论 spring 的零配置与 XML 配置①

张文宇[1], 许明健[1], 薛 昱[2]

[1](西安邮电大学 经济与管理学院, 西安 710061)
[2](北京信息控制研究所, 北京 100089)

摘 要: 从多方面比较了 Spring 框架的 XML 配置和零配置方式的好处与不足, 总结在实际开发中应如何灵活使用两种配置方式, 以提高项目开发的效率. 首先分别介绍了 Spring 框架的 XML 配置和零配置, 其次举例说明了如何通过 XML 配置和零配置实现 Spring 的依赖注入功能, 然后从 Spring 框架的骨骼架构和设计理念方面分析了 XML 配置相对零配置的优势. 实践结果表明, 在实际应用的开发期间应采用零配置方式, 而在开发后期, 项目功能完成时, 应将零配置转换为 XML 配置, 禁用零配置, 从而显著提高项目开发的效率.

关键词: Spring; XML; 零配置; 依赖注入; 控制反转件

Introduction to Spring Zero Configurationand XML Configurationand

ZHANG Wen-Yu[1], XU Ming-Jian[1], XUE Yu[2]

[1](College of Economic and Management, Xian University of Posts and Telecommunications, Xian 710061, China)
[2](Beijing Institute of Information Control, Beijing 100089, China)

Abstract: The benefits and shortcomings of XML configuration and zero configuration of the Spring Framework is compared from several aspects. How to use two configurations flexible in the actual development is summarized to improve the efficiency of project development. Firstly, XML configuration and zero configuration of the Spring Framework is introduced. Secondly, how to implement dependency injection of the Spring Framework is illustrated through XML configuration and zero configuration. Then, the relative advantages of XML configuration to zero-configuration is analyzed from bone architecture and design aspects of the Spring framework. Practice results show that the zero configuration should be used during the development of practical application. And late in the development, when the function of the project is completed, zero configuration should be converted to XML configuration, disable zero configuration. Thus, the efficiency of project development will be improved significantly.

Key words: Spring; XML; zero-configuration; dependency injection; inversion of control

在以前, 许多人认为 Java 是跨平台的语言, 而 XML 是跨平台的数据交换格式, 所以 Java 和 XML 应该是最好的组合. 在这种趋势下, 以前的 Java 框架不约而同地选择了 XML 作为配置文件. 所以早期的 Spring 大多也是采用 XML 文件来配置管理 Bean, 但这种方式需要大量精力去维护管理 XML 配置文件. 时至今日, 受 Rails 框架的启发, 几乎所有主流 Java 框架都支持“零配置”特性了, Spring 也开始支持使用 Annotation 来配置管理 Bean 了. 下面举例说明了如何通过 XML 配置和零配置实现依赖注入.

1 XML配置实现依赖注入

在介绍零配置之前, 先来看看以前是如何配置 Bean 并完成 Bean 之间依赖关系的创建[1].

步骤一: 定义 3 个类, 它们分别是 House 类、Pet 类和 Person 类, 这 3 个类需要在 Spring 容器中配置为

① 基金项目:陕西省自然科学基金(2012GQ8050)
收稿时间:2014-06-17;收到修改稿时间:2014-09-09

Bean:

House 类仅有一个属性:

代码 1: House.java

```
package com.xiyou.test;
public class House {
    private String HouseNo =”20140001”;
    //省略 get/setter 方法
    @Override
    public String toString() {
        return "HouseNo:" + HouseNo;
    }
}
```

Pet 类拥有两个属性:

代码 2: Pet.java

```
package com.xiyou.test;
public class Pet {
    private String name;
    private int age;
    // 省略 get/setter 方法
    @Override
    public String toString() {
        return "Name:" + name + "," + "Age:" + age;
    }
```

Person 类拥有 House 和 Pet 类型的两个属性:

代码 3: Person.java

```
package com.xiyou.test;
public class Person {
    private Pet pet;
    private House house;
    // 省略 get/setter 方法
    @Override
    public String toString() {
        return "Pet:" + pet + "\n" + "House:" + house;
    }
}
```

步骤二: 我们在 Spring 容器中将 House 类和 Pet 类声明为 Bean, 并注入到 Person Bean 中: 下面是使用传统 XML 配置完成这个工作的配置文件 beans.xml:

代码 4: beans.xml 将上述的三个类都配置成 Bean 实例

```
<?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-3.1.xsd">
    <bean id="person" class="com.xiyou.test.Person">
    <property name="Pet" ref="pet"/>
    <property name="House" ref="house" />
    </bean>
    <bean id="house" class="com.xiyou.test.House">
    <property name="HouseNo" value="20140001"/>
    </bean>
    <bean id="pet" class="com.xiyou.test.Pet"
scope="singleton">
    <property name="Name" value="哈哈"/>
    <property name="Age" value="1"/>
    </bean>
    </beans>
```

步骤三: 当我们运行下面的一段测试代码时, 控制台将会准确的输出有关 Person 的详细信息:

代码 5. 测试类: BeanTest.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class BeanTest {
    public static void main(String[] args) {
    //创建 Spring 容器
    ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
    Person person = (Person) ctx.getBean("person");
    System.out.println(person);
    }
}
```

这表明 Spring 已经正确地完成一个 Bean 实例的建

立和组装工作.

2 Keywords 零配置实现依赖注入

现在我们不再使用 Spring 配置文件来配置任何 Bean 实例, 那么我们只能指望 Spring 会自动搜索某些路径下的 Java 类, 并将这些 Java 类注册成 Bean 实例[2].

Rails 框架的处理比较简单, 它采用一种所谓的“约定优于配置”的方式, 它要求将不同组件放在不同路径下, 而 Rails 框架中是加载固定路径下的所有组件.

Spring 没有采用“约定优于配置”的策略, Spring 依然要求程序员显示指定搜索哪些路径下的 Java 类, Spring 将会把合适的 Java 类全部注册成 Spring Bean, 那现在的问题是: Spring 怎么知道应该把哪些 Java 类当成 Bean 类处理呢? 这就需要使用 Annotation 了, Spring 通过使用一些特殊的 Annotation 来标注 Bean 类. Spring 提供了如下几个 Annotation 来标注 Spring Bean[6].

@Component: 标注一个普通的 Spring Bean 类.

@Controller: 标注一个控制器组件类.

@Service: 标注一个业务逻辑组件类.

@Repository: 标注一个 DAO 组件类.

如果我们需要定义一个普通的 Spring Bean, 则直接使用 @Component 标注即可. 但如果用 @Repository、@Service 或@Controller 来标注这些 Bean 类, 这些 Bean 类将被作为特殊的 Java EE 组件对待, 能更好地被工具处理, 或与切面进行关联. 例如, 这些典型化 Annotation 可以成为理想的切入点目标.

在 Spring 未来的版本中, @Repository、@Service 和@Controller 还能携带更多的语义, 因此, 如果需要在 Java EE 应用中使用这些标注时, 尽量考虑使用 @Repository、@Service 和@Controller 来代替通用的 @Component 标注.

指定了某些类可作为 Spring Bean 类使用后, 然后使用@Autowired 对类成员变量、方法及构造函数进行标注, 完成自动装配的工作. 最后还需要让 Spring 搜索指定路径, 此时需要在 Spring 配置文件中导入 context Schema, 并指定一个简单的搜索路径[7].

下面, 我们通过 Spring 提供的 @Component 注释达到完全使用注释定义 Bean 并完成 Bean 之间装配的目标:

步骤一: 仅需要在类定义处, 使用@Component 注释就可以将一个类定义成为 Spring 容器中的 Bean. 下面的代码将 Pet 类定义为一个 Bean:

代码 6. 使用 @Component 注释的 Pet.java

```
package com.xiyou.test;
import org.springframework.stereotype.Component;
@Component
public class Pet {
    …
}
```

步骤二: 下面的代码将 House 类定义为一个 Bean:

代码 7. 使用 @Component 注释的 House.java

```
package com.xiyou.test;
import org.springframework.stereotype.Component;
@Component
public class House {
    private String HouseNo = "20140001";
    …
}
```

步骤三: 使用@Component 注释 Person.java, 并在 Person 类中通过@Autowired 注入前面定义的 Pet 和 House Bean.

代码 8. 使用@Component 注释的 Person.java

```
package com.xiyou.test;
import
org.springframework.beans.factory.annotation.Autowired
;
import
org.springframework.beans.factory.annotation.Required;
import
org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;
@Component("person")
public class Person {
    @Autowired
    private Pet pet;
    @Autowired
    private House house;
    …
}
```

@Component 有一个可选的传入参数, 用于指定 Bean 的名称, 在 Person 中, 我们就将 Bean 名称定义为“person”. 一般情况下, Bean 都是 singleton 的, 需要注入 Bean 的地方仅需要通过 byType 策略就可以自动注入了, 所以可不指定 Bean 的名称.

在使用 @Component 注释后, Spring 容器必须启用类扫描机制以启用注释驱动 Bean 定义和注释驱动 Bean 自动注入的策略.

步骤四: 在 Spring 的配置文件中指定搜索路径, Spring 将会自动搜索该路径下的所有 Java 类, 并根据这些 Java 类来创建 Bean 实例. 本应用的配置文件如下.

代码 9. 实现零配置的 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-3.1.xsd
    http://www.springframework.org/schema/context

\http://www.springframework.org/schema/context/spring-context-3.1.xsd">
    <!-- 自动扫描指定包及其子包下的所有 Bean 类 -->
    <context:component-scan
        base-package="com.xiyou.test"/>
    </beans>
```

这里, 所有通过 <bean>元素定义 Bean 的配置内容已经被移除, 仅需要添加一行 <context:component-scan/>配置就解决所有问题了——Spring XML 配置文件得到了极致的简化(当然配置元数据还是需要的, 只不过以注释形式存在罢了). <context:component-scan/> 的 base-package 属性指定了需要扫描的类包, 类包及其递归子包中所有的类都会被处理.

步骤五: 运行 BeanTest.java 代码时, 控制台也将正确输出相同的有关 Person 的信息.

3 Keywords XML配置与零配置比较

3.1 Keywords 零配置的优势

从以上我们分别通过 XML 配置和零配置实现依赖注入可以看出, 零配置相对于 XML 配置具有很多的优势[3]:

(1) 零配置的注释和 Java 代码位于同一个文件中, 而 XML 配置采用独立的配置文件, 大多数配置信息在程序开发完成后都不会调整, 如果配置信息和 Java 代码放在一起, 有助于增强程序的内聚性. 而采用独立的 XML 配置文件, 程序员在编写一个功能时, 往往需要在程序文件和配置文件中不停切换, 这种思维上的不连贯会降低开发效率.

(2) 零配置在 class 文件中, 可以降低维护成本, 零配置机制简单.

(3) 零配置不需要第三方的解析工具, 零配置可以充分利用 Java 的反射机制获取类结构信息, 这些信息可以有效减少配置的工作. 如使用 JPA 注释配置 ORM 映射时, 就不需要指定 PO 的属性名、类型等信息, 如果关系表字段和 PO 属性名、类型都一致, 甚至无需编写任务属性映射信息——因为这些信息都可以通过 Java 反射机制获取[8].

(4) 零配置在编辑期就可以验证程序的正确性, 使得对程序的错误控制变得容易. 从而在进行大型项目的开发时, 将有助于显著的提高开发效率.

因此在很多情况下, 零配置比 XML 配置更受欢迎, 零配置有进一步流行的趋势.

3.2 Keywords XML 配置的优势

然而, 作者认为, 虽然零配置具有一定的优势, 但是我们不能完全摒除原来 XML 配置的方式, 有一下几点原因[4]:

(1) 零配置不一定在先天上优于 XML 配置. 如果 Bean 的依赖关系是固定的, (如 Service 使用了哪几个 DAO 类), 这种配置信息不会在部署时发生调整, 那么零配置优于 XML 配置; 反之如果这种依赖关系会

在部署时发生调整, XML 配置显然又优于零配置, 因为零配置是对 Java 源代码的调整, 我们需要重新改写源代码并重新编译才可以实施调整.

(2) 零配置在业务类之间的关系方面不如 XML 配置容易把握.

(3) 如果 Bean 不是自己编写的类(如 JdbcTemplate、SessionFactoryBean 等), 零配置将无法实施, 此时 XML 配置是唯一可用的方式[9].

(4) 零配置通常是类级别的, 而 XML 配置在实现方面更加灵活. 例如相较于@Transaction 的事务注释, 使用 aop/tx 命名空间的事务配置方式将会更加灵活和简单.

(5) 最为重要的一点在于, 采用 XML 配置相对于零配置来说更加符合 Spring 框架的骨骼架构和设计理念.

① Spring 框架的骨骼架构

Spring 总共有十几个组件, 但是真正核心的组件只有几个, Spring 框架的总体架构图如图 1 所示[5]:

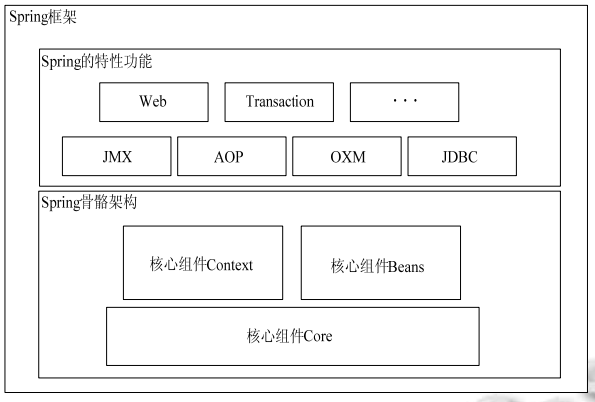

图 1　Spring 框架的总体架构图

从上图中可以看出 Spring 框架中的核心组件只有三个: Core、Context 和 Beans. 它们组成了 Spring 框架的骨骼架构. 如果没有它们, 就不可能有 AOP, Web 等上层的功能特性.

② Spring 框架的设计理念

前面介绍了 Spring 框架的三个核心组件, 如果再在它们三个中选出核心的话, 那就是 Beans 组件, 这是因为, Spring 就是面向 Bean 的编程(BOP,Bean Oriented Programming), Bean 在 Spring 中才是真正的核心[6].

Bean 在 Spring 中的作用就像 Object 对 OOP 的意义一样, 没有对象的概念就没有面向对象编程, Spring 中没有 Bean 也就没有 Spring 存在的意义. Bean 在 Spring 中如此重要的原因, 是由 Spring 框架的设计目标决定的, Spring 为何如此流行, 是因为原来 Spring 帮我们解决了一个非常关键的问题, 那就是它可以让我们把对象之间的依赖关系转而用配置文件来管理, 也就是他的依赖注入机制. 而这个注入关系在一个叫 IOC 容器中管理, 而 IOC 容器就是被 Bean 包裹的对象, 并且通过 XML 配置文件来加以实现. Spring 正是通过把对象包裹在 Bean 中来对这些对象进行管理和进行一系列额外的操作.

Spring 这种设计策略完全类似于 Java 实现 OOP 的设计理念, 当然 Java 本身的设计要比 Spring 框架复杂的多, 但是都是构建一个数据结构, 然后根据这个数据结构设计它的生存环境, 并让它在这个环境中按照一定的规律在不停的运动, 在它们的不停运动中设计一系列与环境或者与其他个体完成信息交换的操作. 其实我们用到的其他框架都是大慨类似的设计理念.

Bean 包装的是 Object, 而 Object 必然有数据, 如何给这些数据提供生存环境就是 Context 要解决的问题, 对 Context 来说他就是要发现每个 Bean 之间的关系, 为它们建立这种关系并且要维护好这种关系. 所以 Context 就是一个 Bean 关系的集合, 这个关系集合又叫 IOC 容器, 一旦建立起这个 IOC 容器后, Spring 就可以工作了. 而 Core 组件就是通过自身所拥有的一系列的工具来发现、建立和维护每个 Bean 之间的相互关系. 它们之间的关系如图 2 所示:

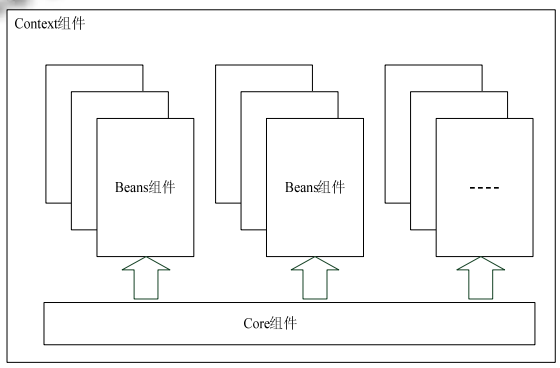

图 2　三个组件之间的关系图

所以, 综合 Spring 框架的骨骼架构和设计理念来看, 我们可以得出, Spring 框架的作用就是解决各层组件之间的硬编码耦合, 可以让各组件之间以“面向接

口”的方式编程, 从而提供更好的可扩展性. 换言之, Spring 框架的核心思想用两个字来描述, 那就是“解耦”. 也就是将应用程序的各个部分之间(包括代码内部和代码与平台之间)尽量形成一种松耦合的结构, 使得应用程序有更多的灵活性. 通过控制反转(IOC)的技术可以实现应用程序内部的”解耦”. 而控制反转主要含义就是将本来由应用程序来主动控制的调用等相关逻辑交由外部配置文件来进行被动的控制. 由于控制反转的概念相对比较广泛, 很多应用服务器实际上也实现了不同程度的控制反转技术, 只是这些应用服务器对应用程序的侵入性太强. 因此 Martin Fowler 专门写了一篇文章讨论控制反转这个概念, 并提出一个描述更为准确的概念, 叫做依赖注入(Dependency Injection). Spring 框架中的各个部分都充分使用了这种依赖注入的技术实现, 从而给应用以最大的灵活度. 简而言之, 要想在实现依赖注入的同时, 更好的实现“解耦”, 那就应该把原来耦合在 Java 代码中的应用逻辑代码, 提取到 XML 配置文件中进行管理. 而如果使用零配置, 将会再次把处理应用逻辑的代码交由 Annotation 管理, 而 Annotation 又耦合在 Java 源代码中. 并没有实现真正的“解耦”, 这也似乎有违 Spring 框架的设计理念.

然而, 虽然零配置似乎有违 Spring 框架的设计理念, 但在实际的应用中, 我们往往需要同时使用零配置和 XML 配置, 对于类级别且不会发生变动的配置可以优先考虑零配置; 同时对于那些后期可能进行调整的配置和第三方类则应优先考虑使用 Spring 的 XML 配置方式. Spring 会在具体实施 Bean 创建和 Bean 注入之前将这两种配置方式的元信息融合在一起[10].

4 结语

Spring 框架在 2.1 以后对零配置提供了强有力的支持, 零配置功能成为 Spring 框架的最大的亮点之一. 为了提高程序的内聚性和减少配置的工作量, 我们可以采用 Spring 的零配置方式. 但是这并不意味着传统 XML 配置将走向消亡, 在第三方类 Bean 的配置, 以及那些诸如数据源、缓存池、持久层操作模板类、事务管理等内容的配置上, XML 配置依然拥有不可替代的地位, 同时 XML 配置也更加符合 Spring 框架的设计理念. 总之, 在实际应用的开发期间我们用零配置, 这样在一定程度上不仅可以省去对 XML 配置文件的维护, 而且大大的提高了开发效率, 缩短了开发周期. 而在开发后期, 项目功能完成, 我们可以将零配置转换为 XML 配置, 禁用零配置即可. 从而显著的提高项目开发的效率.

参考文献

1 李刚.轻量级 Java EE 企业应用实战:Struts 2+Spring 3+Hibernate 整合开发.第 3 版.北京:电子工业出版社, 2012.
2 Mak G.丁雪丰等译.Spring 攻略.北京:人民邮电出版社, 2007.
3 陈雄华,林开雄.Sprin3.0 就这么简单.北京:人民邮电出版社, 2013.
4 Johnson R. Spring 2.0: What’s new and why it matters. http://www.infoq.com/articles/spring-2-intro.
5 Johnson R, et al. Spring framework reference manual. http://static.springframework.org/spring/docs/1.2.x/reference/index. html.