**第一部分Spring核心实现篇**

**第2章 Spring Framework的核心：IoC容器的实现**

朝辞白帝彩云间，千里江陵一日还。

两岸猿声啼不住，轻舟已过万重山。

-【唐】李白《早发白帝城》

**2.1  Spring IoC容器概述**

**2.1.1  IoC容器和依赖反转模式**

子曰：温故而知新。在这里，我们先简要地回顾一下有关依赖反转的相关概念。我们选取维基百科中关于依赖反转的叙述，把这些文字作为我们理解依赖反转概念的参考。这里不会对这些原理进行学理上的考究，只是希望提供一些有用的信息，以便给读者一些启示。这个模式非常重要，它是IoC容器得到广泛应用的基础。

维基百科对"依赖反转"相关概念的叙述

早在2004年，Martin Fowler就提出了"哪些方面的控制被反转了？"这个问题。他得出的结论是：依赖对象的获得被反转了。基于这个结论，他为控制反转创造了一个更好的名字：依赖注入。许多非凡的应用（比HelloWorld.java更加优美、更加复杂）都是由两个或多个类通过彼此的合作来实现业务逻辑，这使得每个对象都需要与其合作的对象（也就是它所依赖的对象）的引用。如果这个获取过程要靠自身实现，那么如你所见，这将导致代码高度耦合并且难以测试。

以上的这段话概括了依赖反转的要义，如果合作对象的引用或依赖关系的管理要由具体对象来完成，会导致代码的高度耦合和可测试性降低，这对复杂的面向对象系统的设计是非常不利的。在面向对象系统中，对象封装了数据和对数据的处理，对象的依赖关系常常体现在对数据和方法的依赖上。这些依赖关系可以通过把对象的依赖注入交给框架或IoC容器来完成，这种从具体对象手中交出控制的做法是非常有价值的，它可以在解耦代码的同时提高代码的可测试性。极限编程中对单元测试和重构等实践的强调体现了软件开发过程中对质量的承诺，这是软件项目成功的一个重要因素。

依赖控制反转的实现方式有很多种。在Spring中，IoC容器是实现这个模式的载体，它可以在对象生成或初始化时直接将数据注入到对象中，也可以通过将对象引用注入到对象数据域中的方式来注入对方法调用的依赖。这种依赖注入是可以递归的，对象被逐层注入。就此而言，这种方案有一种完整而简洁的美感，它把对象的依赖关系有序地建立起来，简化了对象依赖关系的管理，在很大程度上简化了面向对象系统的复杂性。

本篇将对Spring的核心IoC容器和AOP的实现原理进行阐述。IoC容器和AOP是Spring的核心，是Spring系统中其他组件模块和应用开发的基础。从两个核心模块的设计和实现上可以了解到Spring倡导的对企业应用开发所应秉持的思路，比如使用POJO开发企业应用、提供一致的编程模型、强调对接口编程等。对于这些Spring背后的开发思想和设计理念，大家都不会陌生，在Rod Johnson的经典著作里都有全面和深刻的讲解。作为参考，我们可以看到Spring官方网站对Spring项目的描述。如下图所示，Spring的目标和愿景写得很清楚。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091221/151647303.jpg> |
|  |

首先，Spring的目标在于让Java EE的开发变得更容易，这也就意味着Spring框架的使用也应该是容易的。对于开发人员而言，易用性是第一位的。为什么要让Java EE开发变得更容易，难道以前的Java EE开发很艰难？Spring究竟是如何让Java EE的开发变得更容易的呢？了解Java EE开发历史的读者都知道，正如Rod Johnson在他的著作Expert One-on-One Java EE Design and Development中提到的那样，EJB模型为Java EE开发引入了过度的复杂性，这个开发模型对Java EE的开发并不友好。有没有更好的开发模型呢？有，就是POJO！它让Java洗净铅华，恢复其自然的风采。使用POJO不仅能开发复杂的Java企业应用，而且还可以让Java EE开发在开发成本、开发周期、可维护性和性能上获得更大优势。对一般的企业应用需求而言，重要的是如何方便地使用应用需要的服务，而不是各种各样的开发模型和模式。虽然这些模式为我们描绘了设计高可靠性分布式应用的美妙场景，但这些场景是不是大多数企业应用开发者所要面对的呢？

世上都说Java好，唯有Spring忘不了。喜欢Java，是因为它简洁，不但包含了面向对象的语言特性，同时还可以跨平台，可谓是简洁而又强大。但是，进入到企业应用后，作为门外汉的自己一看到复杂的EJB模型就心生畏惧。这时候，我接触到了Spring，她给人的第一印象就是简洁却又具有丰富的内涵，就像第一次遇到Java一样，被她的这种特质深深地吸引了。她降低了企业应用开发的门槛，还原了POJO的本色，让我们直接依赖于Java语言，直接依赖于面向对象编程，使用无所不在的单元测试来保证代码质量，这样我们就有信心能够开发出高质量的企业应用。

也就是说，我们如何才能让开发既变得容易，又能享受到Java EE中提供的各种服务呢？Spring的目标就是通过自己的努力，让用户体会到这种简单之中的强大。同时，作为应用框架，Spring不想把自己作为另外一种复杂开发模型的替代，也就是说不是用另一种复杂性去替代现有的复杂性，那是换汤不换药，并不能解决问题。这就意味着需要有新的突破。要解决这个问题，需要降低应用的负载和框架的侵入性，Spring是怎样做到这一点的呢？

Spring为我们提供的解决方案就是IoC容器和AOP支持。作为依赖反转模式的具体实现，IoC容器很好地降低了框架的侵入性，同时也可以认为依赖反转模式是Spring体现出来的核心模式。这些核心模式是软件架构设计中非常重要的因素，比如说，我们常常看到的MVC模式就是这样的核心模式。不要小看这些体系结构模式的作用和影响，它们就是框架背后所谓的"道"。有了IoC容器和AOP的支持，用户的开发方式发生了很大的变化，具体说来，就是可以使用POJO来完成开发，对用户来说是简化了，但由于有平台的支持，依然能够实现复杂的企业应用开发。对于依赖反转，在Spring中，Java EE的服务都被抽象到IoC容器和AOP中并进行了有效地封装，而且因为依赖注入的特性，这些复杂的依赖关系的管理被反转了，它们的管理交给了容器。

Spring中各个模块的依赖关系可以用简单的IoC配置文件进行描述，信息集中并且明了。在使用其他组件服务时，只需要在配置文件中配置这些服务与应用组件的依赖关系。对应用开发而言，只需要了解服务的接口和依赖关系的配置。这样一来又很好地体现了Spring的第二个信条：让应用开发对接口编程，而不是对类编程。这样POJO使用Java EE服务时，可以将对这些服务实现的依赖降到最低，尽可能地降低框架的侵入性。

在处理与现有优秀解决方案的关系时，根据Spring的既定策略，它不会与这些第三方的解决方案发生竞争，而是致力于为应用提供使用优秀方案的集成平台。真正地把Spring定位在应用平台的地位，使得自己成为一个兼容并包的开放体系的同时，最大程度地降低开发者对Spring API的依赖，这是怎样实现的呢？答案还是IoC容器和AOP技术，也就是说，Spring API在开发过程中并不是必须使用的。

关于如何反转对依赖的控制，把控制权从具体业务对象手中转交到平台或者框架中，是解决面向对象系统设计复杂性和提高面向对象系统可测试性的一个有效的解决方案。它促进了IoC设计模式的发展，是IoC容器要解决的核心问题。同时，也是产品化的IoC容器出现的推动力。

注意  IoC亦称为"依赖倒置原理"（Dependency Inversion Principle），几乎所有框架都使用了倒置注入（Martin Fowler）技巧，是IoC原理的一项应用。SmallTalk、C++、Java或.NET等面向对象语言的程序员已使用了这些原理。控制反转是Spring框架的核心。

IoC原理的应用在不同的语言中有许多实现，比如SmallTalk、C++、Java等。在同一语言的实现中也会有多个具体的产品，Spring是Java语言实现中最著名的一个。同时，IoC也是Spring框架要解决的核心问题。

注意  应用控制反转后，当对象被创建时，由一个调控系统内的所有对象的外界实体将其所依赖的对象的引用传递给它。也就是说，依赖被注入到对象中。所以，控制反转是关于一个对象如何获取它所依赖的对象的引用的，在这里，反转指的是责任的反转。

我们可以认为上面提到的调控系统是应用平台，或者更具体地说是IoC容器。通过使用IoC容器，对象依赖关系的管理被反转了，转到IoC容器中来了，对象之间的相互依赖关系由IoC容器进行管理，并由容器完成对象的注入。这样就在很大程度上简化了应用的开发，把应用从复杂的对象依赖关系管理中解放出来。简单地说，因为很多对象的依赖关系的建立和维护并不需要和系统运行状态有很强的关联性，所以可以把我们在面向对象编程中常常需要执行的诸如新建对象、给对象引用赋值等操作交由容器统一完成。这样一来，这些散落在不同代码中的功能相同的部分就集中成为容器的一部分，也就是成为面向对象系统的基础设施的一部分。

如果对面向对象系统中的对象进行简单地分类，会发现除了一部分是数据对象外，其他有很大一部分对象都是用来处理数据的。这些对象并不会经常发生变化，是系统中基础的部分。在很多情况下，这些对象在系统中以单件的形式存在就可以满足应用的需求，而且它们也不常涉及数据和状态共享的问题。如果涉及数据共享方面的问题，需要在这些单件的基础上做进一步的处理。

同时，这些对象之间的相互依赖关系也是比较稳定的，一般不会随着应用的运行状态的改变而改变。这些特性使得这些对象非常适合由IoC容器来管理，虽然它们存在于应用系统中，但是应用系统并不承担管理这些对象的责任，而是通过依赖反转把责任交给了容器（或者说平台）。了解了这些背景，Spring IoC容器的原理也就不难理解了。在原理的具体实现上，Spring有着自己的独特思路、实现技巧和丰富的产品特性。关于这些原理的实现，下面会进行详细的分析。

第1章中，我们已经对建立本地源代码环境做了简要的介绍，该源代码环境是我们分析Spring原理前要做的重要准备工作。同时，我们还需要针对IoC容器做一些额外的事情：根据Spring 3.0的源代码组织特点，每个模块作为独立的Eclipse项目存在，所以现在需要在Eclipse中建立与IoC容器和上下文相关的代码项目。这样就可以方便地使用Eclipse的代码分析工具来对相关模块的实现进行分析。这个额外的准备过程在分析其他模块时也是需要的，所以这里会做一个说明。

准备过程如图2-1所示，打开Eclipse，依次选择File→Import→General→Existing Projects into Workspace，然后再选择org.springframework.beans和org.springframework.context两个目录，并将其导入到Eclipse本地环境中。这时即可看到在Package Explorer View中的Spring IoC容器的源代码项目。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091221/152039318.jpg> |
|  |