

理解 Spring IoC

IoC 全称为 **Inversion of Control**，翻译为“控制反转”，它还有一个别名为 DI (**Dependency Injection**)，即依赖注入。

如何理解“控制反转”好呢？理解好它的关键在于我们需要回答如下四个问题：

1. 谁控制谁
2. 控制什么
3. 为何是反转
4. 哪些方面反转了

在回答这四个问题之前，我们先看 IOC 的定义：

所谓 IOC，就是由 Spring IOC 容器来负责对象的生命周期和对象之间的关系
上面这句话是整个 IoC 理论的核心。如何来理解这句话？我们引用一个例子来走阐述（看完该例子上面四个问题也就不是问题了）。

已找女朋友为例（对于程序猿来说这个值得探究的问题）。一般情况下我们是如何来找女朋友的呢？首先我们需要根据自己的需求（漂亮、身材好、性格好）找一个妹子，然后到处打听她的兴趣爱好、微信、电话号码，然后各种投其所好送其所要，最后追到手。如下：

```
/**
 * 年轻小伙子
 */
public class YoungMan {
    private BeautifulGirl beautifulGirl;

    YoungMan(){
        // 可能你比较牛逼，指腹为婚
        // beautifulGirl = new BeautifulGirl();
    }

    public void setBeautifulGirl(BeautifulGirl beautifulGirl) {
        this.beautifulGirl = beautifulGirl;
    }

    public static void main(String[] args){
        YoungMan you = new YoungMan();
        BeautifulGirl beautifulGirl = new BeautifulGirl("你的各种条件");
        beautifulGirl.setxxx("各种投其所好");

        // 然后你有女票了
        you.setBeautifulGirl(beautifulGirl);
    }
}
```

这就是我们通常做事的方式，如果我们需要某个对象，一般都是采用这种直接创建的方式(`new BeautifulGirl()`)，这个过程复杂而又繁琐，而且我们必须面对每个环节，同时使用完成之后我们还要负责销毁它，在这种情况下我们的对象与它所依赖的对象耦合在一起。

其实我们需要思考一个问题？我们每次用到自己依赖的对象真的需要自己去创建吗？我们知道，我们依赖对象其实并不是依赖该对象本身，而是依赖它所提供的服务，只要在我们需要它的时候，它能够及时提供服务即可，至于它是我们主动去创建的还是别人送给我们的，其实并不是那么重要。再说了，相比于自己千辛万苦去创建它还要管理、善后而言，直接有人送过来是不是显得更加好呢？

这个给我们送东西的“人”就是 IoC，在上面的例子中，它就相当于一个婚介公司，作为一个婚介公司它管理着很多男男女女的资料，当我们需要一个女朋友的时候，直接跟婚介公司提出我们的需求，婚介公司则会根据我们的需求提供一个妹子给我们，我们只需要负责谈恋爱，生猴子就行了。你看，这样是不是很简单明了。

诚然，作为婚介公司的 IoC 帮我们省略了找女朋友的繁杂过程，将原来的主动寻找变成了现在的被动接受（符合我们的要求），更加简洁轻便。你想啊，原来你还得鞍马前后，各种巴结，什么东西都需要自己去亲力亲为，现在好了，直接有人把现成的送过来，多么美妙的事情啊。所以，简单点说，IoC 的理念就是**让别人为你服务**

在没有引入 IoC 的时候，被注入的对象直接依赖于被依赖的对象，有了 IoC 后，两者及其他的关系都是通过 Ioc Service Provider 来统一管理维护的。被注入的对象需要什么，直接跟 IoC Service Provider 打声招呼，后者就会把相应的被依赖对象注入到被注入的对象中，从而达到 IOC Service Provider 为被注入对象服务的目的。**所以 IoC 就是这么简单！原来是需要什么东西自己去拿，现在是需要什么东西让别人（IOC Service Provider）送过来**

现在在看上面那四个问题，答案就显得非常明显了：

- 1. 谁控制谁：**在传统的开发模式下，我们都是采用直接 `new` 一个对象的方式来创建对象，也就是说你依赖的对象直接由你自己控制，但是有了 IOC 容器后，则直接由 IoC 容器来控制。所以“谁控制谁”，当然是 IoC 容器控制对象。
- 2. 控制什么：**控制对象。
- 3. 为何是反转：**没有 IoC 的时候我们都是在自己对象中主动去创建被依赖的对象，这是正转。但是有了 IoC 后，所依赖的对象直接由 IoC 容器创建后注入到被注入的对象中，依赖的对象由原来的主动获取变成被动接受，所以是反转。
- 4. 哪些方面反转了：**所依赖对象的获取被反转了。

妹子有了，但是如何拥有妹子呢？这也是一门学问。

1. 可能你比较牛逼，刚刚出生的时候就指腹为婚了。

2. 大多数情况我们还是会考虑自己想要什么样的妹子，所以还是需要向婚介公司打招呼的。

3. 还有一种情况就是，你根本就不知道自己想要什么样的妹子，直接跟婚介公司说，我就要一个这样的妹子。

所以，IOC Service Provider 为被注入对象提供被依赖对象也有如下几种方式：

1. 构造方法注入
2. setter方法注入
3. 接口注入

构造器注入

构造器注入，顾名思义就是被注入的对象通过在其构造方法中声明依赖对象的参数列表，让外部知道它需要哪些依赖对象。

```
YoungMan(BeautifulGirl beautifulGirl){  
    this.beautifulGirl = beautifulGirl;  
}
```

构造器注入方式比较直观，对象构造完毕后就可以直接使用，这就好比你出生你家里就给你指定了你媳妇。

setter 方法注入

对于 JavaBean 对象而言，我们一般都是通过 getter 和 setter 方法来访问和设置对象的属性。所以，当前对象只需要为其所依赖的对象提供相对应的 setter 方法，就可以通过该方法将相应的依赖对象设置到被注入对象中。如下：

```
public class YoungMan {  
    private BeautifulGirl beautifulGirl;  
  
    public void setBeautifulGirl(BeautifulGirl beautifulGirl) {  
        this.beautifulGirl = beautifulGirl;  
    }  
}
```

相比于构造器注入，setter 方式注入会显得比较宽松灵活些，它可以在任何时候进行注入（当然是在使用依赖对象之前），这就好比你可以先把自己想要的妹子想好了，然后再跟婚介公司打招呼，你可以要林志玲款式的，赵丽颖款式的，甚至凤姐哪款的，随意性较强。

接口方式注入

接口方式注入显得比较霸道，因为它需要被依赖的对象实现不必要的接口，带有侵入性。一般都不推荐这种方式。