第 38 卷 第 2 期
Vol.38 No.2

计 算 机 工 程
Computer Engineering

2012 年 1 月
January 2012

• 软件技术与数据库 •

文章编号：1000—3428(2012)02—0069—03 文献标识码：A 中图分类号：TP311.5

基于 Spring 的构件动态演化机制

仇书礼，初佃辉，孟凡超

(哈尔滨工业大学(威海)计算机科学与技术学院，山东 威海 264209)

摘 要：针对 Spring 开源框架不支持动态演化的问题，提出一种基于 Spring 的构件动态演化机制。在设计开发模式上，对系统进行面向业务逻辑和配置文件的模块化划分，在体系结构上，引入演化代理，对模块间调用进行解耦，模块调用由实例管理中心进行统一管理控制。在 Spring 框架中实现该机制，并通过计算 Π 值进行实验验证，结果表明，该机制可以使软件系统在运行期间实现演化，对系统效率基本无影响。

关键词：构件；动态演化；控制反转；解耦；演化代理；实例注册中心

Component Dynamic Evolution Mechanism Based on Spring

QIU Shu-li, CHU Dian-hui, MENG Fan-chao

(School of Computer Science and Technology, Harbin Institute of Technology at Weihai, Weihai 264209, China)

【Abstract】For the issue of that Spring, the open source framework does not support dynamic evolutions, this paper comes up with a component dynamic evolution mechanism based on Spring, from the view of the development mode and architecture of software system. This mechanism divides the system's business logic and configuration files into many modules from the point of development mode. In the aspect of architecture, it leads in evolution agent to decouple the invoking among modules, while modules and their callings are managed and controlled uniformly by instance management center. This mechanism is implemented in the Spring framework, and it is proved by calculating the value of Π. Experimental results show that the mechanism can let system implement evolutions at run-time and does not influence the system efficiency.

【Key words】component; dynamic evolution; inversion of control; decouple; evolution agent; instance registration center

DOI: 10.3969/j.issn.1000-3428.2012.02.022

1 概述

瞬息万变的市场要求企业的灵魂、核心的业务应用系统必须迅速地适应需求变化，能够根据需求的变化快速响应需求，变更应用系统[1]，同时，全天候的业务执行又要求企业软件能够 7×24 h 地运行，这些都凸现出动态演化在企业软件系统中的重要性。演化性是软件的基本属性[2]，动态演化是指软件在执行期间的软件演化，它使软件不会存在暂时的失效，具有不间断服务的明显优点[3]。

Spring 框架是目前 Java 企业软件开发领域最流行的软件开发技术，它是一个轻量级的控制反转和面向切面的开源容器框架。控制反转，又称依赖注入，是 Spring 框架的核心，即将所有对象创建的工作交由第三方(Spring 容器)来控制。对控制反转的原生态支持使 Spring 容器具有强大的黏附力，任何一个 Java 应用都可以轻易地集成到 Spring 容器中，几乎所有的 Java 开源项目都提供了对 Spring 的支持。面向切面是 Spring 框架的另一重要特点，它使 Spring 容器中的所有 Java 对象能够享受统一的无缝的切入服务，如日志、事务等。

Spring 框架给企业软件开发者带来了很大便利，然而它仅将 Java 对象间的关系解耦，却没有提供对动态演化的支持，造成了企业系统软件不得不在停止之后再进行升级，影响了企业的服务质量和服务效率。所以，本文针对 Spring 的这个不足，提出基于 Spring 的动态演化机制，并对这个机制进行了实现。

2 相关研究工作

目前，研究人员对软件的演化机制展开了很多深入的研究，主要集中在面向服务、构件等方面。

在面向服务方面，文献[4]提出了一个面向服务对象的动态演化模型，该模型借鉴面向服务的服务注册和查询机制，利用服务对象注册表解耦对象的引用实现软件演化；文献[5]提出一种解决 OSGi 平台上服务动态演化的方法，根据重定向方法来解决服务类定义的动态更新，使用实现和数据相分离的方法来解决演化中公共数据的一致性。

在构件方面，文献[6]使用了 SOFA 的动态构件更新(DCUP)方法，DCUP 提供了一组相互正交的抽象，从而支持构件在运行的应用中进行更新；文献[7]设计出了一种 3RDBA 方法，可用于替换长时间运行系统中的组件，还可根据所作的演化取得各种必要的信息；文献[8]设计了网构软件系统的动态演化模型，给出了构件增加、删除和更新操作的需求算法，并引入一致性检查模型保证了演化的可靠性和安全性。

3 基于 Spring 的构件动态演化机制

要实现 Spring 框架下的动态演化，必须从软件系统的开发模式、体系结构上入手。

如图 1 所示，从开发模式上将系统划分为多个模块，每个模块包含相应的业务逻辑和配置文件。模块是耦合度最小

基金项目：国家“863”计划基金资助项目(2008AA04Z101)；“核高基”重大专项(2009ZX01045-001-002-4)；山东省科技发展计划基金资助项目(2011GGX10108, 2010GGX10104, 2010GGX10116, 2010GZX20126)；威海市科技攻关计划基金资助项目(2011GGA00201109 22082212)

作者简介：仇书礼(1987－)，男，硕士研究生，主研方向：软件工程，软件体系结构；初佃辉，副教授；孟凡超，副教授、博士

收稿日期：2011-07-22 **E-mail：**mengfanchao74@163.com

的单元，也是独立部署、测试的基本单位，每个模块以配置文件为基本的识别因素。配置文件是模块内各对象依赖的描述配置体，它的引入仍然是一种低层次上的解耦，配置的对象在运行过程中仍不能进行动态替换，由此提出体系结构上的解耦机制。

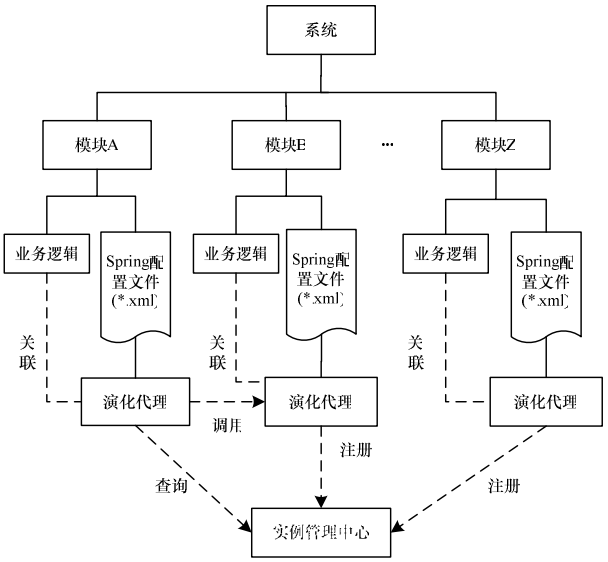

图 1 开发模式与体系结构

如图 1 所示，在体系结构上引入实例管理中心和演化代理 2 个角色。实例管理中心负责模块之间依赖对象的注册工作，同时它以接口的形式为其他模块提供对象引用，在引用的过程中可以根据一定的选择规则进行实例的提取。演化代理位于每一个模块中，它依据配置信息自动创建实例注册和实例引用的代理，通过这个中介的作用，注册者可以动态地进行增加、替换和删除，而引用者无需关注这些处理(如图 2 所示)。

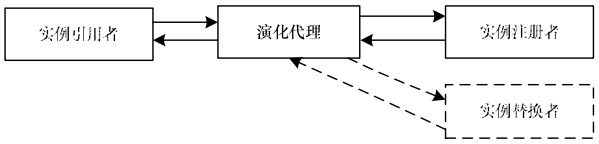

图 2 演化代理的作用

定义 1 实例管理中心(Instance Manage Center, IMC)，管理系统中各模块间相互依赖的实例，它是一个三元组，由实例注册对象集合、实例引用对象集合和实例注册与引用算法组成：

IMC={IREOS, IRFOS, RERFA}

其中，IREOS 指实例注册对象的集合，对实例进行包装以供选择与替换；IRFOS 指实例引用对象的集合，是模块在访问注册对象时所产生的引用信息，这些信息为服务替换提供依据；而实例注册和引用的策略由专门的算法 RERFA 来控制。

定义 2 实例注册对象(Instance Registration Object, IREO)，包含对象实例、注册接口、注册数据和实例运行数据 4 个属性：

IREO={OI, I, RED, RUD}

其中，OI 是对象实例，它是产生服务行为的主体；I 是注册的接口，在注册对象被引用时该属性是关键的选择要素；RED 是注册时指定的数据，用来服务选择时作为参考要素，如版本号等；RUD 是对象实例 OI 运行时使用的数据。

定义 3 实例引用对象(Instance Reference Object, IRFO)，包含引用接口、操作接口、引用数据和实例注册对象 4 个部分：

IRFO={I, OI, RFD, IRFO}

其中，I 是要引用的接口，它是查找实例注册对象的关键判定条件，引用数据(RFD)起到对多个对象的过滤作用，操作接口(OI)供引用对象的使用方调用。

定义 4 演化代理(Evolution Agent, EA)，负责本模块内的实例注册和引用工作，它包含实例注册的接口、实例引用的接口、实例注册对象列表和实例引用列表 4 个部分：

EA={REI, RFI, REOS, RFOS}

其中，REI 为模块提供实例注册的接口，即生成实例注册对象并将其放到实例管理中心；RFI 是为模块提供实例引用的接口，即从实例管理中心中查找满足条件的实例注册对象。REOS 和 RFOS 分别为实例注册对象列表和实例引用列表，它们是实例管理中心实例集合一个引用。

根据以上分析和定义，可以得出整个机制的详细运作流程如图 3 所示，大致过程为：

(1)模块 B 在部署时，其 Java 实例 B1 通过演化代理注册到实例管理中心，演化代理根据注册接口等配置信息创建实例注册对象 Z。

(2)模块 A 在部署时，其内的演化代理根据模块 A 的配置信息请求实例管理中心创建实例引用对象 1，实例引用对象 1 在根据请求的接口等信息查找到实例注册对象 Z。

(3)演化代理将实例引用对象 1 注入到实例 A2，A2 通过实例引用对象 1 提供的操作接口调用模块 B 的实例 B1 方法。

(4)如果模块 B 被更换到了模块 B’，则只需通过演化代理将实例注册对象 Z 的对象实例属性指向模块 B’的实例 B’1。

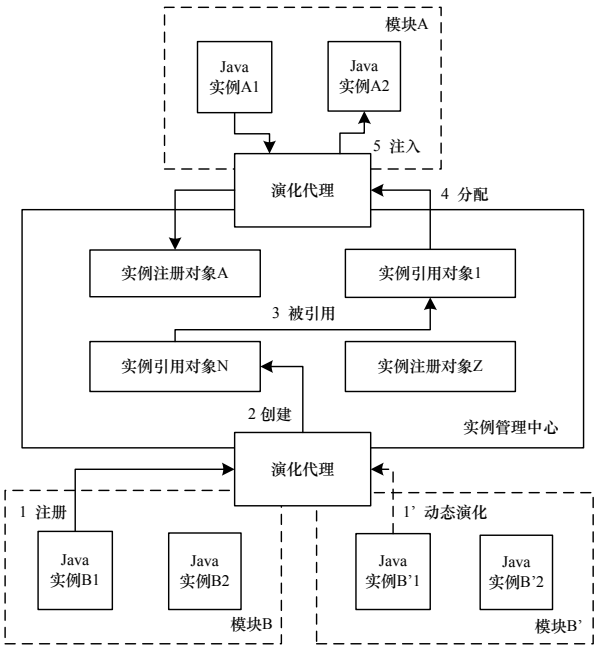

图 3 演化机制原理

该机制具有的优点如下：

(1)大大降低了 Spring 系统模块之间的耦合性，从而使系统具有较强的可维护性和动态演化性。

(2)通过引入实例注册对象实现了动态演化中运行数据的分离，通过引入实例引用对象实现了动态演化中实例行为的分离。

(3)通过引入演化代理提高了机制的可用性，使用者只需简单操作即可使系统具有动态演化的功能。

4　基于 Spring 的构件动态演化实现

要实现动态演化，第一要务是要能控制 Spring 容器，经过对源代码的分析，归纳出以下要点：

(1)获取 Spring 容器：创建一个 Servlet 监听器，该监听器从 ServletContext 中获取容器；

(2)在 Spring 容器中动态增加实例配置文件和动态删除实例：获取容器的具有自动织入实例功能的实例工厂，随后为工厂创建一个配置阅读器，该阅读器可完成配置文件的读取及实例创建的工作，该工厂的 removeBeanDefinition 方法可根据名称删除实例。

在控制住 Spring 容器的基础上，即可对演化机制进行实现，如图 4 所示。

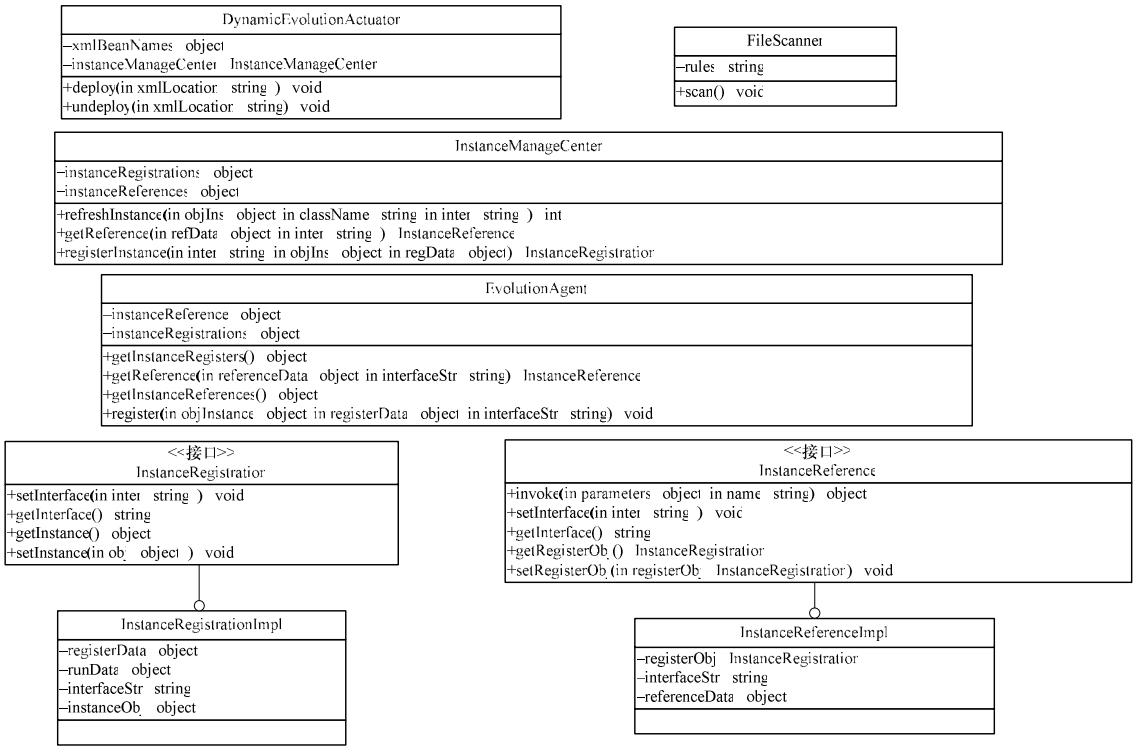

图 4　演化机制实现类示意图

本文机制的实现主要包含三大部分：

(1)实例注册和实例引用对象：该部分关键点是实例引用对象的 invoke 方法的设计，它用来完成对实例注册对象中对象实例的调用，接收 2 个参数：调用的方法名和传递的参数，返回结果是相应对象实例方法的执行结果。

(2)演化代理器和实例管理中心：该部分关键点是实例管理中心的动态替换实现方法，它用来动态替换实例注册对象中的对象实例，接收 3 个参数：新对象实例的引用，新对象实例的接口名称和新对象实例的类全称，后 2 个参数是替换判断的关键条件，这个方法将根据接口名称和类名在实例注册对象列表中查找要替换的对象，若找到，则执行引用转移，否则将该实例注册到管理中心。

(3)演化执行器和文件扫描器：它将完成 Spring 配置文件的扫描、实例管理中心的生成等工作，该部分的关键要是：创建个时钟任务，周期性地触发这个任务并扫描满足一定约束条件的配置文件，并判断配置文件的增加与删除，在增加时调用需完成 XML 的解析与实例注册、引用的工作，在删除时需将该 XML 对应的实例进行从 Spring 容器清除。

通过在 Spring 容器 XML 配置文件中合理地配置以上各类，这样即可在 Spring 容器中达到动态实例更新、增加与删除的工作，可以在不重启的情况下完成企业系统业务模块的更新与增加删除。

5　基于 Spring 的动态演化实验

实验提供一个计算∏值服务，由客户端指定具体的精度要求，采用传统的 MVC 软件设计模式。实验所用计算机采用 CPU 为英特尔 Pentium 双核 E5300 2.60 GHz，内存为 2 GB，带宽为 100 Mb/s 的局域网。运行操作系统为 Windows XP，所有代码均在 Eclipse 下实现，使用了 Spring 2.5 框架。

实验首先对比了基于 Spring 的动态演化系统和纯 Spring 系统之间的运行效率，如图 5 所示。依次对 2 个系统运行进行分析，每个系统各请求服务 2 次，横轴表示运算输入的精度要求，精度为 80x+60。从实验数据上可以看出，2 个系统下服务的运行效率并无明显差别。

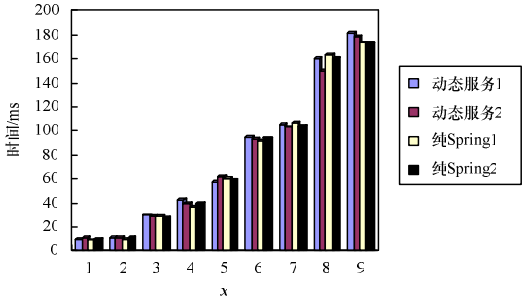

图 5　2 类系统运行效率比较

图 6 反映的是在 Spring 动态演化系统中，客户端连续调用服务 100 次、输入精度要求为 460 时的响应曲线。在调用第 10、30、50、70、90 次后分别进行了 5 次动态更新，从曲线可以注意到，5 次动态更新对系统的效率影响极小。通过多次实验，结果类似，由此得出结论，机制服务的动态更新对客户的调用产生的影响有限，满足应用要求。

(下转第 74 页)