**Spring 依赖注入**

Spring框架的核心功能之一就是通过依赖注入的方式来管理Bean之间的依赖关系。

**依赖注入**

每个基于应用程序的 java 都有几个对象，由这些对象一起工作来呈现出终端用户所看到的工作的应用程序。当编写一个复杂的 Java 应用程序时，应用程序类应该尽可能独立于其他 Java 类来增加这些类重用的可能性，并且在做单元测试时，测试独立于其他类的独立性。依赖注入（或有时称为布线）有助于把这些类粘合在一起，同时保持他们独立。

假设你有一个包含文本编辑器组件的应用程序，并且你想要提供拼写检查。标准代码看起来是这样的：

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor() {

spellChecker = new SpellChecker();

}

}

在这里我们所做的就是创建一个 TextEditor 和 SpellChecker 之间的依赖关系。而在控制反转IoC的场景中，我们会这样做：

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

在这里，TextEditor 不应该担心 SpellChecker 的实现。SpellChecker 将会独立实现，并且在 TextEditor 实例化的时候将提供给 TextEditor，整个过程是由 Spring 框架的控制。

在这里，我们已经从 TextEditor 中删除了全面控制，并且把它保存到其他地方（即 XML 配置文件），且依赖关系（即 SpellChecker 类）通过**类构造函数**被注入到 TextEditor 类中。因此，控制流通过依赖注入（DI）已经“反转”，因为你已经有效地委托依赖关系到一些外部系统。

依赖注入的**第二种方法**是通过 TextEditor 类的 **Setter 方法**，我们将创建 SpellChecker 实例，该实例将被用于调用 setter 方法来初始化 TextEditor 的属性。

因此，DI 主要有两种变体和下面的两个子章将结合实例涵盖它们：

|  |  |
| --- | --- |
| **序号** | **依赖注入类型 & 描述** |
| 1 | [Constructor-based dependency injection](https://www.w3cschool.cn/wkspring/t7n41mm7.html)  当容器调用带有多个参数的构造函数类时，实现基于构造函数的 DI，每个代表在其他类中的一个依赖关系。 |
| 2 | [Setter-based dependency injection](https://www.w3cschool.cn/wkspring/yqdx1mm5.html)  基于 setter 方法的 DI 是通过在调用无参数的构造函数或无参数的静态工厂方法实例化 bean 之后容器调用 beans 的 setter 方法来实现的。 |

你可以混合这两种方法，基于构造函数和基于 setter 方法的 DI，然而使用有强制性依存关系的构造函数和有可选依赖关系的 sette r是一个好的做法。

代码是 DI 原理的清洗机，当对象与它们的依赖关系被提供时，解耦效果更明显。对象不查找它的依赖关系，也不知道依赖关系的位置或类，而这一切都由 Spring 框架控制的。