在方法签名中使用控制反转（IoC）[1] PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/167/2021_2022__E5_9C_A8_E6_96_B9_E6_B3_95_E7_c104_167020.htm 控制反转（IoC）和依赖项注入（DI）是两种引起极大关注的模式（参见 参考资料）。它们主要用在所谓的 IoC 容器中，这些容器以其他组件的形式将依赖项注入到一个组件中。然而，这两种模式并未定义这些依赖项组件方法的设计方式。在经典的设计中，这些方法中的值对象或数据传输对象用作方法参数并在需要复杂对象时返回值。 本文向您展示还可以对方法签名使用 IoC，从而使方法与值对象解耦。为此，要把方法签名中的值对象替换成接口。我会介绍该方法的一些应用场景。我经常使用这种模式，并发现借助它可以更好地分离组件之间的关注点。并且在运行时，它能减少对象创建和复制工作。 使用值对象作为方法参数 关于 IoC 已经有很多描述，所以此处只阐述其总体原则：组件将其使用的组件配置、本地化和生命周期方面“ 外包 ”出去。例如，数据访问 bean 直接从某处“ 获取 ”JDBC 连接并简单地使用该连接，而不是寻找一个 JDBC 数据源（配置和本地化），也许还要自己处理连接池（生命周期）。在 IoC 设置中，这些方面通常由一个 IoC 容器处理，比如，该容器通过调用 setter 方法将这些依赖项注入组件。 IoC 主要处理组件的生命周期。本文并不关注组件，而是关注组件提供的操作的方法参数。 请看下图，这是典型的组件设置的依赖关系图，其中有两个依赖关系并使用了方法参数（参见图 1）。这些方法参数定义成值对象，即不含逻辑只含数据值的对象。 图 1. 使用值对象作为方法参数的组件

依赖关系图 在图 1 中，Component1 依赖于 Component2 和 Component3 （根据 IoC），并分别调用 method2 和 method3 。如果 Component1 直接“ 了解 ”Component2 或 Component3 或只使用 Component2 和 Component3 实现的接口，那么这与我们的讨论不相关。然而，通常方法参数基本都是对象而不是接口。 在这个设置中，当 Component1 调用 method2 时，它必须实例化 Value Object 2 并为其赋值。同样，如果 Component1 调用 method3，它必须实例化 Value Object 3 并为其赋值。 现在假设 Component1 需要用相同的输入数据调用 method2 和 method3 来获取不同的输出数据。例如，Component1 可以是订单准备组件，method2 可以是决定交货期的方法，method3 可以是决定价格的方法。这两种方法需要相同的输入并提供不同的输出。 在本例中，使用值对象作为方法参数需要 Component1 为每个方法调用创建值对象，并主动地将所需值复制到值对象中。同样，必须实例化这每个值对象，尽管这已经不像在 Java&#8482. 早期的版本中那么耗费资源，但仍然需要资源。这些行为都降低了性能。下面几节将介绍如何优化性能。 使用接口改善这种情况 目标是防止在不同的值对象间复制值。为此，可以将方法参数定义为接口。这样，只要对象实现该接口，调用组件就可以将其想使用的任何对象用作方法参数。 图 2 显示了新的依赖关系： 图 2. 使用接口作为方法参数的组件依赖关系图 