**  
   * This is the method to be used to create
```

```

    * a record in the Student table.
    */
    public void create(String name, Integer age);
    /**
     * This is the method to be used to list down
     * a record from the Student table corresponding
     * to a passed student id.
     */
    public Student getStudent(Integer id);
    /**
     * This is the method to be used to list down
     * all the records from the Student table.
     */
    public List<Student> listStudents();
    /**
     * This is the method to be used to delete
     * a record from the Student table corresponding
     * to a passed student id.
     */
    public void delete(Integer id);
    /**
     * This is the method to be used to update
     * a record into the Student table.
     */
    public void update(Integer id, Integer age);
}

```

下面是 Student.java 文件的内容：

```

package com.tutorialspoint;
public class Student {
    private Integer age;
    private String name;
    private Integer id;
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}

```

```

public void setId(Integer id) {
    this.id = id;
}
public Integer getId() {
    return id;
}
}

```

以下是 StudentMapper.java 文件的内容：

```

package com.tutorialspoint;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
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;
    }
}

```

下面是为定义的 DAO 接口 StudentDAO 的实现类文件 StudentJdbcTemplate.java：

```

package com.tutorialspoint;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
public class StudentJdbcTemplate implements StudentDAO {
    private DataSource dataSource;
    private JdbcTemplate jdbcTemplateObject;
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        this.jdbcTemplateObject = new JdbcTemplate(dataSource);
    }
    public void create(String name, Integer age) {
        String SQL = "insert into Student (name, age) values (?, ?)";
        jdbcTemplateObject.update( SQL, name, age);
        System.out.println("Created Record Name = " + name + " Age = " + age);
        return;
    }
    public Student getStudent(Integer id) {
        String SQL = "select * from Student where id = ?";
        Student student = jdbcTemplateObject.queryForObject(SQL,

```

```

        new Object[]{id}, new StudentMapper());
    return student;
}
public List<Student> listStudents() {
    String SQL = "select * from Student";
    List <Student> students = jdbcTemplateObject.query(SQL,
        new StudentMapper());
    return students;
}
public void delete(Integer id){
    String SQL = "delete from Student where id = ?";
    jdbcTemplateObject.update(SQL, id);
    System.out.println("Deleted Record with ID = " + id );
    return;
}
public void update(Integer id, Integer age){
    String SQL = "update Student set age = ? where id = ?";
    jdbcTemplateObject.update(SQL, age, id);
    System.out.println("Updated Record with ID = " + id );
    return;
}
}
}

```

以下是 MainApp.java 文件的内容：

```

package com.tutorialspoint;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tutorialspoint.StudentJDBCTemplate;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");
        StudentJDBCTemplate studentJDBCTemplate =
            (StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
        System.out.println("-----Records Creation-----");
        studentJDBCTemplate.create("Zara", 11);
        studentJDBCTemplate.create("Nuha", 2);
        studentJDBCTemplate.create("Ayan", 15);
        System.out.println("-----Listing Multiple Records-----");
        List<Student> students = studentJDBCTemplate.listStudents();
        for (Student record : students) {
            System.out.print("ID : " + record.getId() );
            System.out.print(", Name : " + record.getName() );
            System.out.println(", Age : " + record.getAge());
        }
    }
}

```



```

    }
    System.out.println("-----Updating Record with ID = 2 -----");
    studentJdbcTemplate.update(2, 20);
    System.out.println("-----Listing Record with ID = 2 -----");
    Student student = studentJdbcTemplate.getStudent(2);
    System.out.print("ID : " + student.getId() );
    System.out.print(", Name : " + student.getName() );
    System.out.println(", Age : " + student.getAge());
}
}

```

下述是配置文件 Beans.xml 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">

    <!-- Initialization for data source -->
    <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>

    <!-- Definition for studentJdbcTemplate bean -->
    <bean id="studentJdbcTemplate"
        class="com.tutorialspoint.StudentJdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>

```

当你完成创建源和 bean 配置文件后，运行应用程序。如果你的应用程序一切运行顺利的话，将会输出如下所示的消息：

```

-----Records Creation-----
Created Record Name = Zara Age = 11
Created Record Name = Nuha Age = 2
Created Record Name = Ayan Age = 15
-----Listing Multiple Records-----
ID : 1, Name : Zara, Age : 11
ID : 2, Name : Nuha, Age : 2
ID : 3, Name : Ayan, Age : 15
-----Updating Record with ID = 2 -----
Updated Record with ID = 2
-----Listing Record with ID = 2 -----
ID : 2, Name : Nuha, Age : 20

```

你可以尝试自己删除在我的例子中我没有用到的操作，但是现在你有一个基于 Spring JDBC 框架的工作应用程序，你可以根据你的项目需求来扩展这个框架，添加复杂的功能。还有其他方法来访问你使用 NamedParamete

`JdbcTemplate` 和 `SimpleJdbcTemplate` 类的数据库，所以如果你有兴趣学习这些类的话，那么你可以查看 Spring 框架的参考手册。

Spring 中 SQL 的存储过程

SimpleJdbcCall 类可以被用于调用一个包含 IN 和 OUT 参数的存储过程。你可以在处理任何一个 RDBMS 时使用这个方法，就像 Apache Derby，DB2，MySQL，Microsoft SQL Server，Oracle，和 Sybase。

为了了解这个方法，我们使用我们的 Student 表，它可以在 MySQL TEST 数据库中使用下面的 DDL 进行创建：

```
CREATE TABLE Student(
  ID INT NOT NULL AUTO_INCREMENT,
  NAME VARCHAR(20) NOT NULL,
  AGE INT NOT NULL,
  PRIMARY KEY (ID)
);
```

下一步，考虑接下来的 MySQL 存储过程，该过程使用 学生 Id 并且使用 OUT 参数返回相应的学生的姓名和年龄。所以让我们在你的 TEST 数据库中使用 MySQL 命令提示符创建这个存储过程：

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `TEST`.`getRecord` $$
CREATE PROCEDURE `TEST`.`getRecord` (
  IN in_id INTEGER,
  OUT out_name VARCHAR(20),
  OUT out_age INTEGER)
BEGIN
  SELECT name, age
  INTO out_name, out_age
  FROM Student where id = in_id;
END $$
DELIMITER ;
```

现在，让我们编写我们的 Spring JDBC 应用程序，它可以实现对我们的 Student 数据库表的创建和读取操作。让我们使 Eclipse IDE 处于工作状态，然后按照如下步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并且在所创建项目的 src 文件夹下创建一个名为 <i>com.tutorialspoint</i> 的包。
2	使用 <i>Add External JARs</i> 选项添加所需的 Spring 库文件，就如在 <i>Spring Hello World Example</i> 章节中解释的那样。

- | | |
|---|--|
| 3 | 在项目中添加 Spring JDBC 指定的最新的库文件 <code>mysql-connector-java.jar</code> , <code>org.springframework.jdbc.jar</code> 和 <code>org.springframework.transaction.jar</code> 。如果你还没有这些所需要的库文件, 你可以下载它们。 |
| 4 | 创建 DAO 接口 <code>StudentDAO</code> 并且列出所有需要的方法。即使他不是必需的, 你可以直接编写 <code>StudentJDBCTemplate</code> 类, 但是作为一个良好的实践, 让我们编写它。 |
| 5 | 在 <code>com.tutorialspoint</code> 包下创建其他所需要的 Java 类 <code>Student</code> , <code>StudentMapper</code> , <code>StudentJDBCTemplate</code> 和 <code>MainApp</code> 。 |
| 6 | 确保你已经在 TEST 数据库中创建了 <code>Student</code> 表。同样确保你的 MySQL 服务器是正常工作的, 并且保证你可以使用给定的用户名和密码对数据库有读取/写入的权限。 |
| 7 | 在 <code>src</code> 文件夹下创建 Beans 配置文件 <code>Beans.xml</code> 。 |
| 8 | 最后一步是创建所有 Java 文件和 Bean 配置文件的内容, 并且按如下解释的那样运行应用程序。 |

下面是数据访问对象接口文件 `StudentDAO.java` 的内容:

```
package com.tutorialspoint;
import java.util.List;
import javax.sql.DataSource;
public interface StudentDAO {
    /**
     * This is the method to be used to initialize
     * database resources ie. connection.
     */
    public void setDataSource(DataSource ds);
    /**
     * This is the method to be used to create
     * a record in the Student table.
     */
    public void create(String name, Integer age);
    /**
     * This is the method to be used to list down
     * a record from the Student table corresponding
     * to a passed student id.
     */
    public Student getStudent(Integer id);
    /**
     * This is the method to be used to list down
     * all the records from the Student table.
     */
    public List<Student> listStudents();
}
```

下面是 `Student.java` 文件的内容:

```
package com.tutorialspoint;
public class Student {
```

```

private Integer age;
private String name;
private Integer id;
public void setAge(Integer age) {
    this.age = age;
}
public Integer getAge() {
    return age;
}
public void setName(String name) {
    this.name = name;
}
public String getName() {
    return name;
}
public void setId(Integer id) {
    this.id = id;
}
public Integer getId() {
    return id;
}
}

```

下面是 `StudentMapper.java` 文件的内容：

```

package com.tutorialspoint;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
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;
    }
}

```

下面是实现类文件 `StudentJdbcTemplate.java`，定义了 DAO 接口 `StudentDAO`：

```

package com.tutorialspoint;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

```

```

import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
public class StudentJDBCTemplate implements StudentDAO {
    private DataSource dataSource;
    private SimpleJdbcCall jdbcCall;
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        this.jdbcCall = new SimpleJdbcCall(dataSource).
            withProcedureName("getRecord");
    }
    public void create(String name, Integer age) {
        JdbcTemplate jdbcTemplateObject = new JdbcTemplate(dataSource);
        String SQL = "insert into Student (name, age) values (?, ?)";
        jdbcTemplateObject.update( SQL, name, age);
        System.out.println("Created Record Name = " + name + " Age = " + age);
        return;
    }
    public Student getStudent(Integer id) {
        SqlParameterSource in = new MapSqlParameterSource().
            addValue("in_id", id);
        Map<String, Object> out = jdbcCall.execute(in);
        Student student = new Student();
        student.setId(id);
        student.setName((String) out.get("out_name"));
        student.setAge((Integer) out.get("out_age"));
        return student;
    }
    public List<Student> listStudents() {
        String SQL = "select * from Student";
        List<Student> students = jdbcTemplateObject.query(SQL,
            new StudentMapper());
        return students;
    }
}

```

关于上述项目的几句话：你编写的课调用执行的代码涉及创建一个包含 IN 参数的 *SqlParameterSource*。名称的匹配是很重要的，该名称可以使用在存储过程汇总声明的参数名称来提供输入值。*execute* 方法利用 IN 参数返回一个包含在存储过程中由名称指定的任何外部参数键的映射。现在让我们移动主应用程序文件 *MainApp.java*，如下所示：

```

package com.tutorialspoint;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tutorialspoint.StudentJDBCTemplate;

```

```

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");
        StudentJDBCTemplate studentJDBCTemplate =
            (StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
        System.out.println("-----Records Creation-----");
        studentJDBCTemplate.create("Zara", 11);
        studentJDBCTemplate.create("Nuha", 2);
        studentJDBCTemplate.create("Ayan", 15);
        System.out.println("-----Listing Multiple Records-----");
        List<Student> students = studentJDBCTemplate.listStudents();
        for (Student record : students) {
            System.out.print("ID : " + record.getId() );
            System.out.print(", Name : " + record.getName() );
            System.out.println(", Age : " + record.getAge());
        }
        System.out.println("-----Listing Record with ID = 2 -----");
        Student student = studentJDBCTemplate.getStudent(2);
        System.out.print("ID : " + student.getId() );
        System.out.print(", Name : " + student.getName() );
        System.out.println(", Age : " + student.getAge());
    }
}

```

下面是配置文件 **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">

    <!-- Initialization for data source -->
    <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>

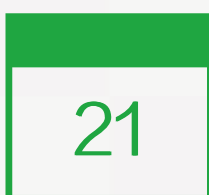
    <!-- Definition for studentJDBCTemplate bean -->
    <bean id="studentJDBCTemplate"
        class="com.tutorialspoint.StudentJDBCTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>

```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
-----Records Creation----- Created Record Name = Zara Age = 11 Created Record Name = Nuha
Age = 2 Created Record Name = Ayan Age = 15 -----Listing Multiple Records----- ID : 1, Name : Z
ara, Age : 11 ID : 2, Name : Nuha, Age : 2 ID : 3, Name : Ayan, Age : 15 ----Listing Record with ID = 2 ----
- ID : 2, Name : Nuha, Age : 2
```

事务管理



一个数据库事务是一个被视为单一的工作单元的操作序列。这些操作应该要么完整地执行，要么完全不执行。事务管理是一个重要组成部分，RDBMS 面向企业应用程序，以确保数据完整性和一致性。事务的概念可以描述为具有以下四个关键属性说成是 ACID：

- **原子性**：事务应该当作一个单独单元的操作，这意味着整个序列操作要么是成功，要么是失败的。
- **一致性**：这表示数据库的引用完整性的一致性，表中唯一的主键等。
- **隔离性**：可能同时处理很多有相同的数据集的事务，每个事务应该与其他事务隔离，以防止数据损坏。
- **持久性**：一个事务一旦完成全部操作后，这个事务的结果必须是永久性的，不能因系统故障而从数据库中删除。

一个真正的 RDBMS 数据库系统将为每个事务保证所有的四个属性。使用 SQL 发布到数据库中的事务的简单视图如下：

- 使用 *begin transaction* 命令开始事务。
- 使用 SQL 查询语句执行各种删除、更新或插入操作。
- 如果所有的操作都成功，则执行提交操作，否则回滚所有操作。

Spring 框架在不同的底层事务管理 APIs 的顶部提供了一个抽象层。Spring 的事务支持旨在通过添加事务能力到 POJOs 来提供给 EJB 事务一个选择方案。Spring 支持编程式和声明式事务管理。EJBs 需要一个应用程序服务器，但 Spring 事务管理可以在不需要应用程序服务器的情况下实现。

局部事物 vs. 全局事务

局部事务是特定于一个单一的事务资源，如一个 JDBC 连接，而全局事务可以跨多个事务资源事务，如在一个分布式系统中的事务。

局部事务管理在一个集中的计算环境中是有用的，该计算环境中应用程序组件和资源位于一个单位点，而事务管理只涉及到一个运行在一个单一机器中的本地数据管理器。局部事务更容易实现。

全局事务管理需要在分布式计算环境中，所有的资源都分布在多个系统中。在这种情况下事务管理需要同时在局部和全局范围内进行。分布式或全局事务跨多个系统执行，它的执行需要全局事务管理系统和所有相关系统的局部数据管理人员之间的协调。

编程式 vs. 声明式

Spring 支持两种类型的事务管理:

- [编程式事务管理：\(页 159\)](#)这意味着你在编程的帮助下有管理事务。这给了你极大的灵活性，但却很难维护。
- [声明式事务管理：\(页 165\)](#)这意味着你从业务代码中分离事务管理。你仅仅使用注释或 XML 配置来管理事务。

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

Spring 事务抽象

Spring 事务抽象的关键是由 `org.springframework.transaction.PlatformTransactionManager` 接口定义，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;
}
```

序号	方法 & 描述
----	---------

1	<div>TransactionStatus getTransaction(TransactionDefinition definition)</div> <div>根据指定的传播行为，该方法返回当前活动事务或创建一个新的事务。</div>
2	<div>void commit(TransactionStatus status)</div> <div>该方法提交给定的事务和关于它的状态。</div>
3	<div>void rollback(TransactionStatus status)</div>

该方法执行一个给定事务的回滚。

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();  
}
```

序号	方法 & 描述
1	int getPropagationBehavior() 该方法返回传播行为。Spring 提供了与 EJB CMT 类似的的所有的事务传播选项。
2	int getIsolationLevel() 该方法返回该事务独立于其他事务的工作的程度。
3	String getName() 该方法返回该事务的名称。
4	int getTimeout() 该方法返回以秒为单位的时间间隔，事务必须在该时间间隔内完成。
5	boolean isReadOnly() 该方法返回该事务是否是只读的。

下面是隔离级别的可能值：

序号	隔离 & 描述
1	TransactionDefinition.ISOLATION_DEFAULT

	这是默认的隔离级别。
2	TransactionDefinition.ISOLATION_READ_COMMITTED 表明能够阻止误读；可以发生不可重复读和虚读。
3	TransactionDefinition.ISOLATION_READ_UNCOMMITTED 表明可以发生误读、不可重复读和虚读。
4	TransactionDefinition.ISOLATION_REPEATABLE_READ 表明能够阻止误读和不可重复读；可以发生虚读。
5	TransactionDefinition.ISOLATION_SERIALIZABLE 表明能够阻止误读、不可重复读和虚读。

下面是传播类型的可能值:

序号	传播 & 描述
1	TransactionDefinition.PROPAGATION_MANDATORY 支持当前事务；如果不存在当前事务，则抛出一个异常。
2	TransactionDefinition.PROPAGATION_NESTED 如果存在当前事务，则在一个嵌套的事务中执行。
3	TransactionDefinition.PROPAGATION_NEVER 不支持当前事务；如果存在当前事务，则抛出一个异常。
4	TransactionDefinition.PROPAGATION_NOT_SUPPORTED

	不支持当前事务；而总是执行非事务性。
5	TransactionDefinition.PROPROPAGATION_REQUIRED 支持当前事务；如果不存在事务，则创建一个新的事务。
6	TransactionDefinition.PROPROPAGATION_REQUIRES_NEW 创建一个新事务，如果存在一个事务，则把当前事务挂起。
7	TransactionDefinition.PROPROPAGATION_SUPPORTS 支持当前事务；如果不存在，则执行非事务性。
8	TransactionDefinition.TIMEOUT_DEFAULT 使用默认超时的底层事务系统，或者如果不支持超时则没有。

TransactionStatus 接口为事务代码提供了一个简单的方法来控制事务的执行和查询事务状态。

```
public interface TransactionStatus extends SavepointManager {
    boolean isNewTransaction();
    boolean hasSavepoint();
    void setRollbackOnly();
    boolean isRollbackOnly();
    boolean isCompleted();
}
```

序号	方法 & 描述
1	boolean hasSavepoint() 该方法返回该事务内部是否有一个保存点，也就是说，基于一个保存点已经创建了嵌套事务。
2	boolean isCompleted() 该方法返回该事务是否完成，也就是说，它是否已经提交或回滚。

3

`boolean isNewTransaction()`

在当前事务时新的情况下，该方法返回 true。

4

`boolean isRollbackOnly()`

该方法返回该事务是否已标记为 rollback-only。

5

`void setRollbackOnly()`

该方法设置该事务为 rollback-only 标记。

Spring 程式事务管理

程式事务管理方法允许你在对你的源代码编程的帮助下管理事务。这给了你极大地灵活性，但是它很难维护。

在我们开始之前，至少要有两个数据库表，在事务的帮助下我们可以执行多种 CRUD 操作。以 **Student** 表为例，用下述 DDL 可以在 MySQL TEST 数据库中创建该表：

```
CREATE TABLE Student(
  ID INT NOT NULL AUTO_INCREMENT,
  NAME VARCHAR(20) NOT NULL,
  AGE INT NOT NULL,
  PRIMARY KEY (ID)
);
```

第二个表是 **Marks**，用来存储基于年份的学生的标记。这里 **SID** 是 **Student** 表的外键。

```
CREATE TABLE Marks(
  SID INT NOT NULL,
  MARKS INT NOT NULL,
  YEAR INT NOT NULL
);
```

让我们直接使用 *PlatformTransactionManager* 来实现程式方法从而实现事务。要开始一个新事务，你需要有一个带有适当的 *transaction* 属性的 *TransactionDefinition* 的实例。这个例子中，我们使用默认的 *transaction* 属性简单的创建了 *DefaultTransactionDefinition* 的一个实例。

当 *TransactionDefinition* 创建后，你可以通过调用 *getTransaction()* 方法来开始你的事务，该方法会返回 *TransactionStatus* 的一个实例。*TransactionStatus* 对象帮助追踪当前的事务状态，并且最终，如果一切运行顺利，你可以使用 *PlatformTransactionManager* 的 *commit()* 方法来提交这个事务，否则的话，你可以使用 *rollback()* 方法来回滚整个操作。

现在让我们编写我们的 Spring JDBC 应用程序，它能够在 **Student** 和 **Mark** 表中实现简单的操作。让我们适当的使用 Eclipse IDE，并按照如下所示的步骤来创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并在创建的项目中的 <i>src</i> 文件夹下创建包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项添加必需的 Spring 库，解释见 <i>Spring Hello World Example chapter</i> 。
3	在项目中添加 Spring JDBC 指定的最新的库 <i>mysql-connector-java.jar</i> ， <i>org.springframework.jdbc.jar</i> 和 <i>org.springframework.transaction.jar</i> 。如果你还没有这些库，你可以下载它们。

- | | |
|---|---|
| 4 | 创建 DAO 接口 <i>StudentDAO</i> 并列所有需要的方法。尽管它不是必需的并且你可以直接编写 <i>StudentJDBCTemplate</i> 类，但是作为一个好的实践，我们还是做吧。 |
| 5 | 在 <i>com.tutorialspoint</i> 包下创建其他必需的 Java 类 <i>StudentMarks</i> , <i>StudentMarksMapper</i> , <i>StudentJDBCTemplate</i> 和 <i>MainApp</i> 。如果需要的话，你可以创建其他的 POJO 类。 |
| 6 | 确保你已经在 TEST 数据库中创建了 Student 和 Marks 表。还要确保你的 MySQL 服务器运行正常并且你使用给出的用户名和密码可以读/写访问数据库。 |
| 7 | 在 <i>src</i> 文件夹下创建 Beans 配置文件 <i>Beans.xml</i> 。 |
| 8 | 最后一步是创建所有 Java 文件和 Bean 配置文件的内容并按照如下所示的方法运行应用程序。 |

下面是数据访问对象接口文件 *StudentDAO.java* 的内容：

```
package com.tutorialspoint;
import java.util.List;
import javax.sql.DataSource;
public interface StudentDAO {
    /**
     * This is the method to be used to initialize
     * database resources ie. connection.
     */
    public void setDataSource(DataSource ds);
    /**
     * This is the method to be used to create
     * a record in the Student and Marks tables.
     */
    public void create(String name, Integer age, Integer marks, Integer year);
    /**
     * This is the method to be used to list down
     * all the records from the Student and Marks tables.
     */
    public List<StudentMarks> listStudents();
}
```

下面是 *StudentMarks.java* 文件的内容：

```
package com.tutorialspoint;
public class StudentMarks {
    private Integer age;
    private String name;
    private Integer id;
    private Integer marks;
    private Integer year;
    private Integer sid;
    public void setAge(Integer age) {
        this.age = age;
    }
}
```

```

public Integer getAge() {
    return age;
}
public void setName(String name) {
    this.name = name;
}
public String getName() {
    return name;
}
public void setId(Integer id) {
    this.id = id;
}
public Integer getId() {
    return id;
}
public void setMarks(Integer marks) {
    this.marks = marks;
}
public Integer getMarks() {
    return marks;
}
public void setYear(Integer year) {
    this.year = year;
}
public Integer getYear() {
    return year;
}
public void setSid(Integer sid) {
    this.sid = sid;
}
public Integer getSid() {
    return sid;
}
}

```

以下是 StudentMarksMapper.java 文件的内容：

```

package com.tutorialspoint;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
public class StudentMarksMapper implements RowMapper<StudentMarks> {
    public StudentMarks mapRow(ResultSet rs, int rowNum) throws SQLException {
        StudentMarks studentMarks = new StudentMarks();
        studentMarks.setId(rs.getInt("id"));
        studentMarks.setName(rs.getString("name"));
    }
}

```

```

        studentMarks.setAge(rs.getInt("age"));
        studentMarks.setSid(rs.getInt("sid"));
        studentMarks.setMarks(rs.getInt("marks"));
        studentMarks.setYear(rs.getInt("year"));
        return studentMarks;
    }
}

```

下面是定义的 DAO 接口 StudentDAO 实现类文件 StudentJdbcTemplate.java:

```

package com.tutorialspoint;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;
public class StudentJdbcTemplate implements StudentDAO {
    private DataSource dataSource;
    private JdbcTemplate jdbcTemplateObject;
    private PlatformTransactionManager transactionManager;
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        this.jdbcTemplateObject = new JdbcTemplate(dataSource);
    }
    public void setTransactionManager(
        PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }
    public void create(String name, Integer age, Integer marks, Integer year){
        TransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);
        try {
            String SQL1 = "insert into Student (name, age) values (?, ?)";
            jdbcTemplateObject.update( SQL1, name, age);
            // Get the latest student id to be used in Marks table
            String SQL2 = "select max(id) from Student";
            int sid = jdbcTemplateObject.queryForInt( SQL2 );
            String SQL3 = "insert into Marks(sid, marks, year) " +
                "values (?, ?, ?)";
            jdbcTemplateObject.update( SQL3, sid, marks, year);
            System.out.println("Created Name = " + name + ", Age = " + age);
            transactionManager.commit(status);
        } catch (DataAccessException e) {

```

```

        System.out.println("Error in creating record, rolling back");
        transactionManager.rollback(status);
        throw e;
    }
    return;
}
public List<StudentMarks> listStudents() {
    String SQL = "select * from Student, Marks where Student.id=Marks.sid";
    List <StudentMarks> studentMarks = jdbcTemplateObject.query(SQL,
        new StudentMarksMapper());
    return studentMarks;
}
}

```

现在让我们改变主应用程序文件 `MainApp.java`，如下所示：

```

package com.tutorialspoint;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tutorialspoint.StudentJDBCTemplate;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");
        StudentJDBCTemplate studentJDBCTemplate =
            (StudentJDBCTemplate)context.getBean("studentJDBCTemplate");
        System.out.println("-----Records creation-----" );
        studentJDBCTemplate.create("Zara", 11, 99, 2010);
        studentJDBCTemplate.create("Nuha", 20, 97, 2010);
        studentJDBCTemplate.create("Ayan", 25, 100, 2011);
        System.out.println("-----Listing all the records-----" );
        List<StudentMarks> studentMarks = studentJDBCTemplate.listStudents();
        for (StudentMarks record : studentMarks) {
            System.out.print("ID : " + record.getId() );
            System.out.print(", Name : " + record.getName() );
            System.out.print(", Marks : " + record.getMarks());
            System.out.print(", Year : " + record.getYear());
            System.out.println(", Age : " + record.getAge());
        }
    }
}

```

下面是配置文件 `Beans.xml` 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">

<!-- Initialization for data source -->
<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>

<!-- Initialization for TransactionManager -->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- Definition for studentJdbcTemplate bean -->
<bean id="studentJdbcTemplate"
    class="com.tutorialspoint.StudentJdbcTemplate">
    <property name="dataSource" ref="dataSource" />
    <property name="transactionManager" ref="transactionManager" />
</bean>

</beans>

```

当你完成了创建源和 bean 配置文件后，让我们运行应用程序。如果你的应用程序运行顺利的话，那么将会输出如下所示的消息：

```

-----Records creation-----
Created Name = Zara, Age = 11
Created Name = Nuha, Age = 20
Created Name = Ayan, Age = 25
-----Listing all the records-----
ID : 1, Name : Zara, Marks : 99, Year : 2010, Age : 11
ID : 2, Name : Nuha, Marks : 97, Year : 2010, Age : 20
ID : 3, Name : Ayan, Marks : 100, Year : 2011, Age : 25

```

Spring 声明式事务管理

声明式事务管理方法允许你在配置的帮助下而不是源代码硬编程来管理事务。这意味着你可以将事务管理从事务代码中隔离出来。你可以只使用注释或基于配置的 XML 来管理事务。bean 配置会指定事务型方法。这是与声明式事务相关的步骤：

- 我们使用 `@Transactional` 标签，它创建一个事务处理的建议，同时，我们定义一个匹配所有方法的切入点，我们希望这些方法是事务型的并且会引用事务型的建议。
- 如果在事务型配置中包含了一个方法的名称，那么创建的建议在调用方法之前就会在事务中开始进行。
- 目标方法会在 `try / catch` 块中执行。
- 如果方法正常结束，AOP 建议会成功的提交事务，否则它执行回滚操作。

让我们看看上述步骤是如何实现的。但是在我们开始之前，至少有两个数据库表是至关重要的，在事务的帮助下，我们可以实现各种 CRUD 操作。以 `Student` 表为例，该表是使用下述 DDL 在 MySQL TEST 数据库中创建的。

```
CREATE TABLE Student(
  ID INT NOT NULL AUTO_INCREMENT,
  NAME VARCHAR(20) NOT NULL,
  AGE INT NOT NULL,
  PRIMARY KEY (ID)
);
```

第二个表是 `Marks`，我们用来存储基于年份的学生标记。在这里，`SID` 是 `Student` 表的外键。

```
CREATE TABLE Marks(
  SID INT NOT NULL,
  MARKS INT NOT NULL,
  YEAR INT NOT NULL
);
```

现在让我们编写 Spring JDBC 应用程序来在 `Student` 和 `Marks` 表中实现简单的操作。让我们适当的使用 Eclipse IDE，并按照如下所示的步骤来创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 <i>SpringExample</i> 的项目，并在创建的项目中的 <code>src</code> 文件夹下创建包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项添加必需的 Spring 库，解释见 <i>Spring Hello World Example chapter</i> 。

- 3 在项目中添加其它必需的库 `mysql-connector-java.jar`, `org.springframework.jdbc.jar` 和 `org.springframework.transaction.jar`。如果你还没有这些库，你可以下载它们。
- 4 创建 DAO 接口 `StudentDAO` 并列出所有需要的方法。尽管它不是必需的并且你可以直接编写 `StudentJDBCTemplate` 类，但是作为一个好的实践，我们还是做吧。
- 5 在 `com.tutorialspoint` 包下创建其他必需的 Java 类 `StudentMarks`, `StudentMarksMapper`, `StudentJDBCTemplate` 和 `MainApp`。如果需要的话，你可以创建其他的 POJO 类。
- 6 确保你已经在 TEST 数据库中创建了 `Student` 和 `Marks` 表。还要确保你的 MySQL 服务器运行正常并且你使用给出的用户名和密码可以读/写访问数据库。
- 7 在 `src` 文件夹下创建 Beans 配置文件 `Beans.xml`。
- 8 最后一步是创建所有 Java 文件和 Bean 配置文件的内容并按照如下所示的方法运行应用程序。

下面是数据访问对象接口文件 `StudentDAO.java` 的内容：

```
package com.tutorialspoint;
import java.util.List;
import javax.sql.DataSource;
public interface StudentDAO {
    /**
     * This is the method to be used to initialize
     * database resources ie. connection.
     */
    public void setDataSource(DataSource ds);
    /**
     * This is the method to be used to create
     * a record in the Student and Marks tables.
     */
    public void create(String name, Integer age, Integer marks, Integer year);
    /**
     * This is the method to be used to list down
     * all the records from the Student and Marks tables.
     */
    public List<StudentMarks> listStudents();
}
```

以下是 `StudentMarks.java` 文件的内容：

```
package com.tutorialspoint;
public class StudentMarks {
    private Integer age;
    private String name;
    private Integer id;
    private Integer marks;
    private Integer year;
    private Integer sid;
    public void setAge(Integer age) {
```

```

        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public Integer getId() {
        return id;
    }
    public void setMarks(Integer marks) {
        this.marks = marks;
    }
    public Integer getMarks() {
        return marks;
    }
    public void setYear(Integer year) {
        this.year = year;
    }
    public Integer getYear() {
        return year;
    }
    public void setSid(Integer sid) {
        this.sid = sid;
    }
    public Integer getSid() {
        return sid;
    }
}

```

下面是 StudentMarksMapper.java 文件的内容：

```

package com.tutorialspoint;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
public class StudentMarksMapper implements RowMapper<StudentMarks> {
    public StudentMarks mapRow(ResultSet rs, int rowNum) throws SQLException {
        StudentMarks studentMarks = new StudentMarks();
    }
}

```



```

        studentMarks.setId(rs.getInt("id"));
        studentMarks.setName(rs.getString("name"));
        studentMarks.setAge(rs.getInt("age"));
        studentMarks.setSid(rs.getInt("sid"));
        studentMarks.setMarks(rs.getInt("marks"));
        studentMarks.setYear(rs.getInt("year"));
        return studentMarks;
    }
}

```

下面是定义的 DAO 接口 StudentDAO 实现类文件 StudentJDBCTemplate.java:

```

package com.tutorialspoint;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
public class StudentJDBCTemplate implements StudentDAO{
    private JdbcTemplate jdbcTemplateObject;
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplateObject = new JdbcTemplate(dataSource);
    }
    public void create(String name, Integer age, Integer marks, Integer year){
        try {
            String SQL1 = "insert into Student (name, age) values (?, ?)";
            jdbcTemplateObject.update( SQL1, name, age);
            // Get the latest student id to be used in Marks table
            String SQL2 = "select max(id) from Student";
            int sid = jdbcTemplateObject.queryForInt( SQL2 );
            String SQL3 = "insert into Marks(sid, marks, year) " +
                "values (?, ?, ?)";
            jdbcTemplateObject.update( SQL3, sid, marks, year);
            System.out.println("Created Name = " + name + ", Age = " + age);
            // to simulate the exception.
            throw new RuntimeException("simulate Error condition" );
        } catch (DataAccessException e) {
            System.out.println("Error in creating record, rolling back");
            throw e;
        }
    }
    public List<StudentMarks> listStudents() {
        String SQL = "select * from Student, Marks where Student.id=Marks.sid";
        List <StudentMarks> studentMarks=jdbcTemplateObject.query(SQL,
            new StudentMarksMapper());
        return studentMarks;
    }
}

```

```
}
}
```

现在让我们改变主应用程序文件 `MainApp.java`，如下所示：

```
package com.tutorialspoint;
import java.util.List;
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");
        StudentDAO studentJDBCTemplate =
            (StudentDAO)context.getBean("studentJDBCTemplate");
        System.out.println("-----Records creation-----");
        studentJDBCTemplate.create("Zara", 11, 99, 2010);
        studentJDBCTemplate.create("Nuha", 20, 97, 2010);
        studentJDBCTemplate.create("Ayan", 25, 100, 2011);
        System.out.println("-----Listing all the records-----");
        List<StudentMarks> studentMarks = studentJDBCTemplate.listStudents();
        for (StudentMarks record : studentMarks) {
            System.out.print("ID : " + record.getId() );
            System.out.print(", Name : " + record.getName() );
            System.out.print(", Marks : " + record.getMarks());
            System.out.print(", Year : " + record.getYear());
            System.out.println(", Age : " + record.getAge());
        }
    }
}
```

以下是配置文件 `Beans.xml` 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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

    <!-- Initialization for data source -->
    <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"/>
    </bean>
</beans>
```

```

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

<tx:advice id="txAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="create"/>
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="createOperation"
    expression="execution(* com.tutorialspoint.StudentJDBCTemplate.create(..)"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="createOperation"/>
</aop:config>

<!-- Initialization for TransactionManager -->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource" />
</bean>

<!-- Definition for studentJDBCTemplate bean -->
<bean id="studentJDBCTemplate"
class="com.tutorialspoint.StudentJDBCTemplate">
  <property name="dataSource" ref="dataSource" />
</bean>

</beans>

```

当你完成了创建源和 bean 配置文件后，让我们运行应用程序。如果你的应用程序运行顺利的话，那么会输出如下所示的异常。在这种情况下，事务会回滚并且在数据库表中不会创建任何记录。

```

-----Records creation-----
Created Name = Zara, Age = 11
Exception in thread "main" java.lang.RuntimeException: simulate Error condition

```

在删除异常后，你可以尝试上述示例，在这种情况下，会提交事务并且你可以在数据库中看见一条记录。



22

MVC 框架教程

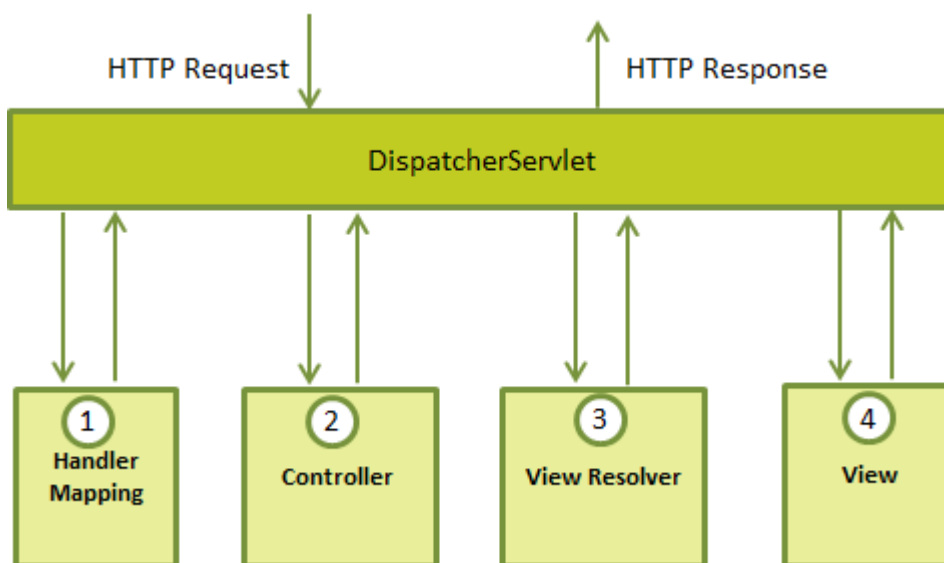


Spring web MVC 框架提供了模型-视图-控制的体系结构和可以用来开发灵活、松散耦合的 web 应用程序的组件。MVC 模式导致了应用程序的不同方面(输入逻辑、业务逻辑和 UI 逻辑)的分离，同时提供了在这些元素之间的松散耦合。

- 模型封装了应用程序数据，并且通常它们由 POJO 组成。
- 视图主要用于呈现模型数据，并且通常它生成客户端的浏览器可以解释的 HTML 输出。
- 控制器主要用于处理用户请求，并且构建合适的模型并将其传递到视图呈现。

DispatcherServlet

Spring Web 模型-视图-控制（MVC）框架是围绕 *DispatcherServlet* 设计的，*DispatcherServlet* 用来处理所有的 HTTP 请求和响应。Spring Web MVC *DispatcherServlet* 的请求处理的工作流程如下图所示：



下面是对应于 *DispatcherServlet* 传入 HTTP 请求的事件序列：

- 收到一个 HTTP 请求后，*DispatcherServlet* 根据 *HandlerMapping* 来选择并且调用适当的控制器。
- 控制器接受请求，并基于使用的 GET 或 POST 方法来调用适当的 service 方法。Service 方法将设置基于定义的业务逻辑的模型数据，并返回视图名称到 *DispatcherServlet* 中。
- *DispatcherServlet* 会从 *ViewResolver* 获取帮助，为请求检索定义视图。
- 一旦确定视图，*DispatcherServlet* 将把模型数据传递给视图，最后呈现在浏览器中。

上面所提到的所有组件，即 `HandlerMapping`、`Controller` 和 `ViewResolver` 是 `WebApplicationContext` 的一部分，而 `WebApplicationContext` 是带有一些对 web 应用程序必要的额外特性的 `ApplicationContext` 的扩展。

需求的配置

你需要映射你想让 `DispatcherServlet` 处理的请求，通过使用在 `web.xml` 文件中的一个 URL 映射。下面是一个显示声明和映射 `HelloWeb DispatcherServlet` 的示例：

```
<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` 文件将被保留在你的应用程序的 `WebContent/WEB-INF` 目录下。好的，在初始化 `HelloWeb DispatcherServlet` 时，该框架将尝试加载位于该应用程序的 `WebContent/WEB-INF` 目录中文件名为 `[servlet-name]-servlet.xml` 的应用程序内容。在这种情况下，我们的文件将是 `HelloWeb-servlet.xml`。

接下来，`<servlet-mapping>` 标签表明哪些 URLs 将被 `DispatcherServlet` 处理。这里所有以 `.jsp` 结束的 HTTP 请求将由 `HelloWeb DispatcherServlet` 处理。

如果你不想使用默认文件名 `[servlet-name]-servlet.xml` 和默认位置 `WebContent/WEB-INF`，你可以通过在 `web.xml` 文件中添加 `servlet` 监听器 `ContextLoaderListener` 自定义该文件的名称和位置，如下所示：

```
<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，检查 `HelloWeb-servlet.xml` 文件的请求配置，该文件位于 web 应用程序的 `WebContent/WEB-INF` 目录下：

```

<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.tutorialspoint" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>

```

以下是关于 `HelloWeb-servlet.xml` 文件的一些要点：

- `[servlet-name]-servlet.xml` 文件将用于创建 bean 定义，重新定义在全局范围内具有相同名称的任何已定义的 bean。
- `<context:component-scan...>` 标签将用于激活 Spring MVC 注释扫描功能，该功能允许使用注释，如 `@Controller` 和 `@RequestMapping` 等等。
- `InternalResourceViewResolver` 将使用定义的规则来解决视图名称。按照上述定义的规则，一个名称为 `hello` 的逻辑视图将发送给位于 `/WEB-INF/jsp/hello.jsp` 中实现的视图。

下一节将向你展示如何创建实际的组件，例如控制器，模式和视图。

定义控制器

DispatcherServlet 发送请求到控制器中执行特定的功能。`@Controller` 注释表明一个特定类是一个控制器的作用。`@RequestMapping` 注释用于映射 URL 到整个类或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";
    }
}
```

`@Controller` 注释定义该类作为一个 Spring MVC 控制器。在这里，第一次使用的 `@RequestMapping` 表明在该控制器中处理的所有方法都是相对于 `/hello` 路径的。下一个注释 `@RequestMapping(method = RequestMethod.GET)` 用于声明 `printHello()` 方法作为控制器的默认 service 方法来处理 HTTP GET 请求。你可以在相同的 URL 中定义其他方法来处理任何 POST 请求。

你可以用另一种形式来编写上面的控制器，你可以在 `@RequestMapping` 中添加额外的属性，如下所示：

```
@Controller
public class HelloController{
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

值属性表明 URL 映射到哪个处理方法，方法属性定义了 service 方法来处理 HTTP GET 请求。关于上面定义的控制器的，这里有以下几个要注意的要点：

- 你将在一个 service 方法中定义需要的业务逻辑。你可以根据每次需求在这个方法中调用其他方法。
- 基于定义的业务逻辑，你将在这个方法中创建一个模型。你可以设置不同的模型属性，这些属性将被视图访问并显示最终的结果。这个示例创建了一个带有属性 “message” 的模型。
- 一个定义的 service 方法可以返回一个包含视图名称的字符串用于呈现该模型。这个示例返回 “hello” 作为逻辑视图的名称。

创建 JSP 视图

对于不同的表示技术，Spring MVC 支持许多类型的视图。这些包括 JSP、HTML、PDF、Excel 工作表、XML、Velocity 模板、XSLT、JSON、Atom 和 RSS 提要、JasperReports 等等。但我们最常使用利用 JSTL 编写的 JSP 模板。所以让我们在 /WEB-INF/hello/hello.jsp 中编写一个简单的 **hello** 视图：

```
<html>
<head>
<title>Hello Spring MVC</title>
</head>
<body>
<h2>${message}</h2>
</body>
</html>
```

其中，`${message}` 是我们在控制器内部设置的属性。你可以在你的视图中有多个属性显示。

Spring Web MVC 框架例子

基于上述概念，让我们看看一些重要的例子来帮助你建立 Spring Web 应用程序：

序号	例子 & 描述
1	Spring MVC Hello World Example (mvc-framework/spring-mvc-hello-world-example.md) 这个例子将解释如何编写一个简单的 Spring Web Hello World 应用程序。
2	Spring MVC Form Handling Example (mvc-framework/spring-mvc-form-handling-example.md) 这个例子将解释如何编写一个 Spring Web 应用程序，它使用 HTML 表单提交数据到控制器，并且显示处理结果。
3	Spring Page Redirection Example (mvc-framework/spring-page-redirection-example.md) 学习在 Spring MVC 框架中如何使用页面重定向功能。
4	Spring Static Pages Example (mvc-framework/spring-static-pages-example.md) 学习在 Spring MVC 框架中如何访问静态页面和动态页面。

-
- 5 [Spring Exception Handling Example \(mvc-framework/spring-exception-handling-example.md\)](#)

学习在 Spring MVC 框架中如何处理异常。

Spring MVC Hello World 例子

下面的例子说明了如何使用 Spring MVC 框架来编写一个简单的基于 web 的 Hello World 应用程序。为了开始使用它，让我们在恰当的位置使用 Eclipse IDE，然后按照下面的步骤使用 Spring 的 Web 框架来开发一个动态 Web 应用程序：

步骤	描述
1	创建一个名称为 <i>HelloWeb</i> 的动态 Web 项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	将上面提到的 Spring 和其他库拖拽到文件夹 <i>WebContent/WEB-INF/lib</i> 中。
3	在 <i>com.tutorialspoint</i> 包下创建一个 Java 类 <i>HelloController</i> 。
4	在 <i>WebContent/WEB-INF</i> 文件夹下创建 Spring 的配置文件 <i>Web.xml</i> 和 <i>HelloWeb-servlet.xml</i> 。
5	在 <i>WebContent/WEB-INF</i> 文件夹下创建名称为 <i>jsp</i> 的子文件夹。在这个子文件夹下创建一个视图文件 <i>hello.jsp</i> 。
6	最后一步是创建所有的源代码和配置文件的内容，并导出该应用程序，正如下面解释的一样。

这里是 *HelloController.java* 文件的内容：

```
package com.tutorialspoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.ui.ModelMap;
@Controller
@RequestMapping("/hello")
public class HelloController{
    @RequestMapping(method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

下面是 Spring Web 配置文件 *web.xml* 的内容

```
<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>/</url-pattern>
</servlet-mapping>

</web-app>

```

下面是另一个 Spring Web 配置文件 `HelloWeb-servlet.xml` 的内容

```

<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.tutorialspoint" />

    <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 Spring 视图文件 `hello.jsp` 的内容

```

<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
    <h2>${message}</h2>
</body>
</html>

```

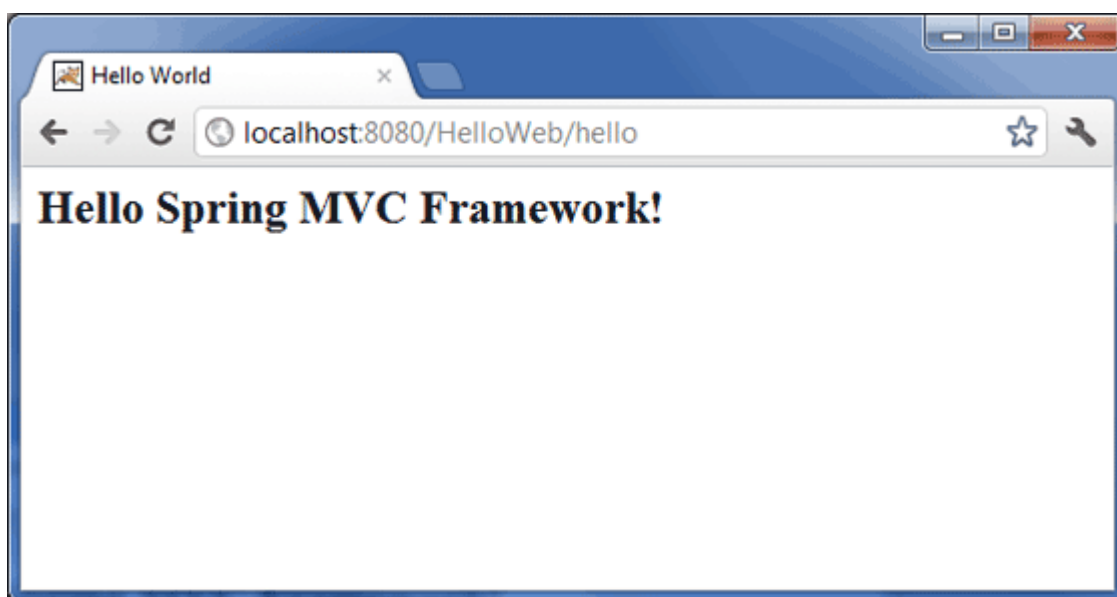
最后，下面是包含在你的 web 应用程序中的 Spring 和其他库。你仅仅需要将这些文件拖拽到 `WebContent/WEB-INF/lib` 文件夹中。

- commons-logging-x.y.z.jar
- org.springframework.asm-x.y.z.jar
- org.springframework.beans-x.y.z.jar
- org.springframework.context-x.y.z.jar

- org.springframework.core-x.y.z.jar
- org.springframework.expression-x.y.z.jar
- org.springframework.web.servlet-x.y.z.jar
- org.springframework.web-x.y.z.jar
- spring-web.jar

一旦你完成了创建源代码和配置文件后，导出你的应用程序。右键单击你的应用程序，并且使用 **Export > WAR File** 选项，并且在 Tomcat 的 *webapps* 文件夹中保存你的 **HelloWeb.war** 文件。

现在启动你的 Tomcat 服务器，并且确保你能够使用标准的浏览器访问 *webapps* 文件夹中的其他 web 页面。现在尝试访问该 URL <http://localhost:8080/HelloWeb/hello>。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



你应该注意，在给定的 URL 中，**HelloWeb** 是这个应用程序的名称，并且 **hello** 是我们在控制器中使用 `@RequestMapping("/hello")` 提到的虚拟子文件夹。当使用 `@RequestMapping("/")` 映射你的 URL 时，你可以使用直接 root，在这种情况下，你可以使用短 URL <http://localhost:8080/HelloWeb/> 访问相同的页面，但是建议在不同的文件夹下有不同的功能。

Spring MVC 表单处理例子

下面的例子说明了如何编写一个简单的基于 web 的应用程序，它利用了使用 Spring 的 Web MVC 框架的 HTML 表单。为了开始使用它，让我们在恰当的位置使用 Eclipse IDE，然后按照下面的步骤使用 Spring 的 Web 框架来开发一个动态的基于表单的 Web 应用程序：

步骤	描述
1	创建一个名称为 <i>HelloWeb</i> 的动态 Web 项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	将上面提到的 Spring 和其他库拖拽到文件夹 <i>WebContent/WEB-INF/lib</i> 中。
3	在 <i>com.tutorialspoint</i> 包下创建一个 Java 类 <i>Student</i> 和 <i>StudentController</i> 。
4	在 <i>WebContent/WEB-INF</i> 文件夹下创建 Spring 的配置文件 <i>Web.xml</i> 和 <i>HelloWeb-servlet.xml</i> 。
5	在 <i>WebContent/WEB-INF</i> 文件夹下创建名称为 <i>jsp</i> 的子文件夹。在这个子文件夹下创建视图文件 <i>student.jsp</i> 和 <i>result.jsp</i> 。
6	最后一步是创建所有的源代码和配置文件的内容，并导出该应用程序，正如下面解释的一样。

这里是 *Student.java* 文件的内容：

```
package com.tutorialspoint;

public class Student {
    private Integer age;
    private String name;
    private Integer id;
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public Integer getId() {
        return id;
    }
}
```

```

}
}

```

下面是 `StudentController.java` 文件的内容：

```

package com.tutorialspoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.ui.ModelMap;
@Controller
public class StudentController {
    @RequestMapping(value = "/student", method = RequestMethod.GET)
    public ModelAndView student() {
        return new ModelAndView("student", "command", new Student());
    }
    @RequestMapping(value = "/addStudent", method = RequestMethod.POST)
    public String addStudent(@ModelAttribute("SpringWeb")Student student,
        ModelMap model) {
        model.addAttribute("name", student.getName());
        model.addAttribute("age", student.getAge());
        model.addAttribute("id", student.getId());
        return "result";
    }
}

```

在这里，第一个 service 方法 `student()`，我们已经在名称为 “command” 的 `ModelAndView` 对象中传递一个空的 `Student` 对象，因为 spring 框架需要一个名称的 “command” 的对象，如果你在 JSP 文件中使用 `<form:form>` 标签。所以，当 `student()` 方法被调用时，它返回 `student.jsp` 视图。

第二个 service 方法 `addStudent()` 将调用 `HelloWeb/addStudent` URL 中的 POST 方法。你将根据提交的信息准备好你的模型对象。最后一个 “result” 视图会从 service 方法中返回，它将导致呈现 `result.jsp`。

下面是 Spring Web 配置文件 `web.xml` 的内容

```

<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 Form Handling</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>/</url-pattern>
</servlet-mapping>

</web-app>

```

下面是另一个 Spring Web 配置文件 `HelloWeb-servlet.xml` 的内容

```

<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.tutorialspoint" />

    <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 Spring 视图文件 `student.jsp` 的内容

```

<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
    <title>Spring MVC Form Handling</title>
</head>
<body>

<h2>Student Information</h2>
<form:form method="POST" action="/HelloWeb/addStudent">
    <table>
        <tr>
            <td><form:label path="name">Name</form:label></td>
            <td><form:input path="name" /></td>
        </tr>
        <tr>
            <td><form:label path="age">Age</form:label></td>
            <td><form:input path="age" /></td>
        </tr>
        <tr>
            <td><form:label path="id">id</form:label></td>
            <td><form:input path="id" /></td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="Submit"/>
            </td>
        </tr>
    </table>
</form:form>

```



```
</body>
</html>
```

下面是 Spring 视图文件 `result.jsp` 的内容

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring MVC Form Handling</title>
</head>
<body>

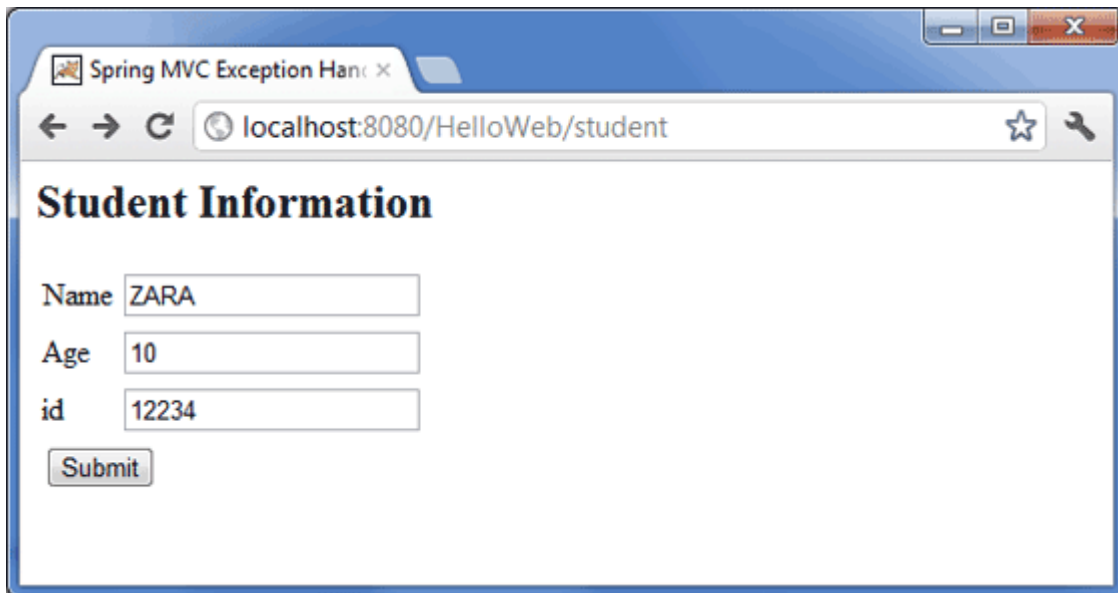
<h2>Submitted Student Information</h2>
<table>
  <tr>
    <td>Name</td>
    <td>${name}</td>
  </tr>
  <tr>
    <td>Age</td>
    <td>${age}</td>
  </tr>
  <tr>
    <td>ID</td>
    <td>${id}</td>
  </tr>
</table>
</body>
</html>
```

最后，下面是包含在你的 web 应用程序中的 Spring 和其他库的列表。你仅仅需要将这些文件拖拽到 `WebContent/WEB-INF/lib` 文件夹中。

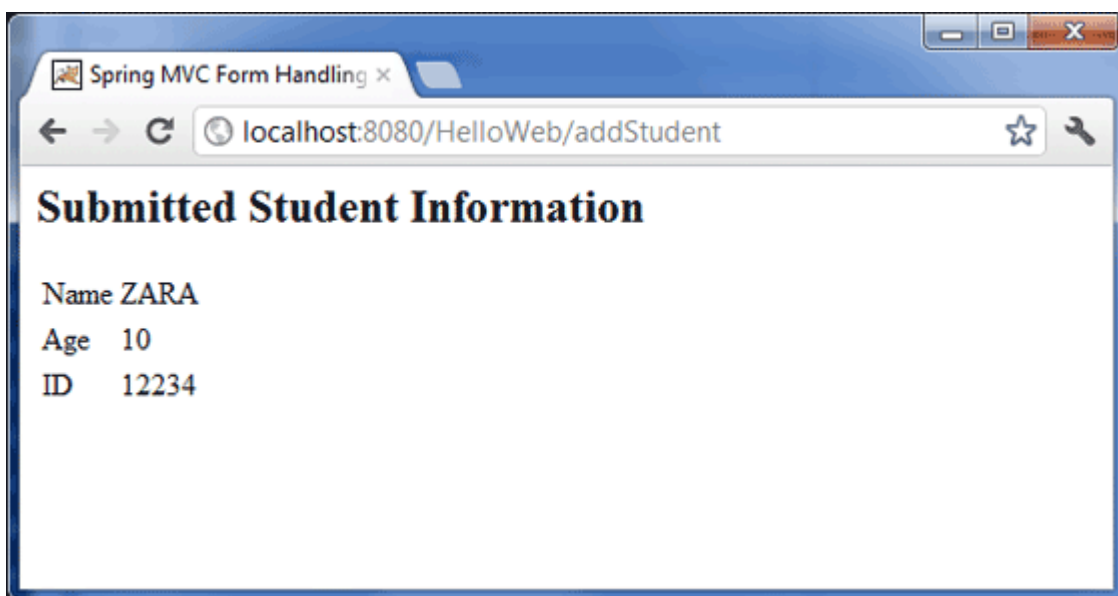
- commons-logging-x.y.z.jar
- org.springframework.asm-x.y.z.jar
- org.springframework.beans-x.y.z.jar
- org.springframework.context-x.y.z.jar
- org.springframework.core-x.y.z.jar
- org.springframework.expression-x.y.z.jar
- org.springframework.web.servlet-x.y.z.jar
- org.springframework.web-x.y.z.jar
- spring-web.jar

一旦你完成了创建源代码和配置文件后，导出你的应用程序。右键单击你的应用程序，并且使用 `Export > WAR File` 选项，并且在 Tomcat 的 `webapps` 文件夹中保存你的 `HelloWeb.war` 文件。

现在启动你的 Tomcat 服务器，并确保你能够使用标准的浏览器访问 webapps 文件夹中的其他 web 页面。现在尝试访问该 URL `http://localhost:8080/SpringWeb/student`。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



在提交必需的信息之后，单击提交按钮来提交这个表单。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



Spring 页面重定向例子

下面的例子说明了如何编写一个简单的基于 web 的应用程序，它利用重定向来传送一个 http 请求到另一个页面中。为了开始使用它，让我们在恰当的位置使用 Eclipse IDE，然后按照下面的步骤使用 Spring 的 Web 框架来开发一个动态的基于表单的 Web 应用程序：

步骤	描述
1	创建一个名称为 <i>HelloWeb</i> 的动态 Web 项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	将上面提到的 Spring 和其他库拖拽到文件夹 <i>WebContent/WEB-INF/lib</i> 中。
3	在 <i>com.tutorialspoint</i> 包下创建一个 Java 类 <i>WebController</i> 。
4	在 <i>WebContent/WEB-INF</i> 文件夹下创建 Spring 的配置文件 <i>Web.xml</i> 和 <i>HelloWeb-servlet.xml</i> 。
5	在 <i>WebContent/WEB-INF</i> 文件夹下创建名称为 <i>jsp</i> 的子文件夹。在这个子文件夹下创建视图文件 <i>index.jsp</i> 和 <i>final.jsp</i> 。
6	最后一步是创建所有的源代码和配置文件的内容，并导出该应用程序，正如下面解释的一样。

这里是 *WebController.java* 文件的内容：

```
package com.tutorialspoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
@Controller
public class WebController {
    @RequestMapping(value = "/index", method = RequestMethod.GET)
    public String index() {
        return "index";
    }
    @RequestMapping(value = "/redirect", method = RequestMethod.GET)
    public String redirect() {
        return "redirect:finalPage";
    }
    @RequestMapping(value = "/finalPage", method = RequestMethod.GET)
    public String finalPage() {
        return "final";
    }
}
```

下面是 Spring Web 配置文件 *web.xml* 的内容

```
<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 Page Redirection</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>/</url-pattern>
</servlet-mapping>

</web-app>

```

下面是另一个 Spring Web 配置文件 `HelloWeb-servlet.xml` 的内容

```

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

  <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>

```

下面是 Spring 视图文件 `index.jsp` 文件的内容。这将是一个登陆页面，这个页面将发送一个请求来访问重定向 `service` 方法，该方法将把这个请求重定向到另一个 `service` 方法中，最后将显示 `final.jsp` 页面。

```

<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring Page Redirection</title>
</head>
<body>
<h2>Spring Page Redirection</h2>
<p>Click below button to redirect the result to new page</p>
<form:form method="GET" action="/HelloWeb/redirect">
<table>
  <tr>
    <td>
      <input type="submit" value="Redirect Page"/>
    </td>
  </tr>
</table>
</form:form>

```

```
</body>
</html>
```

下面是 Spring 视图文件 `final.jsp` 的内容。这是最终的重定向页面。

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring Page Redirection</title>
</head>
<body>

<h2>Redirected Page</h2>

</body>
</html>
```

最后，下面是包含在你的 web 应用程序中的 Spring 和其他库的列表。你仅仅需要将这些文件拖拽到 `WebContent/WEB-INF/lib` 文件夹中。

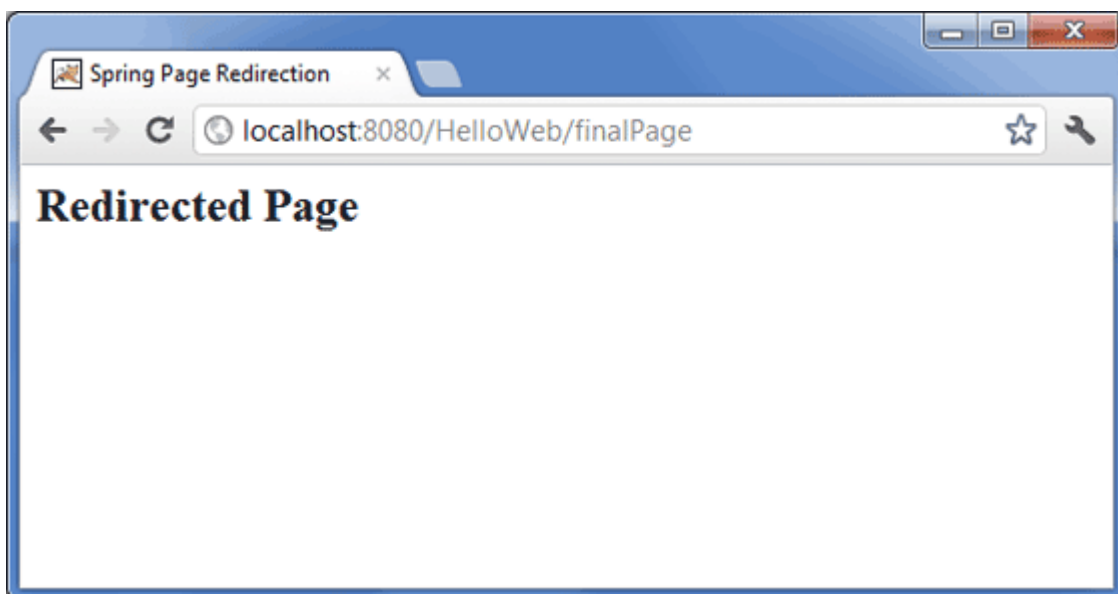
- commons-logging-x.y.z.jar
- org.springframework.asm-x.y.z.jar
- org.springframework.beans-x.y.z.jar
- org.springframework.context-x.y.z.jar
- org.springframework.core-x.y.z.jar
- org.springframework.expression-x.y.z.jar
- org.springframework.web.servlet-x.y.z.jar
- org.springframework.web-x.y.z.jar
- spring-web.jar

一旦你完成了创建源代码和配置文件后，导出你的应用程序。右键单击你的应用程序，并且使用 `Export > WAR File` 选项，并且在 Tomcat 的 `webapps` 文件夹中保存你的 `HelloWeb.war` 文件。

现在启动你的 Tomcat 服务器，并且确保你能够使用标准的浏览器访问 `webapps` 文件夹中的其他 web 页面。现在尝试访问该 URL `http://localhost:8080/HelloWeb/index`。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



现在单击“Redirect Page”按钮来提交表单，并且得到最终的重定向页面。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



Spring 静态页面例子

下面的例子说明了如何使用 Spring MVC 框架来编写一个简单的基于 web 的应用程序，它可以在 `<mvc:resources>` 标签的帮助下访问静态页面和动态页面。为了开始使用它，让我们在恰当的位置使用 Eclipse IDE，然后按照下面的步骤使用 Spring 的 Web 框架来开发一个动态的基于表单的 Web 应用程序：

步骤	描述
1	创建一个名称为 <i>HelloWeb</i> 的动态 Web 项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	将上面提到的 Spring 和其他库拖拽到文件夹 <i>WebContent/WEB-INF/lib</i> 中。
3	在 <i>com.tutorialspoint</i> 包下创建一个 Java 类 <i>WebController</i> 。
4	在 <i>WebContent/WEB-INF</i> 文件夹下创建 Spring 的配置文件 <i>Web.xml</i> 和 <i>HelloWeb-servlet.xml</i> 。
5	在 <i>WebContent/WEB-INF</i> 文件夹下创建名称为 <i>jsp</i> 的子文件夹。在这个子文件夹下创建一个视图文件 <i>index.jsp</i> 。
6	在 <i>WebContent/WEB-INF</i> 文件夹下创建名称为 <i>pages</i> 的子文件夹。在这个子文件夹下创建一个静态文件 <i>final.htm</i>
7	最后一步是创建所有的源代码和配置文件的内容，并导出该应用程序，正如下面解释的一样。

这里是 *WebController.java* 文件的内容：

```
package com.tutorialspoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
@Controller
public class WebController {
    @RequestMapping(value = "/index", method = RequestMethod.GET)
    public String index() {
        return "index";
    }
    @RequestMapping(value = "/staticPage", method = RequestMethod.GET)
    public String redirect() {
        return "redirect:/pages/final.htm";
    }
}
```

下面是 Spring Web 配置文件 *web.xml* 的内容

```
<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 Page Redirection</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>/</url-pattern>
</servlet-mapping>

</web-app>

```

下面是另一个 Spring Web 配置文件 `HelloWeb-servlet.xml` 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"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-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.tutorialspoint" />

  <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
  </bean>

  <mvc:resources mapping="/pages/**" location="/WEB-INF/pages/" />
  <mvc:annotation-driven/>

</beans>

```

在这里，`<mvc:resources.../>` 标签被用来映射静态页面。`mapping` 属性必须是一个指定一个 http 请求的 URL 模式的 Ant 模式。`location` 属性必须指定一个或者多个具有包含图片，样式表，JavaScript 和其他静态内容的静态页面的资源目录位置。多个资源位置可以使用逗号分隔这些值的列表来被指定。

下面是 Spring 视图文件 `WEB-INF/jsp/index.jsp` 的内容。这将是一个登陆页面，这个页面将发送一个请求来访问 `staticPage` 的 `service` 方法，它将重定向这个请求到 `WEB-INF/pages` 文件夹中的一个可用的静态页面。

```

<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring Landing Page</title>
</head>
<body>
<h2>Spring Landing Pag</h2>

```



```
<p>Click below button to get a simple HTML page</p>
<form:form method="GET" action="/HelloWeb/staticPage">
<table>
  <tr>
    <td>
      <input type="submit" value="Get HTML Page"/>
    </td>
  </tr>
</table>
</form:form>
</body>
</html>
```

下面是 Spring 视图文件 WEB-INF/pages/final.htm 的内容。

```
<html>
<head>
  <title>Spring Static Page</title>
</head>
<body>

<h2>A simple HTML page</h2>

</body>
</html>
```

最后，下面是包含在你的 web 应用程序中的 Spring 和其他库的列表。你仅仅需要将这些文件拖拽到 WebContent/WEB-INF/lib 文件夹中。

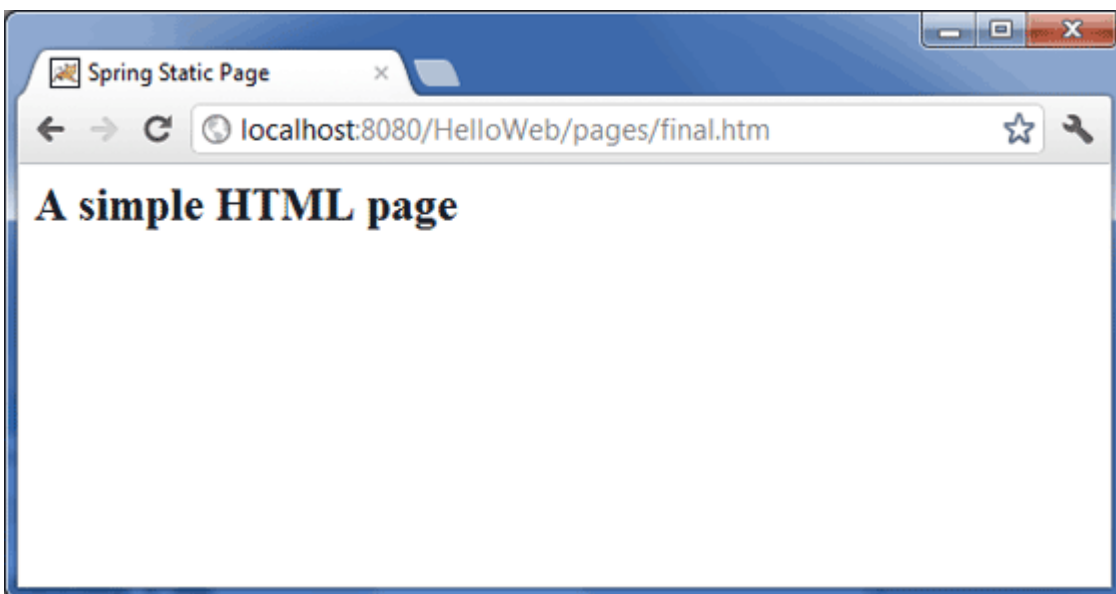
- commons-logging-x.y.z.jar
- org.springframework.asm-x.y.z.jar
- org.springframework.beans-x.y.z.jar
- org.springframework.context-x.y.z.jar
- org.springframework.core-x.y.z.jar
- org.springframework.expression-x.y.z.jar
- org.springframework.web.servlet-x.y.z.jar
- org.springframework.web-x.y.z.jar
- spring-web.jar

一旦你完成了创建源代码和配置文件后，导出你的应用程序。右键单击你的应用程序，并且使用 Export > WAR File 选项，并且在 Tomcat 的 webapps 文件夹中保存你的 HelloWeb.war 文件。

现在启动你的 Tomcat 服务器，并确保你能够使用标准的浏览器访问 webapps 文件夹中的其他 web 页面。现在尝试访问该 URL `http://localhost:8080/HelloWeb/index`。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



单击“Get HTML Page”按钮来访问 staticPage 中的 service 方法中提到的一个静态页面。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



Spring 异常处理例子

下面的例子说明了如何使用 Spring MVC 框架来编写一个简单的基于 web 的应用程序，它可以处理它的内置控制器产生的一个或多个异常。为了开始使用它，让我们在恰当的位置使用 Eclipse IDE，然后按照下面的步骤使用 Spring 的 Web 框架来开发一个动态的基于表单的 Web 应用程序：

步骤	描述
1	创建一个名称为 <i>HelloWeb</i> 的动态 Web 项目，并且在已创建的项目的 <i>src</i> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	将上面提到的 Spring 和其他库拖拽到文件夹 <i>WebContent/WEB-INF/lib</i> 中。
3	在 <i>com.tutorialspoint</i> 包下创建一个 Java 类 <i>Student</i> ， <i>StudentController</i> 和 <i>SpringException</i> 。
4	在 <i>WebContent/WEB-INF</i> 文件夹下创建 Spring 的配置文件 <i>Web.xml</i> 和 <i>HelloWeb-servlet.xml</i> 。
5	在 <i>WebContent/WEB-INF</i> 文件夹下创建名称为 <i>jsp</i> 的子文件夹。在这个子文件夹下创建视图文件 <i>student.jsp</i> ， <i>result.jsp</i> ， <i>error.jsp</i> 和 <i>ExceptionPage.jsp</i> 。
6	最后一步是创建所有的源代码和配置文件的内容，并导出该应用程序，正如下面解释的一样。

这里是 *Student.java* 文件的内容：

```
package com.tutorialspoint;

public class Student {
    private Integer age;
    private String name;
    private Integer id;
    public void setAge(Integer age) {
        this.age = age;
    }
    public Integer getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public Integer getId() {
        return id;
    }
}
```

```

}
}

```

下面是 `SpringException.java` 文件的内容：

```

package com.tutorialspoint;

public class SpringException extends RuntimeException{
    private String exceptionMsg;
    public SpringException(String exceptionMsg) {
        this.exceptionMsg = exceptionMsg;
    }
    public String getExceptionMsg(){
        return this.exceptionMsg;
    }
    public void setExceptionMsg(String exceptionMsg) {
        this.exceptionMsg = exceptionMsg;
    }
}

```

下面是 `StudentController.java` 文件的内容。这里，你需要使用 `@ExceptionHandler` 注解一个 service 方法，你可以指定要处理的一个或多个异常。如果你要指定一个以上的异常，那么你可以使用逗号分隔这些值。

```

package com.tutorialspoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.ui.ModelMap;
@Controller
public class StudentController {
    @RequestMapping(value = "/student", method = RequestMethod.GET)
    public ModelAndView student() {
        return new ModelAndView("student", "command", new Student());
    }
    @RequestMapping(value = "/addStudent", method = RequestMethod.POST)
    @ExceptionHandler({SpringException.class})
    public String addStudent( @ModelAttribute("HelloWeb")Student student,
        ModelMap model) {
        if(student.getName().length() < 5 ){
            throw new SpringException("Given name is too short");
        }else{
            model.addAttribute("name", student.getName());
        }
        if( student.getAge() < 10 ){

```

```

        throw new SpringException("Given age is too low");
    }else{
        model.addAttribute("age", student.getAge());
    }
    model.addAttribute("id", student.getId());
    return "result";
}
}

```

下面是 Spring Web 配置文件 **web.xml** 的内容

```

<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 Exception Handling</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>/</url-pattern>
    </servlet-mapping>

</web-app>

```

下面是另一个 Spring Web 配置文件 **HelloWeb-servlet.xml** 的内容

```

<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.tutorialspoint" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <bean class="org.springframework.web.servlet.handler.
        SimpleMappingExceptionResolver">
        <property name="exceptionMappings">
            <props>
                <prop key="com.tutorialspoint.SpringException">
                    ExceptionPage
                </prop>
            </props>
        </property>
    </bean>

```

```

    </props>
  </property>
  <property name="defaultErrorView" value="error"/>
</bean>

</beans>

```

在这里，你指定 *ExceptionHandler* 作为一个异常视图，以便 *SpringExceptionHandler* 发生，如果有任何其他类型的异常发生，那么一个通用的视图 *error* 会发生。

下面是 Spring 视图文件 *student.jsp* 的内容：

```

<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring MVC Exception Handling</title>
</head>
<body>

<h2>Student Information</h2>
<form:form method="POST" action="/HelloWeb/addStudent">
  <table>
    <tr>
      <td><form:label path="name">Name</form:label></td>
      <td><form:input path="name" /></td>
    </tr>
    <tr>
      <td><form:label path="age">Age</form:label></td>
      <td><form:input path="age" /></td>
    </tr>
    <tr>
      <td><form:label path="id">id</form:label></td>
      <td><form:input path="id" /></td>
    </tr>
    <tr>
      <td colspan="2">
        <input type="submit" value="Submit"/>
      </td>
    </tr>
  </table>
</form:form>
</body>
</html>

```

下面是 Spring 视图文件 *error.jsp* 的内容：

```

<html>
<head>
  <title>Spring Error Page</title>
</head>
<body>

<p>An error occurred, please contact webmaster.</p>

</body>
</html>;

```

下面是 Spring 视图文件 *ExceptionHandler.jsp* 的内容。在这里，你将通过 `${exception}` 访问异常实例。

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring MVC Exception Handling</title>
</head>
<body>

<h2>Spring MVC Exception Handling</h2>

<h3>${exception.exceptionMsg}</h3>

</body>
</html>
```

下面是 Spring 视图文件 **result.jsp** 的内容：

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>
<head>
  <title>Spring MVC Form Handling</title>
</head>
<body>

<h2>Submitted Student Information</h2>
  <table>
    <tr>
      <td>Name</td>
      <td>${name}</td>
    </tr>
    <tr>
      <td>Age</td>
      <td>${age}</td>
    </tr>
    <tr>
      <td>ID</td>
      <td>${id}</td>
    </tr>
  </table>
</body>
</html>
```

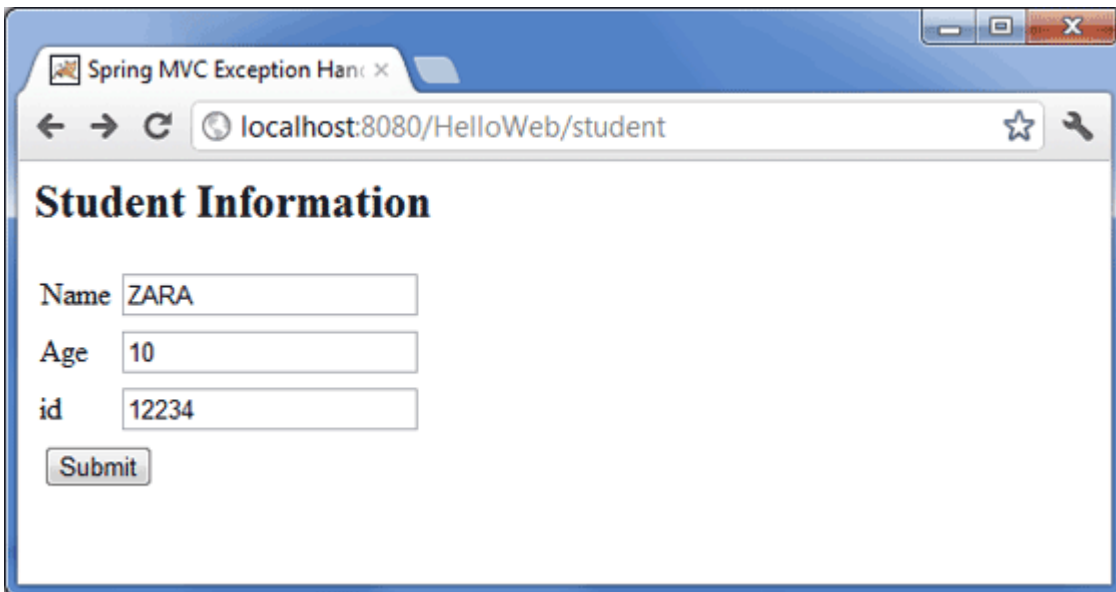
最后，下面是包含在你的 web 应用程序中的 Spring 和其他库的列表。你仅仅需要将这些文件拖拽到 **WebContent/WEB-INF/lib** 文件夹中。

- commons-logging-x.y.z.jar
- org.springframework.asm-x.y.z.jar
- org.springframework.beans-x.y.z.jar
- org.springframework.context-x.y.z.jar
- org.springframework.core-x.y.z.jar
- org.springframework.expression-x.y.z.jar

- org.springframework.web.servlet-x.y.z.jar
- org.springframework.web-x.y.z.jar
- spring-web.jar

一旦你完成了创建源代码和配置文件后，导出你的应用程序。右键单击你的应用程序，并且使用 **Export > WAR File** 选项，并且在 Tomcat 的 *webapps* 文件夹中保存你的 **HelloWeb.war** 文件。

现在启动你的 Tomcat 服务器，并且确保你能够使用标准的浏览器访问 *webapps* 文件夹中的其他 web 页面。现在尝试访问该 URL <http://localhost:8080/SpringWeb/student>。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：



The screenshot shows a web browser window with the title "Spring MVC Exception Han...". The address bar displays "localhost:8080/HelloWeb/student". The main content area is titled "Student Information" and contains a form with three input fields: "Name" (containing "ZARA"), "Age" (containing "10"), and "id" (containing "12234"). Below these fields is a "Submit" button.

输入如上图所示的值，然后单击提交按钮。如果你的 Spring Web 应用程序一切都正常，你应该看到下面的结果：





23

使用 Log4J 记录日志



在 Spring 应用程序中使用 Log4J 的功能是很容易的。下面的例子将带你通过简单的步骤解释 Log4J 和 Spring 之间的简单集成。

假设你已经在你的机器上安装了 Log4J，如果你还没有 Log4J，你可以从 <http://logging.apache.org/> 中下载，并且仅仅在任何文件夹中提取压缩文件。在我们的项目中，我们将只使用 `log4j-x.y.z.jar`。

接下来，我们让 Eclipse IDE 在恰当的位置工作，遵循以下步骤，使用 Spring Web 框架开发一个基于 Web 应用程序的动态表单：

步骤	描述
1	创建一个名称为 <i>SpringExample</i> 的项目，并且在创建项目的 <code>src</code> 文件夹中创建一个包 <i>com.tutorialspoint</i> 。
2	使用 <i>Add External JARs</i> 选项，添加所需的 Spring 库，解释见 <i>Spring Hello World Example</i> 章节。
3	使用 <i>Add External JARs</i> 选项，同样在你的项目中添加 log4j 库 <i>log4j-x.y.z.jar</i> 。
4	在 <i>com.tutorialspoint</i> 包下创建 Java 类 <i>HelloWorld</i> 和 <i>MainApp</i> 。
5	在 <code>src</code> 文件中创建 Bean 配置文件 <i>Beans.xml</i> 。
6	在 <code>src</code> 文件中创建 log4J 配置文件 <i>log4j.properties</i> 。
7	最后一步是创建的所有 Java 文件和 Bean 配置文件的内容，并运行应用程序，解释如下所示。

这个是 `HelloWorld.java` 文件的内容：

```
package com.tutorialspoint;
public class HelloWorld {
    private String message;
    public void setMessage(String message){
        this.message = message;
    }
    public void getMessage(){
        System.out.println("Your Message : " + message);
    }
}
```

下面的是第二个文件 `MainApp.java` 的内容：

```
package com.tutorialspoint;
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 HelloWorld Obj");
    HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
    obj.getMessage();
    log.info("Exiting the program");
}
}

```

使用与我们已经生成信息消息类似的方法，你可以生成调试和错误消息。现在让我们看看 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.tutorialspoint.HelloWorld">
    <property name="message" value="Hello World!"/>
  </bean>

</beans>

```

下面是 log4j.properties 的内容，它定义了使用 Log4J 生成日志信息所需的标准规则：

```

# 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

```

一旦你完成了创建源和 bean 的配置文件后，我们就可以运行该应用程序。如果你的应用程序一切都正常，在 Eclipse 控制台将输出以下信息：

```
Your Message : Hello World!
```

同时如果你检查你的 C:\ 驱动，那么你应该发现含有各种日志消息的日志文件 `log.out`，其中一些如下所示：

```
<!-- initialization log messages -->

Going to create HelloWorld Obj
Returning cached instance of singleton bean 'helloWorld'
Exiting the program
```

Jakarta Commons Logging (JCL) API

或者，你可以使用 Jakarta Commons Logging(JCL) API 在你的 Spring 应用程序中生成日志。JCL 可以从 <http://jakarta.apache.org/commons/logging/> 下载。我们在技术上需要这个包的唯一文件是 `commons-logging-x.y.z.jar` 文件，需要使用与上面的例子中你使用 `log4j-x.y.z.jar` 类似的方法把 `commons-logging-x.y.z.jar` 放在你的类路径中。

为了使用日志功能，你需要一个 `org.apache.commons.logging.Log` 对象，然后你可以根据你的需要调用任何一个下面的方法：

- `fatal(Object message)`
- `error(Object message)`
- `warn(Object message)`
- `info(Object message)`
- `debug(Object message)`
- `trace(Object message)`

下面是使用 JCL API 对 `MainApp.java` 的替换：

```
package com.tutorialspoint;
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 HelloWorld Obj");
    HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
    obj.getMessage();
    log.info("Exiting the program");
}
}
```

你应该确保在编译和运行该程序之前在你的项目中已经引入了 *commons-logging-x.y.z.jar* 文件。

现在保持在上面的例子中剩下的配置和内容不变，如果你编译并运行你的应用程序，你就会得到与使用 Log4J API 后获得的结果类似的结果。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/spring/>