14 离开了Spring，IoC容器还可以怎么做？

更新时间：2020-06-10 17:20:24

> 低头要有勇气，抬头要有底气。——韩寒

背景

众所周知，做 Java 开发的几乎没有人没听说过 Spring 框架，它作为一个轻量级的开源框架不仅给我们的开发工作带来了许多便利，同时也为众多开源框架的研究提供了不可或缺的指导思想。

提到 Spring，很多人不由自主的就想到了 IoC，想到了著名的好莱坞法则/好莱坞原则：

Hollywood principle:“Don’t call me; I’ll call you.” (don’t call us, we’ll call you)

IoC（Inversion of Control） 控制反转，即“不用打电话过来，我们会打给你”。

IoC 容器负责实例化、定位、配置应用程序中的对象及建立这些对象间的依赖。应用程序无需在代码中 new 相关的对象，应用程序由 IoC 容器进行组装。

还有人会说，AOP 面向切面编程，Spring 的两大灵魂。那我们就看看这两大王牌可以做些什么吧？

Spring 实例

- 依赖管理实例

pojo：

```
package com.davidwang456.test;
public class Employee {
    private int age;
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

/resource 下配置文件 springBeans.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/sp
   <bean id="bean1" class="com.davidwang456.test.Employee">
     <property name="name" value="Rakesh" />
     <property name="age" value="20" />
   </bean>
</beans>
```

容器测试类：

```
package com.davidwang456.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ApplicationContextTest {
    public static void main(String[] args) {
        @SuppressWarnings("resource")
        ApplicationContext context = new ClassPathXmlApplicationContext("SpringBeans.xml");
        Employee emp = (Employee) context.getBean("bean1");
        System.out.println(emp.getAge());
    }
}
```

输出：

> 20

- 切面编程实例

  新增 CustomBeanPostProcessor：

```
package com.davidwang456.test;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;

public class CustomBeanPostProcessor implements BeanPostProcessor
{
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException
    {
        System.out.println("Called postProcessBeforeInitialization() for :" + beanName);
        return bean;
    }

    public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException
    {
        System.out.println("Called postProcessAfterInitialization() for :" + beanName);
        return bean;
    }
}
```

  修改配置文件 springBeans.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-bean
  <bean id="bean1" class="com.davidwang456.test.Employee">
    <property name="name" value="Rakesh" />
    <property name="age" value="20" />
  </bean>
 <bean id="customBeanPostProcessor" class="com.davidwang456.test.CustomBeanPostProcessor" />
</beans>
```

输出：

> Called postProcessBeforeInitialization() for :bean1
>
> Called postProcessAfterInitialization() for :bean1
>
> 20

我不想用 Spring IoC 容器这样重的框架，还有其它选择吗

Spring 框架的依赖注入是家喻户晓的，但是在实际的开发中我们想使用便捷的依赖注入功能，但是又不想引入 Spring 框架的复杂性，该怎么办呢？

Guice IoC 容器

Guice 是由 Google 大牛 Bob lee 开发的一款绝对轻量级的 Java IoC 容器。其优势在于：

速度快，号称比 Spring 快 100 倍；

无外部配置（如需要使用外部可以可以选用 Guice 的扩展包），完全基于 annotation 特性，支持重构，代码静态检查；

简单，快速，基本没有学习成本。

Guice 常用术语：

Guice：整个框架的门面；

Injector：一个依赖的管理上下文；

Binder：一个接口和实现的绑定；

Module：一组 Binder；

Provider：bean 的提供者；

Key：Binder 中对应一个 Provider；

Scope：Provider 的作用域；

Stage：运行方式（为了不同的要求）。

依赖管理实例：

```
import com.google.inject.AbstractModule;
import com.google.inject.Guice;
import com.google.inject.Inject;
import com.google.inject.Injector;

public class GuiceTester {
   public static void main(String[] args) {
      Injector injector = Guice.createInjector(new TextEditorModule());
      TextEditor editor = injector.getInstance(TextEditor.class);
      editor.makeSpellCheck();
   }
}
class TextEditor {
   private SpellChecker spellChecker;
   @Inject

   public TextEditor(SpellChecker spellChecker) {
      this.spellChecker = spellChecker;
   }
   public void makeSpellCheck() {
      spellChecker.checkSpelling();
   }
}

//Binding Module
class TextEditorModule extends AbstractModule {
   @Override

   protected void configure() {
      bind(SpellChecker.class).to(SpellCheckerImpl.class);
      bind(SpellCheckerImpl.class).to(WinWordSpellCheckerImpl.class);
   }
}

//spell checker interface
interface SpellChecker {
   public void checkSpelling();
}
```

业务逻辑实现：

```
//spell checker implementation
class SpellCheckerImpl implements SpellChecker {
   @Override

   public void checkSpelling() {
      System.out.println("Inside checkSpelling." );
   }
}

//subclass of SpellCheckerImpl
class WinWordSpellCheckerImpl extends SpellCheckerImpl {
   @Override

   public void checkSpelling() {
      System.out.println("Inside WinWordSpellCheckerImpl.checkSpelling." );
   }
}
```

输出结果：

> Inside checkSpelling.

- 切面编程实例：

新增切面类：

```
package com.david456.test;


import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;


public class MessagePrinter implements MethodInterceptor{
    @Override
    public Object invoke(MethodInvocation invocation) throws Throwable {
        System.out.println("print message before processed ");
        Object ob=invocation.proceed();
         System.out.println("print message after processed ");
        return ob;
    }
}
```

新增切面 module：

```
package com.david456.test;

import com.google.inject.AbstractModule;
import com.google.inject.matcher.Matchers;

public class AOPModule extends AbstractModule {

    @Override
    protected void configure() {
        bindInterceptor(
            Matchers.any(),
            Matchers.annotatedWith(MessageSentLoggable.class),
            new MessagePrinter()
        );
    }
}
```

测试类新增切面：

```
import com.google.inject.Guice;
import com.google.inject.Injector;

public class GuiceTester {
   public static void main(String[] args) {
      Injector injector = Guice.createInjector(new TextEditorModule(),new AOPModule());
        /*
         * Map binds= injector.getAllBindings(); Iterator
         * it=binds.entrySet().iterator(); while (it.hasNext()) {
         * System.out.println(it.next()); }
         */
      TextEditor editor = injector.getInstance(TextEditor.class);
       editor.makeSpellCheck();
   }
}
```

输出结果：

> print message before processed
>
> Inside checkSpelling.
>
> print message after processed

总结

作为 IoC 容器，Guice 和 Spring 各有所长，Guice 更适合与嵌入式或者高性能但项目简单方案，如 OSGI 容器，Spring 更适合大型项目组织。

}