Spring IoC（控制反转）

IoC 是 Inversion of Control 的简写，译为“控制反转”，它不是一门技术，而是一种设计思想，是一个重要的面向对象编程法则，能够指导我们如何设计出松耦合、更优良的程序。

Spring 通过 IoC 容器来管理所有 Java 对象的实例化和初始化，控制对象与对象之间的依赖关系。我们将由 IoC 容器管理的 Java 对象称为 Spring Bean，它与使用关键字 new 创建的 Java 对象没有任何区别。

IoC 容器是 Spring 框架中最重要的核心组件之一，它贯穿了 Spring 从诞生到成长的整个过程。

控制反转（IoC）

在传统的 Java 应用中，一个类想要调用另一个类中的属性或方法，通常会先在其代码中通过 new Object() 的方式将后者的对象创建出来，然后才能实现属性或方法的调用。为了方便理解和描述，我们可以将前者称为“调用者”，将后者称为“被调用者”。也就是说，调用者掌握着被调用者对象创建的控制权。

但在 Spring 应用中，Java 对象创建的控制权是掌握在 IoC 容器手里的，其大致步骤如下。

1. 开发人员通过 XML 配置文件、注解、Java 配置类等方式，对 Java 对象进行定义，例如在 XML 配置文件中使用 <bean> 标签、在 Java 类上使用 @Component 注解等。
2. Spring 启动时，IoC 容器会自动根据对象定义，将这些对象创建并管理起来。这些被 IoC 容器创建并管理的对象被称为 Spring Bean。
3. 当我们想要使用某个 Bean 时，可以直接从 IoC 容器中获取（例如通过 ApplicationContext 的 getBean() 方法），而不需要手动通过代码（例如 new Obejct() 的方式）创建。

IoC 带来的最大改变不是代码层面的，而是从思想层面上发生了“主从换位”的改变。原本调用者是主动的一方，它想要使用什么资源就会主动出击，自己创建；但在 Spring 应用中，IoC 容器掌握着主动权，调用者则变成了被动的一方，被动的等待 IoC 容器创建它所需要的对象（Bean）。

这个过程在职责层面发生了控制权的反转，把原本调用者通过代码实现的对象的创建，反转给 IoC 容器来帮忙实现，因此我们将这个过程称为 Spring 的“控制反转”。

依赖注入（DI）

在了解了 IoC 之后，我们还需要了解另外一个非常重要的概念：依赖注入。

依赖注入（Denpendency Injection，简写为 DI）是 Martin Fowler 在 2004 年在对“控制反转”进行解释时提出的。Martin Fowler 认为“控制反转”一词很晦涩，无法让人很直接的理解“到底是哪里反转了”，因此他建议使用“依赖注入”来代替“控制反转”。

在面向对象中，对象和对象之间是存在一种叫做“依赖”的关系。简单来说，依赖关系就是在一个对象中需要用到另外一个对象，即对象中存在一个属性，该属性是另外一个类的对象。

例如，有一个名为 B 的 Java 类，它的代码如下。

```
01. public class B {
02.     String bid;
03.     A a;
04. }
```

从代码可以看出，B 中存在一个 A 类型的对象属性 a，此时我们就可以说 B 的对象依赖于对象 a。而依赖注入就是就是基于这种“依赖关系”而产生的。

我们知道，控制反转核心思想就是由 Spring 负责对象的创建。在对象创建过程中，Spring 会自动根据依赖关系，将它依赖的对象注入到当前对象中，这就是所谓的“依赖注入”。

依赖注入本质上是 Spring Bean 属性注入的一种，只不过这个属性是一个对象属性而已。

IoC 的工作原理

在 Java 软件开发过程中，系统中的各个对象之间、各个模块之间、软件系统和硬件系统之间，或多或少都存在一定的耦合关系。

若一个系统的耦合度过高，那么就会造成难以维护的问题，但完全没有耦合的代码几乎无法完成任何工作，这是由于几乎所有的功能都需要代码之间相互协作、相互依赖才能完成。因此我们在程序设计时，所秉承的思想一般都是在不影响系统功能的前提下，最大限度的降低耦合度。

IoC 底层通过工厂模式、Java 的反射机制、XML 解析等技术，将代码的耦合度降低到最低限度，其主要步骤如下。

1. 在配置文件（例如 Bean.xml）中，对各个对象以及它们之间的依赖关系进行配置；
2. 我们可以把 IoC 容器当做一个工厂，这个工厂的产品就是 Spring Bean；
3. 容器启动时会加载并解析这些配置文件，得到对象的基本信息以及它们之间的依赖关系；
4. IoC 利用 Java 的反射机制，根据类名生成相应的对象（即 Spring Bean），并根据依赖关系将这个对象注入到依赖它的对象中。

由于对象的基本信息、对象之间的依赖关系都是在配置文件中定义的，并没有在代码中紧密耦合，因此即使对象发生改变，我们也只需要在配置文件中进行修改即可，而无须对 Java 代码进行修改，这就是 Spring IoC 实现解耦的原理。

IoC 容器的两种实现

IoC 思想基于 IoC 容器实现的，IoC 容器底层其实就是一个 Bean 工厂。Spring 框架为我们提供了两种不同类型 IoC 容器，它们分别是 BeanFactory 和 ApplicationContext。

BeanFactory

BeanFactory 是 IoC 容器的基本实现，也是 Spring 提供的最简单的 IoC 容器，它提供了 IoC 容器最基本的功能，由 org.springframework.beans.factory.BeanFactory 接口定义。

BeanFactory 采用懒加载（lazy-load）机制，容器在加载配置文件时并不会立刻创建 Java 对象，只有程序中获取（使用）这个对对象时才会创建。

示例 1

下面我们通过一个实例演示，来演示下 BeanFactory 的使用。

1. 在 HelloSpring 项目中，将 MainApp 的代码修改为使用 BeanFactory 获取 HelloWorld 的对象，具体代码如下。

```
01. public static void main(String[] args) {
02.     BeanFactory context = new ClassPathXmlApplicationContext("Beans.xml");
03.     HelloWorld obj = context.getBean("helloWorld", HelloWorld.class);
04.     obj.getMessage();
05. }
```

2. 运行 MainApp.java，控制台输出如下。

```
message : Hello World!
```

> 注意：BeanFactory 是 Spring 内部使用接口，通常情况下不提供给开发人员使用。

ApplicationContext

ApplicationContext 是 BeanFactory 接口的子接口，是对 BeanFactory 的扩展。ApplicationContext 在 BeanFactory 的基础上增加了许多企业级的功能，例如 AOP（面向切面编程）、国际化、事务支持等。

ApplicationContext 接口有两个常用的实现类，具体如下表。

实现类	描述	示例代码
ClassPathXmlApplicationContext	加载类路径 ClassPath 下指定的 XML 配置文件，并完成 ApplicationContext 的实例化工作	ApplicationContext applicationContext = new ClassPathXmlApplicationContext(String configLocation);
FileSystemXmlApplicationContext	加载指定的文件系统路径中指定的 XML 配置	ApplicationContext applicationContext = new

	文件，并完成 ApplicationContext 的实例化工作	FileSystemXmlApplicationContext(String configLocation);

> 在上表的示例代码中，参数 configLocation 用于指定 Spring 配置文件的名称和位置，如 Beans.xml。

示例 2

下面我们就通过一个实例，来演示 ApplicationContext 的使用。

1. 修改 HelloSpring 项目 MainApp 类中 main() 方法的代码，具体代码如下。

```
01. public static void main(String[] args) {
02.     //使用 FileSystemXmlApplicationContext 加载指定路径下的配置文件 Bean.xml
03.     BeanFactory context = new FileSystemXmlApplicationContext("D:\\eclipe
workspace\\spring workspace\\HelloSpring\\src\\Beans.xml");
04.     HelloWorld obj = context.getBean("helloWorld", HelloWorld.class);
05.     obj.getMessage();
06. }
```

2. 运行 MainApp.java，控制台输出如下。

```
message : Hello World!
```