链滴

IOC 学习笔记二

作者：zyjImmortal
原文链接：https://ld246.com/article/1588494190806
来源网站：链滴

上篇文慢慢了解IOC是怎么来的一我们介绍了如何更方便的创建对象，最后一反射模式的使用消除了量的new的代码，使得程序变得更加稳定，但是反射是有问题的，因为在使用反射时，他是要动态的创建对象，所以他的性能会比较低，尤其是频繁的使用反射，系统的性能会下降特别多。

那有没有更好的方式去管理对象的创建与传递呢，必须的有，就是接下来要探讨的IOC和DI了。

什么是IOC(控制反转)

回忆一下上文，用户输入要创建对象的全限定类名，传给我们的工厂，然后通过反射去创建一个对象返回给用户，这是一种正向的思维，就是说我们需要什么，然后就去实例化什么对象，调用对象的方。虽然实现了开闭原则，把对象实例化的操作封装起来了，这种方式在使用起来并不是特别的方便，什么呢?

```
public class Test {

    public static void main(String[] args) throws Exception{
        String name = "Irelia";
        Skill hero = HeroFactory.getHeroByReflect(name);
        hero.r();
    }
}
```

因为这里还是要引入factory的，有什么方法连引入factory的都不用引入就能够获取到对象呢，那这方式是不是会更加的方便呢?

IOC它是一种反向的思维，在我们需要某个对象的地方，由一个容器来给我们实例化对象并赋值给特的引用，把本来由程序控制的对象的管理交给了容器，这就是控制反转。

```
public class Test {
    private Skill hero; // 注意这里Skill是一个接口
    public void print(){
```

```
        this.hero.r();
    }
}
```

比如说就有这么超级牛叉的容器，我们不再需要手动new或者调用工厂类去获取对象了，而是容器在到这里需要这么一个对象，就主动给放进来，那这样的代码就会更加的稳定了。

为什么引入容器后会更加稳定

我们知道面向对象编程，其实就是使用一个一个类构成的系统，那么要想使得系统变得稳定，就需要证各个类是能够正常运转的。

在上篇文章在最开始，我们在一个类里需要操作英雄的时候，需要手动new一个具体的英雄类，那这操作类和具体的英雄类，就具备了依赖关系，如果英雄类又依赖于其他的类，那一个一个的类都是紧耦合在一起的，如果某个类出现了问题，那整个系统就会出现问题。

当容器出现以后，由容器来管理类与类之间的强依赖关系，类与类之间变成了松耦合的关系，那程序稳定就会得到大大的提高。

DI(Dependency Inject)

前面我们提到容器会在需要的地方给我们把实例化好的对象放进来，其实这个过程就是依赖注入，把容器把程序依赖的对象注入进来。

```
public class Test {
    private Skill hero; // 注意这里Skill是一个接口
    public void print(){
        this.hero.r();
    }
}
```

注入的方式有跟多种，用的最多的是属性注入和构造注入，下篇文章，我们结合spring来说具体的IO和DI注入方式。