轻量级 IoC 容器的研究与设计

娄 锋,孙 涌
(苏州大学 计算机科学与技术学院,江苏 苏州 215006)

摘 要:研究了支持依赖注入的 IoC 容器的设计问题。其目的是减小容器中组件或对象间的耦合度,应用程序中的组件在运行时不再主动调用其协作组件或协作对象,而是由容器在程序运行期间动态地将应用程序所需的组件或对象注入到应用程序中。与当前存在的 IoC 容器相比,它能够实现更多的组件注册方式,组件的管理方式也更灵活,基于 IoC 容器进行的开发可以明显减小对象间及对象和服务间的耦合度,便于组件的重用,缩短程序开发的周期,构建清晰、简洁的解决方案。

关键词:翻转控制;依赖注入;轻量级容器
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2007)01-0091-03

Design and Implementation of Lightweight IoC Container

LOU Feng, SUN Yong
(School of Computer Science & Technology, Soochow University, Suzhou 215006, China)

Abstract: Study the design and implementation of IoC containers. It aimed at reducing applications coupling between application components in operation, no longer use their own initiative in collaboration components or collaborators, on the contrary the container allows the configuration and wiring together of objects using dependency injection. Compared to the existing IoC containers it can achieve more components registered types, components management are more flexible. The use of IoC containers can clearly reduce the coupling between services for ease of components. The development of software design is shorten, and will give developers a clear and concise solution.

Key words: inverion of control; dependency injection; lightweight container

0 引 言

J2EE 提供了一个能够整合企业各种服务的开发平台,但是并没有提供一个令人满意的程序开发模型。很多组件模型为了兼容更多的中间件厂商,在开发中使用了"大而全"的重量级框架,由于在绝大多数项目中并不全需要使用组件提供的各种服务,导致人为地增加了程序开发的复杂性。由于组件提供的各种服务的复杂性,开发者很难真正掌握组件的内涵,在开发中遇到的问题往往很难分清楚是开发者自身出现的错误,还是组件提供的服务本身有问题。针对这些问题,轻量级 IoC 容器应运而生。因为通过研究发现,基于轻量级容器进行的开发能够极大地简化开发的复杂性,降低开发的难度及减小企业的开发成本。

1 IoC 简介

IoC[1]全称 Inversion of Control,是 Apache Avalon 项目创始人之一 Stefano Mazzocchi 提出的,目的是强调设计的安全性。IoC 模式并不是一种全新的设计理念,它是建立在面向对象的基础之上的。IoC 即控制反转,著名的好莱坞原则:"Don't Call us, We will call you",以及 Robert C. MartinFowler 在其敏捷软件开发中所描述的依赖倒置原则(Dependency Inversion Principle, DIP),都是这一思想的体现。依赖注入(Dependency Injection)是 Martin FowLer 对 IoC 模式的一种扩展的解释。IoC 是一种用来解决组件之间依赖关系、配置及生命周期的设计模式,其中对程序中的组件依赖关系的处理是 IoC 的核心。

IoC 的类型主要有以下几种:

1)基于方法(Method)的 IoC。

在每个方法调用中传递其依赖的组件。如果方法需要某个组件,就把该组件作为参数传递给方法。

2)基于接口的 IoC(type 1)。

使用接口来声明依赖。EJB 容器就是一个基于接口的重量级容器,部署在它内部的 EJB 组件需要使用接口来声明依赖关系。

3)基于 Setter 方法的 IoC (type 2)。

使用 setters 来设置依赖组件,使用 getters 来取得组件。把依赖的组件作为一个属性,通过 setters 方法来动态设置其依赖的组件。

4)基于构造函数的 IoC(type 3)。

使用构造函数来声明依赖。通过传递组件参数到构造函数中,来实现依赖关系。

收稿日期:2006-04-18
作者简介:娄 锋(1978-),男,山东邹平人,硕士,主要研究方向为软件工程;孙 涌,教授,硕士生导师,主要研究方向为软件工程、智能化信息处理。

2 IoC 模式与工厂模式

IoC 的设计思路正是基于上述问题而被提出来的。可以把 IoC 模式看作是工厂模式的升华，即把 IoC 看作是一个大工厂，只不过这个工厂要生成的对象都是在程序中或者是在配置文件中给出定义的，然后利用 Java“反射机制”，根据给出的类名生成相应的对象。从实现来看，IoC 是把以前在工厂方法不灵活的对象生成代码，改变为配置文件来定义，也就是把工厂和对象生成这两者独立分开，这样就大大提高灵活性和可维护性。

在传统的工厂设计模式中，需要知道工厂所创建的所有类对该工厂的实现，这严重限制了可扩展性：如果工厂对象不知道它们的具体类，工厂对象将不能创建对象。在轻量级容器设计中当然需要克服这种局限性。

2.1 IoC 容器的特点

IoC 容器是在反复研究设计原则、开发方法，并在结合实际开发经验基础上提出来的，基于轻量级容器的开发可以更好更快地满足客户需求[2]，构建企业级应用程序，包含以下 4 点：

1)基于轻量级容器的开发的主要目的，就是减少对象间及对象和服务间的耦合度，便于程序的复用和维护。使用 IoC 容器，组件没有被要求实现、扩展或使用任何 IoC 容器的类或者接口，在任何地方重用组件都是可行的。

2)尽量使用简单的方法解决问题，构建清晰、简洁的解决方案，缩短程序开发的周期。简单性并不意味着使应用程序变得简单，而是如何使程序中模块间的调用以简单的方式来完成。

3)轻量级开发的设计是建构在 POJO(Plain Old Java Object)对象的基础之上的，减少重量级面向组件模型的使用，例如 EJB。

4)自动化单元测试。优先编写测试用例，测试会间接带来重构代码的便利。单元测试提高代码的设计水平，解耦联系过于紧密的代码。

2.2 设计思想

本容器主要是按照以下的思路来设计：首先，将系统按照功能划分模块，各个模块之间相对独立，并通过特定的方式交互；然后，在各个模块中设计相应的组件，具体化各模块的功能，实事上各个模块的功能是由组件的组合或集成来实现的。其主要的设计思想如下：

(1)基于依赖注射的思想。

用程序代码应避免依赖于容器，只有很简单的对象单独地工作，大多数商业对象之间存在依赖，比如其它的商业对象、数据存取对象和资源等。可以肯定的是，对象需要查找其它的管理对象和资源，这种需要只有依赖容器来满足。反转控制和依赖注射可以满足这种需要[3]。它们不需要引进对容器的依赖，就能够满意地查询定位其它的管理对象和资源。

反转控制是框架领域的一个重要概念，它允许一个对象在上层接受其它对象的创建。使用反转控制方式，可以让你的对象从创建中释放出来，降低了藕合度[4]。

(2)分离的思想。

IoC 容器和容器的功能应该是分离的。它是在软件设计中处理复杂性的一个重要原则。首先，分离技术问题和业务问题，通过不断地重构，系统功能逐渐演变为框架，而业务功能则演变为组件；其次，分离框架的核心逻辑和实现细节，通过一个微内核和若干接口，把具体的系统功能从框架核心中剥离出来，以系统组件的形式加以实现；最后，分离组件的功能需求和非功能性需求，从而容器的各个功能模块应该是以插入式方式来为组件提供相应的容器服务。

(3)动态配置思想。

容器管理的组件所需要的资源应该是可定制的，这种资源不应该在组件初始化时预先绑定，而是在程序中由容器根据依赖关系动态加载。

(4)微内核思想。

微内核设计模式可应用到需求不断变化的软件系统，微内核将系统最基础的核心服务与扩展功能、用户相关的服务进行分离。

最核心的服务封装在微内核内，功能都封装微内核之外。微内核可以被看作可插入外部扩展功能的总线，并协调这些扩展服务的交互，使得运行在独立进程空间中的部件能相互通信。此外，微内核还负责维护系统级的资源管理，提供对外的接口[5]。客户通过适配器访问扩展服务，微内核提供底层的通信设施。

2.3 容器功能模块

容器主要包括几部分：容器核心、依赖解析、配置组件、生命周期管理、缓存管理、容器优化技术。

IoC 容器主要是为应用程序组装组件，容器采用了工厂设计模式，主要是创建和分发组件或者对象，不像其他工厂模式的实现，它们只是分发一种类型的对象，而 IoC 容器是一个通用的工厂，可以创建和分发各种类型的组件。容器的整体架构图如图 1 所示。

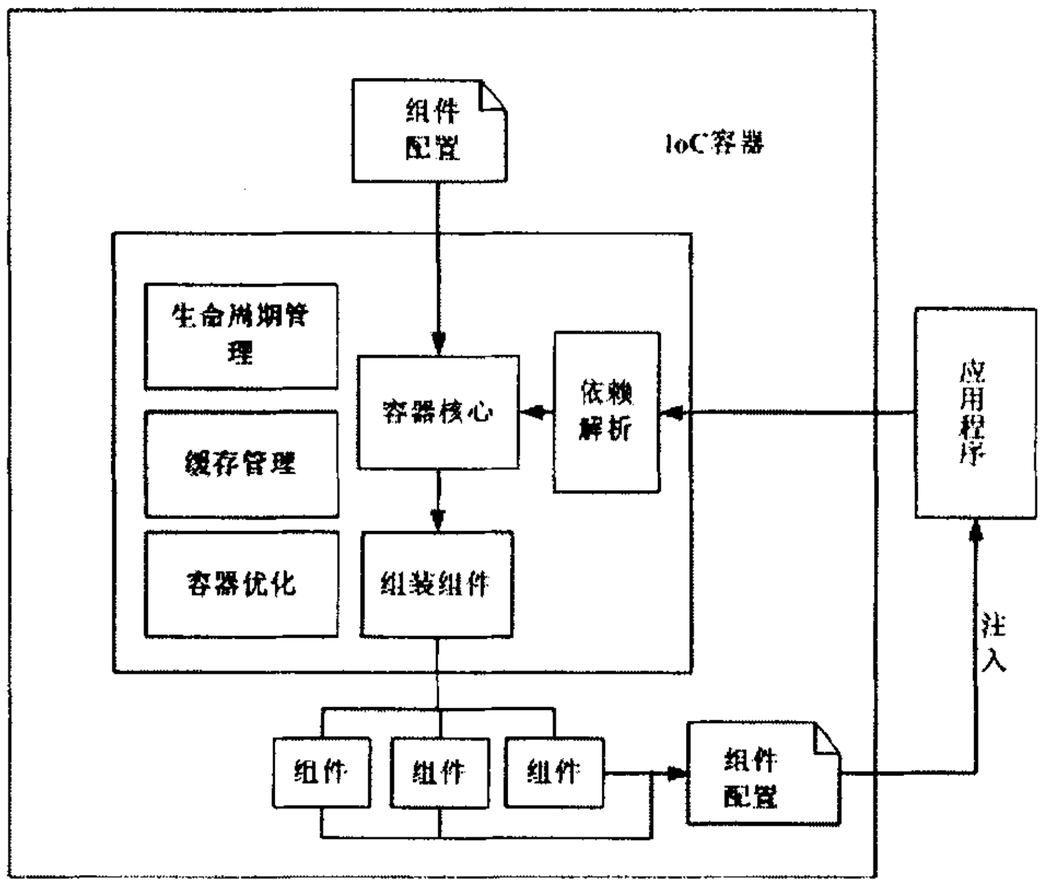

图 1 容器整体架构图

除了实例化对象和分发对象以外,容器还有很多工作要做,由于容器知道应用系统中的很多对象,所以它在实例化这些对象的时候,创建协作对象的管理关系,这样就把配置的负担从对象自身以及对象的调用者中脱离出来。这样,容器分发出来的对象都已经配置好,得到了它们的关联对象,已经可以被使用了。

容器还要参与对象的生命周期,调用用户定义的初始化和销毁方法。此外为了提高容器的性能容器还应该采用容器优化技术。

2.4　容器工作原理

在通常的容器执行业务逻辑时,首先由容器生成业务逻辑对象所需要的功能类实例,接着执行业务逻辑时再调用功能类实例中的方法。而 IoC 容器的工作原理与此截然不同,其工作原理如图 2 所示。

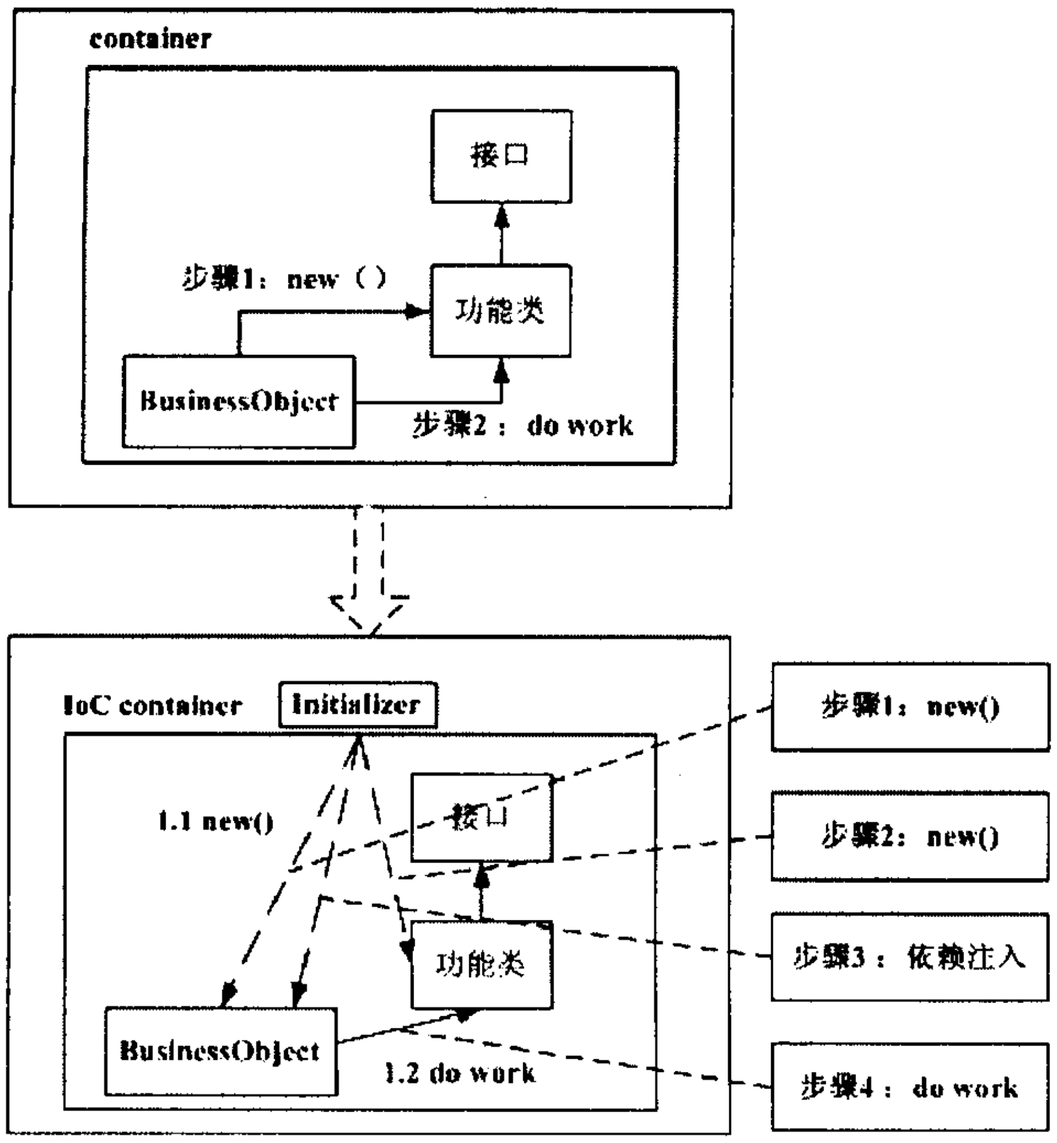

图 2　容器工作原理图

第一步:执行业务逻辑的对象,这与一般的容器相同;

第二步:当执行业务逻辑对象时发现此对象有依赖对象,生成其依赖对象;

第三步:执行依赖注入,将业务逻辑对象的依赖对象功能类实例通过某种注入方式,注入到业务逻辑对象中;

第四:步执行业务逻辑。

3　组件注册过程和删除方法

3.1　注册过程

容器管理的对象是组件或者是单个对象,每一个组件都有一个 id。这个 id 在管理组件的容器中必须是惟一的。一个组件有且只有一个 id, 在 Container 中可以使用 id 或者组件名来对组件进行注册和删除。

从图 3 中可以看出容器在一个组件可以被使用之前要完成很多工作,每一步的详细内容如下:

1)容器查找组件的定义并且将其实例化。

2)使用依赖注入,容器按照 Bean 定义信息配置 Bean 的所有属性。

3)容器调用 components 中的 SetBeanName()方法传递 Bean 的 ID。

4)容器调用 setBeanFactory()方法传递工厂自身。

5)如果有 BeanPostProcessor 和 Bean 关联,那么它们的 PostProcessBeforeInitialzation()方法将被调用。

6)如果 Bean 指定了 init - method 方法,它将被调用。

7)最后,如果有 BeanPostProcessor 和 Bean 关联,那么它们的 postProcessAfterInitialization()将被调用。

这时,Bean 组件可以被应用系统使用了,并且保留在容器中直到不再被使用为止。

3.2　删除方法

容器中有两种方法可以将 Bean 从容器中删除:

1)如果 Bean 实现了 DisposableBean 接口,destory()方法将被调用。

2)如果指定了定制的销毁方法,就调用这个定制的销毁方法。组件在实例化的时候,有时候需要作一些初始化的工作,然后才能使用,同样当组件不再需要,从容器中删除的时候,需要按照顺序做一些清理工作,因此容器在创建和销毁组件的时候要调用容器中的方法。

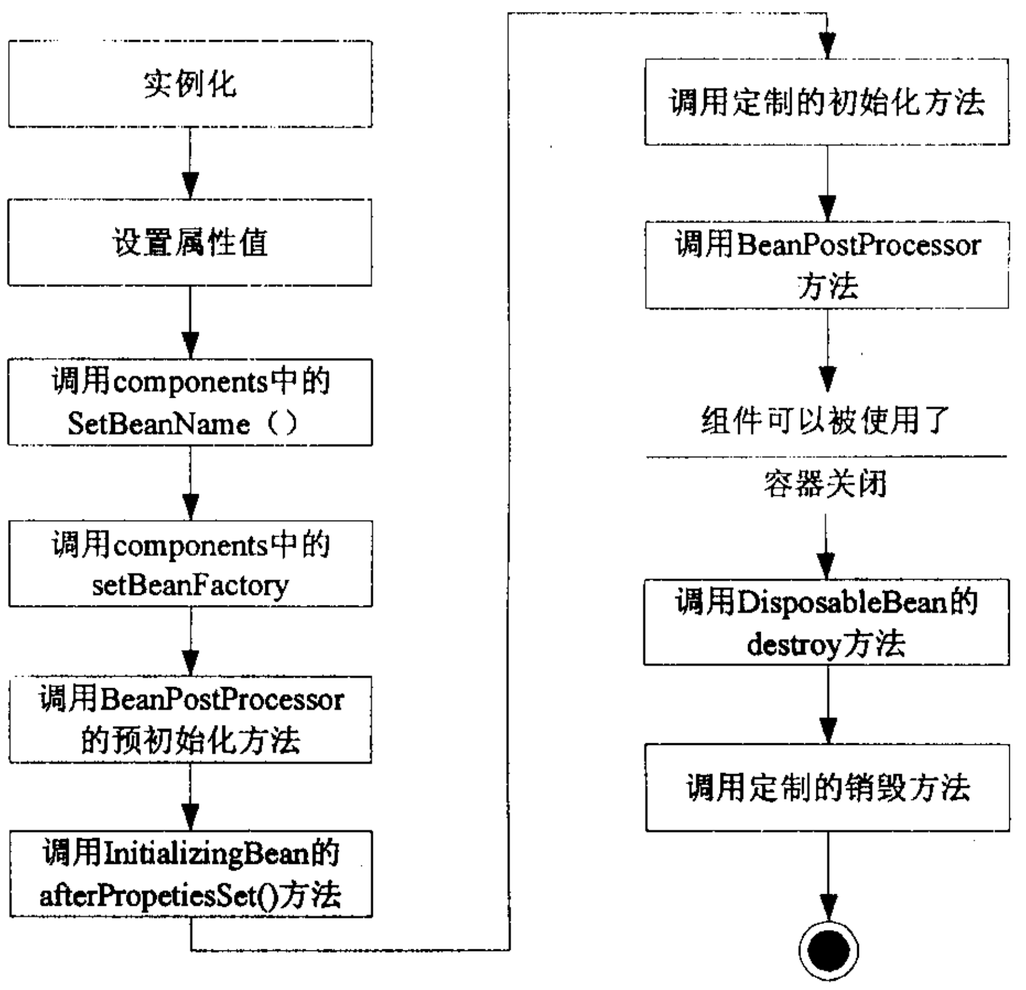

图 3　Java Bean 类型组件在容器中的注册过程

在组件的定义中设置自己的初始化方法,这个方法在组件被实例化的时候马上被调用。同样也可以设置自己的销毁方法,这个方法在组件从容器中被删除之前调用,这样的好处就是不需要配置就可以调用它们的方法。在容器中要实现这样的功能需要实现容器中的接口,因此在容器外不能单独运行这样的程序。容器中的接口是 InitializingComponent 和 DisposableComponent, InitializingComponent 接口中定义了一个方法 afterPropertiesSet(),这个方法在组件的所有属性被设置好后调用,同样 DisposableComponent 的方法 destroy()在组件从容器中删除的时候被调用。

(下转第 97 页)

```
# include "top_rm.vrh"
//引用顶层 RM 类
# include "bfm.vrh"
//引用测试平台的激励生成类
program test_top {
  RM rm; //例化顶层 RM
  BFM bfm; //例化 BFM
  fork //并行进程执行
    rm.rm_run(); //启动整个 RM 系统
    bfm.bfm_run(); //施加测试激励
    runtime_ctrl(); //设定仿真时间
  join any
}
```

时钟的引入是 RM 设计不可或缺的一环。在单独进行 RM 仿真时对于只有一个时钟域的情况下使用 OpenVera 定义的系统时钟(System CLOCK)。这个时钟是一个虚拟的时钟,当它与 RTL 捆绑后就具有了与 RTL 在物理上一致的时钟概念。而对于有很多个时钟域的情况,通过引入虚端口(Virtual Port)的方法将在 RTL 中定义好的时钟通过包含这些时钟信号的接口文件(.if.vrh)施加到 RM 上。这些都保证了在纯 Vera 仿真(standalone Vera simulation)模式下仍然可以获得与 RTL 一致的仿真环境,增加了验证的可靠性,实现了后期 RTL 和 RM 联合仿真同步运行的目的。

图 8 为 RTL 和 RM 运行结果对比(上半部分为 RTL 运行结果,下半部分为 RM 运行结果),可以看出 RM 运行时间差不多只相当于 RTL 的四分之一。这说明了基于更高抽象行为级描述的 RM 在仿真速度上具有 RTL 不可比拟的优势,直接方便了调试测试平台(Testbench)和测试向量的工作。

4　结束语

根据数字集成电路设计中验证的需要,提出了一个包括验证平台、参考模型和测试向量在内的一整套基于 OpenVera 的验证环境。实例表明基于 OpenVera 的参考模型可缩短仿真调试时间,方便早期验证平台的搭建。另外还可做到 RM 与 RTL 的模块替换,直接在已搭好的验证平台上进行模块级调试,减轻了模块级验证的负担。

```
Vera: finish encountered at time  605688 cycle   80759
        total       mismatch: 0
                  vca_error: 0
              fail(expected): 0
                      drive: 2055227
                     expect: 0
                     sample: 1997912
                       sync: 9698712
          VCS  Simulation  Report
Time: 164248778 ps
CPU Time:    148.830 seconds;       Data structure size: 169.8Mb
Tue Oct 25 10:52:46 2005
```

```
Vera: finish encountered at time  605768 cycle   80759
        total       mismatch: 0
                  vca_error: 0
              fail(expected): 0
                      drive: 77654
                     expect: 0
                     sample: 74312
                       sync: 891520
          VCS  Simulation  Report
Time: 14183740 ps
CPU Time:    35.640 seconds;       Data structure size: 55.8Mb
Tue Oct 25 14:13:18 2005
```

图 8　单独 RTL 和纯 Vera 运行的仿真时间对比

5　致　谢

本项目的完成得益于深圳市国微通讯有限公司提供的全方位的软件平台和硬件设施,同时也得到了该公司 ASIC 设计部项目组组长唐炎和崔松叶工程师的精心指导,在此谨致谢忱。

参考文献:

[1] Rashinkar P, Paterson P, Singh L. System on a Chip Verfication - Methodology and Techniques[M]. New York: Kluwer Academic Publisher, 2002:107 - 115.

[2] Bergeron J. Writing Testbenches Functional Verification of HDL Models[M]. New York: Kluwer Academic Publisher, 2000:155 - 268.

[3] 华为技术有限公司高端以太网芯片项目组. 一种 CAL systemC RM 验证 ASIC 的方法[M]. 上海:SNUG China presentations, 2003.

[4] Haque·Khizar F A, Khan·Jonathan M. The Art of Verification With Vera[M]. California: Verification Central Fremont, 2001.

[5] Synopsys. Open Vera/Vera user guide[EB/OL]. 2004. www.synopsys.com.

(上接第 93 页)

4　总　结

分析了基于轻量级容器开发的优点,设计一个支持多种注入方式的 IoC 容器,使得运行其中的程序代码能够很灵活地集成到 J2EE 环境中,而不受环境的束缚。在一般情况下容器管理的对象不需要调用 J2EE 中的 API,不需要编写繁琐的 JNDI,提高了代码的重用性。

参考文献:

[1] Martinfowler. Inversion of Control Containers and the Dependency Injection pattern[EB/OL]. 2003. http://www.martinfowler.com/articles/injection.html.

[2] Johnson R. Introducing the Spring Framework[EB/OL]. 2005. http://www.theserverside.com/articles/article.tss?l=Spring Framework.

[3] Tate B, Gehtland J. Better, Faster, Lighter Java[M]. CA:O'Reilly, 2003.

[4] Gulzar N, Kartik. Practical J2EE Application Architecture [M]. [s.l.]:The McGraw - Hill Companies, 2003.

[5] Sriganesh R P. Mastering Enterprise JavaBeans 3.0[M]. [s.l.]:Wiley Publishing, Inc, 2006.