为什么要使用 SpringIOC?

作者：valarchie
原文链接：https://ld246.com/article/1568140057921
来源网站：链滴

思考

Spring已经占据我们Java开发框架中的半壁江山了，从一开始工作我们就在使用Spring。但是到底为么要用Spring，可能很多人都没有去思考过这个问题？许多人可能也疲于应对需求，无暇思考这种看理所当然的问题。那今天，我们就好好来讨论一下究竟为什么要使用Spring IOC?

逆向思考

假设在最初没有Spring IOC这种框架的时候，我们采用传统MVC的方式来开发一段常见的用户逻辑。

用户DAO

```
public class UserDAO {

    private String database;

    public UserDAO(String dataBase) {
        this.database = dataBase;
    }
    public void doSomething() {
        System.out.println("保存用户！ ");
    }

}
```

用户Service

```
public class UserService {

    private UserDAO dao;

    public UserService(UserDAO dao) {
        this.dao = dao;
    }
    public void doSomething() {
        dao.doSomething();
    }

}
```

用户Controller

```
public class Controller {

    public UserService service;

    public Controller(UserService userService) {
        this.service = userService;
    }

    public void doSomething() {
        service.doSomething();
    }
```

原文链接：为什么要使用 SpringIOC?

```
}
```

接下来我们就必须手动一个一个创建对象，并将dao、service、controller依次组装起来，然后才能用。

```
    public static void main(String[] args) {

        UserDAO dao = new UserDAO("mysql");
        UserService service = new UserService(dao);
        Controller controller = new Controller(service);

        controller.doSomething();

    }
```

分析一下这种做法的弊端有哪些呢?

1. 在生成Controller的地方我们都必须先创建dao再创建service最后再创建Controller,这么一条繁琐创建过程。

2. 在这三层结构当中，上层都需要知道下层是如何创建的，上层必须自己创建下层，这样就形成了紧耦合。为什么业务程序员在写业务的时候却需要知道数据库的密码并自己创建dao呢？不仅如此，当果dao的数据库密码变化了，在每一处生成Controller的地方都需要进行修改。

3. 通过new关键字生成了具体的对象，这是一种硬编码的方式，违反了面向接口编程的原则。当有一我们从Hibernate更换到Mybatis的时候,在每一处new DAO的地方，我们都需要进行更换。

4. 我们频繁的创建对象，浪费了资源。

这时候我们再来看看如果用SpringIOC的情况，刚才的代码变成如下。

```
public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("classpath:applicatio
.xml");
        Controller controller = (Controller) context.getBean("controller");
        controller.doSomething();

    }
```

很明显，使用IOC之后，我们只管向容器索取所需的Bean即可。IOC便解决了以下的痛点:

1. Bean之间的解耦，这种解耦体现在我们没有在代码中去硬编码bean之间的依赖。（不通过new操依次构建对象，由springIOC内部帮助我们实现了依赖注入）。一方面，IOC容器将通过new对象设依赖的方式转变为运行期动态的进行设置依赖。

2. IOC容器天然地给我们提供了单例。

3. 当需要更换dao的时候，我们只需要在配置文件中更换dao的实现类，完全不会破坏到之前的代码。

4. 上层现在不需要知道下层是如何创建的。

通过这个例子可能读者有点若有所思了，那我们再来看一下IOC的定义是什么。

定义

控制反转（Inversion of Control，缩写为IoC），是面向对象编程中的一种设计原则，可以用来减低算机代码之间的耦合度。其中最常见的方式叫做依赖注入（Dependency Injection，简称DI），还一种方式叫“依赖查找”（Dependency Lookup）。通过控制反转，对象在被创建的时候，由一个控系统内所有对象的外界实体将其所依赖的对象的引用传递给它。也可以说，依赖被注入到对象中。

解读

控制反转，控制体现在什么地方？控制就体现在我们一开始的例子，上层需要去控制new下层对象。反转意味着什么呢？

反转意味着我们把对象创建的控制权交出去，交给谁呢？交给IOC容器，由IOC容器负责创建特定对，并填充到各个声明需要特定对象的地方。同学们可能会对IOC和DI觉得很绕，其实IOC相当于接口它规定了要实现什么？而依赖注入（DI）就是具体的实现，它实现了IOC想要的效果。IOC的思想很地体现了面向对象设计法则之一——好莱坞法则：“别找我们，我们找你”；即由IoC容器帮对象找应的依赖对象并注入，而不是由对象主动去找。

如何减低耦合？我们举个例子，当在IDEA开发环境中我们添加插件的时候，却需要去知道该插件需要么依赖，要怎么才能成功开启插件？是不是对用户相当不友好呢？既然我们要使用插件，意味着我们插件之间是耦合的，无法避免。但是如果我们还需要知道插件需要的依赖和开启插件的步骤，说明我和插件之间的耦合关系更强烈了。所以SpringIOC的定义当中写道“降低耦合”而非“消除耦合”，只要耦合度降低的话，就有利于代码的维护和扩展。思考一下？是不是跟Maven的机制很类似呢？在aven中声明依赖的话，Maven的自动将该依赖所需的其他依赖递归的寻找。你可以把你自己想象成个对象的设计师，你设计了一张对象内部构造的图纸，交给SpringIOC，它会自动地根据这份图纸生这个对象。

IOC容器实现了开发中对象粒度的一种组件化，将每个对象当做组件一样放进容器当中。而每个组件是可插拔，这种是最佳的对象解耦方式。一方面通过将对象之间的耦合关系从编译期推迟到运行期。旦有需要这个对象的话，就直接使用，客户程序员完全不需要知道这个对象的来龙去脉。并且IOC容对Bean的统一管理使得AOP的实现更加的方便。这样一来，我们客户程序员就可以更专注在业务代的编写上。

笔者水平有限，希望这篇文章对读者有帮助。如有错误，烦请大家指出。