J2EE基础：浅谈依赖注入实现的方法Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_J2EE_E5_9F_BA_E7_A1_80_c104_644493.htm 1接口注入 我们常常借助接口来将调用者与实现者分离。如： public class ClassA { private InterfaceB clzB. public doSomething() { Ojbect obj = Class.forName(Config.BImplementation).newInstance(). clzB = (InterfaceB)obj. clzB.doIt() } …… } 上面的代码中，ClassA依赖于InterfaceB的实现，如何获得InterfaceB实现类的实例？传统的方法是在代码中创建InterfaceB实现类的实例，并将起赋予clzB. 而这样一来，ClassA在编译期即依赖于InterfaceB的实现。为了将调用者与实现者在编译期分离，于是有了上面的代码，我们根据预先在配置文件中设定的实现类的类名（Config.BImplementation），动态加载实现类，并通过InterfaceB强制转型后为ClassA所用。这就是接口注入的一个最原始的雏形。 而对于一个1型IOC容器而言，加载接口实现并创建其实例的工作由容器完成。 如下面这个类： public class ClassA { private InterfaceB clzB. public Object doSomething(InterfaceB b) { clzB = b. return clzB.doIt(). } …… } 在运行期，InterfaceB实例将由容器提供。 1型IOC发展较早（有意或无意），在实际中得到了普遍应用，即使在IOC的概念尚未确立时，这样的方法也已经频繁出现在我们的代码中。 下面的代码大家应该非常熟悉： public class MyServlet extends HttpServlet { public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { …… } } 在运行期动态注入。 另，Apache Avalon

是一个较为典型的1型IOC容器。 2 设值注入 在各种类型的依赖注入模式中，设值注入模式在实际开发中得到了最广泛的应用（其中很大一部分得力于Spring框架的影响）。 在笔者看来，基于设置模式的依赖注入机制更加直观、也更加自然。Quick Start中的示例，就是典型的设置注入，即通过类的setter方法完成依赖关系的设置。 3 构造子注入 构造子注入，即通过构造函数完成依赖关系的设定，如： public class DIByConstructor { private final DataSource dataSource. private final String message. public DIByConstructor(DataSource ds, String msg) { this.dataSource = ds. this.message = msg. } …… } 可以看到，在3类型的依赖注入机制中，依赖关系是通过类构造函数建立，容器通过调用类的构造方法，将其所需的依赖关系注入其中。 PicoContainer（另一种实现了依赖注入模式的轻量级容器）首先实现了3类型的依赖注入模式。 几种依赖注入模式的对比总结 接口注入模式因为历史较为悠久，在很多容器中都已经得到应用。但由于其在灵活性、易用性上不如其他两种注入模式，因而在IOC的专题世界内并不被看好。 2和3型的依赖注入实现则是目前主流的IOC实现模式。这两种实现方式各有特点，也各具优势（一句经典废话J）。 2 设值注入的优势 1. 对于习惯了传统JavaBean开发的程序员而言，通过setter方法设定依赖关系显得更加直观，更加自然。 2. 如果依赖关系（或继承关系）较为复杂，那么3模式的构造函数也会相当庞大（我们需要在构造函数中设定所有依赖关系），此时2模式往往更为简洁。 3. 对于某些第三方类库而言，可能要求我们的组件必须提供一个默认的构造函数（如Struts中的Action），此时3类型的依赖注入机制就体现出其局限性，

难以完成我们期望的功 能。 3 构造子注入的优势： 1. “ 在构造期即创建一个完整、合法的对象 ”，对于这条Java设计原则， 3无疑是最好的响应者。 2. 避免了繁琐的setter方法的编写，所有依赖关系均在构造函数中设定，依赖关系集中呈现，更加易读。 3. 由于没有setter方法，依赖关系在构造时由容器一次性设定，因此组件在被创建之后即处于相对 “ 不变 ” 的稳定状态，无需担心上层代码在调用过程中执行 setter方法对组件依赖关系产生破坏，特别是对于Singleton模式的组件而言，这可能对整个系统产生重大的影响。 4. 同样，由于关联关系仅在构造函数中表达，只有组件创建者需要关心组件内部的依赖关系。对调用者而言，组件中的依赖关系处于黑盒之中。对上层屏蔽不必要的信息，也为系统的层次清晰性提供了保证。 5. 通过构造子注入，意味着我们可以在构造函数中决定依赖关系的注入顺序，对于一个大量依赖外部服务的组件而言，依赖关系的获得顺序可能非常重要，比如某个依赖关系注入的 先决条件是组件的DataSource及相关资源已经被设定。 可见， 3和2模式各有千秋，而Spring、PicoContainer都对3和2类型的依赖注入机制提供了良好支持。这也就为我们提供了更多的选择余地。理论上，以3类型为主，辅之以2类型机制作为补充，可以达到最好的依赖注入效果，不过对于基于Spring Framework开发的应用而言，2使用更加广泛。