Spring面试问题集锦

2012/11/19 | 分类： [职业生涯](http://www.importnew.com/cat/career) | [1 条评论](http://www.importnew.com/1019.html#comments) | 标签： [SPRING](http://www.importnew.com/tag/spring), [面试](http://www.importnew.com/tag/%e9%9d%a2%e8%af%95)

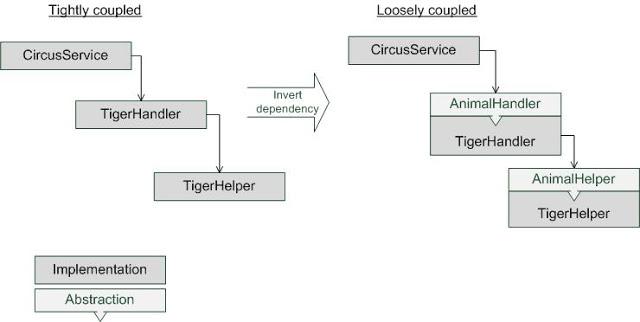
分享到：**37**

本文作者： [ImportNew](http://www.importnew.com/) - [郑雯](http://www.importnew.com/author/zhengwen) 未经许可，禁止转载！

**Q.** 对于依赖倒置原则（Dependency Inversion Principle，DIP），依赖注入（Dependency Injection，DI）和控制反转（Inversion of Control，IoC）容器，你是怎么理解的？

**A.**

* **依赖倒置原则（Dependency Inversion Principle, DIP）**。这个设计准则某种程度上和依赖注入模式有些关联。DIP的出发点是：在应用开发中，高层模块不应当直接依赖低层模块。DIP并不意味着依赖注入。这个准则并没有讲到高层模块如何知道调用哪个低层模块。不过这一点通过实现工厂模式接口可以间接获知，或者通过类似Spring框架、Pico容器、Guice或者Apache HiveMind之类的loC容器实现依赖注入从而得知高层模块调用的具体哪个低层模块。



DIP意味着：

* 高层模块不应当依赖低层模块，它们都应当依赖抽象。
* 抽象不应该依赖具体实现。具体实现应该依赖抽象。

应用这个准则后，高层模块并不直接同低层模块交互，而是通过一个抽象层来跟低层模块进行交互。这使得需求变更后增加的成本更加灵 活可控。这里有些实现DIP的示例代码片段。

首先定义抽象层：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | package principle\_dip2;  public interface AnimalHandler {      public abstract void handle( );  }  package principle\_dip2;  public interface AnimalHelper {      public abstract void help( );  } |

接着是依赖于抽象类而非具体实现的高层代码。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package principle\_dip2;  public class CircusService {      AnimalHandler handler;      public void setHandler(AnimalHandler handler) {          this.handler = handler;      }      public void showStarts( ) {          //code omitted for brevity          handler.handle( );      }  }  package principle\_dip2;  public class TigerHandler implements AnimalHandler{      AnimalHelper helper;      public void setHelper(AnimalHelper helper) {          this.helper = helper;      }      public void handle( ){          //...          helper.help( );          //...      }  }  package principle\_dip2;  public class TigerHelper implements AnimalHelper{      public void help( ){          //......      }  } |

* **依赖注入模式（Dependency Injection）**：在运行时将类的依赖注入到代码中。通过将依赖定义为接口，并将实现这个接口的实体类注入到主类的构造器中来实现这个模式。这允许程序员在不同的实现之间转换而不用去修改主类。依赖注入模式可以通过**单一责任原则**（**Single Responsibility Principle**）SRP来使得代码**高内聚（high cohesion）**，因为所依赖的通常都是完成独立的功能的对象，例如，（通过DAO进行）数据存取或（通过Service和Delegate类实现）业务服务。
* **控制反转容器（Inversion of Control Container，IoC），**是一个支持依赖注入的容器。这种方式下，可以采用一个中心容器，例如Spring框架，Guice或者HiveMind，来定义哪个依赖应该使用哪个实体类。Ioc的松耦合性可以带来更多的灵活性，并且在程序运行时更容易去切换到正确的依赖对象上。控制反转模式的基本概念是，不去实际生成对象，而是去定义如何生成对象。不用直接在代码中将模块和服务硬编码在一起，而是在配置文件中描述哪个模块需要哪个服务。容器（例如Spring框架这个IoC容器）会负责将这两者绑定起来。应用IoC的时候，某对象所需的依赖会在创建的时候通过外部实体传入，这些外部实体用来协调系统中的不同对象。也就是说，依赖是被注入到对象中去的。因此，IoC就是关于一个对象如何获得其协作对象的引用的一种责任反转机制。

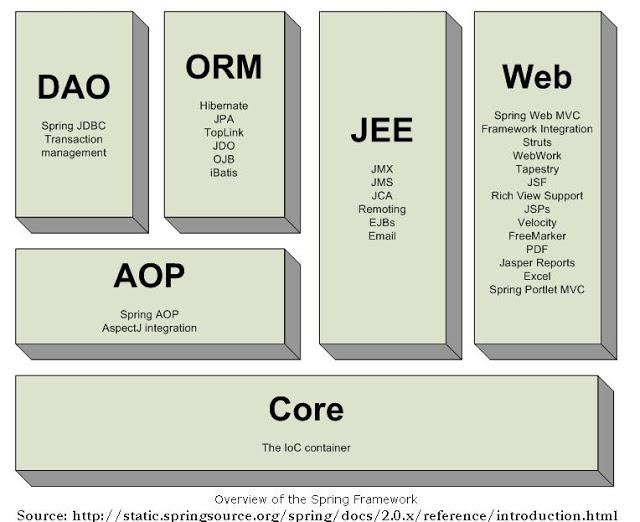
**DI**和**IoC**的真正强大之处在于，在运行时而非编译时绑定类间关系。例如，在Seam框架中，你可以对一个接口进行两种实现：真正的实现和模拟(mock)的实现，而在运行时根据某个属性、另一个文件存在与否或者某个优先值去决定真正调用哪一个实现。这尤其当你希望程序在不同场景下表现不同的行为时，这是非常好用的。DI和IoC的另外一个好处是，使得代码更容易进行单元测试。当然也有其他一些好处，例如，不用使用工厂或者单例模式就可以实现松耦合，其实现方法一致因此适合缺乏经验的程序员，等等。当然，享受这些好处是要付出代价的，例如系统复杂性会随之增加，另外在使用时也需要更加小心，不能因为这个技术受欢迎就滥用，而是在能够真正体现其优势的地方才去使用。

**注意**：上下文依赖注入（**Contexts and Dependency Injection**）是用来描述标准依赖注入的一个尝试。CDI是Java EE 6 stack的一部分，也就是说任何一个运行在Java EE 6兼容容器之上的应用都可以轻松使用CDI。Weld就是CDI的一个可参考的实现。

**Q. 以你的经验来看，为什么要选择使用Spring框架呢？**

**A.**

* Spring采用层次结构，有超过20个模块可供选用。这就意味着你可以根据需要自由取舍。Spring通过简单Java对象（Plain Old Java Object，POJO）编程简化了J2EE。在Spring中J2EE编程并没有什么特别的。POJO编程提供了代码的持续集成能力和易测性。



* Spring框架的核心功能是依赖注入（DI）。DI使得代码的单元测试更加方便、系统更好维护、代码也更加灵活。DI代码自身很容易测试，通过构建实现了应用所需的接口的“模拟”对象就可以进行功能的黑盒测试。DI代码也更容易复用，因为其“被依赖的”功能封装在在定义良好的接口中，允许其他对象根据需要将其插入到所需的对象中，这些对象是在其他应用平台中进行配置的。DI代码更加灵活，由于其天生的松耦合性，它允许程序员仅需考虑自己所需的接口和其他模块暴露出来的接口来就可以决定对象之间如何关联。
* Spring支持面向切面编程（Aspect Oriented Programming ，AOP），允许通过分离应用业务逻辑和系统服务从而进行内聚性的开发。AOP支持审计（auditing）、搜集性能和内存指标等功能。
* Spring还提供了许多实现基本功能的模板类，使得J2EE开发更加容易。例如，JdbcTemplate类和JDBC、JpaTemplate类和JPA，JmsTemplate类和JMS都可以很好地结合起来使用。RestTemplate类非常简洁，使用这个模板的代码的可读性和可维护性也都很好。
* 尽量把中间层代码从业务逻辑中剥离出来是很重要的。最好的远程调用方式就是利用Spring的远程接口调用，这个功能支持使用任何消息或者远程技术来完成远程调用。Apache Camel是一个强大的基于已知的包括Bean集成的企业级集成模式的开源集成框架。Apache Camel设计之初就是为了尽可能的和Spring框架能够很好的结合使用。
* Spring提供了声明性事务处理，工作调度，身份认证，成熟的MVC web框架以及和其他框架的集成，例如Hibernate、iBatis、JasperReports、JSF、Struts、Tapestry、Seam和Quartz job scheduler等等。
* Spring bean对象可以通过Terracotta在不同的JVM之间共享。这就允许使用已有的bean并在集群中共享 ，将Spring应用上下文事件变为分布式事件，还可以通过Spring JMX导出集群bean，使得Spring应用高可用、集群化。Spring还可以和其他集群应用方案集成起来，例如Oracle的Coherance。
* Spring倾向于使用未检查异常（unchecked exceptions）和减少不当try，catch和finally代码块（或者finally中的try/catch块）。像JpaTemplate 这样的Spring模板类会负责关闭或释放数据库连接，这避免了潜在的资源泄露问题并提高了代码的可读性。
* 在非Spring或者Guice这种DI框架中，工厂模式和单例模式可以用来提高代码的松耦合度。使用了Spring可以有效避免这些模式的滥用。

**Q．**根据你的项目经验，Spring框架的哪些地方是你不喜欢的？你认为Spring有缺陷吗？

**A．**

* Spring变得过于庞大和笨重。因此，我的建议是不要因为大家对Spring的赞誉有加而不加思考大肆使用其所有的功能。而是应该去使用Spring中真正对你的项目有用的功能。大多数情况下，从可维护性和不重复造车轮子这方面考虑，使用成熟的Spring这样的框架比自己从零开始构建一个类似的解决方案要好得多。例如，所有的Spring模板（jdbc，rest，jpa等等）都有如下优势：构建方式一致，所以你可以跳过这些常规步骤从而将精力放在更重要的业务逻辑上。
* Spring MVC并不一定是最好的web框架。还有一些其他的，例如，Struts 2，Wicket和JSF等。话虽如此，其实Spring也可以和这些框架（Struts，JSF等）很好的集成起来。
* XML文件过于臃肿庞大，这一点可以通过其他手段来得到改善。比如使用annotations，JavaConfig或者使用独立的XML配置文件.

**Q．**IoC中支持的依赖注入有哪些类型？

**A．**依赖注入有三种类型：

* 构造子注入（例如，Spring框架）：依赖是通过构造器参数提供的。
* 设值方法注入（例如，Spring框架）：依赖是通过JavaBeans属性注入的（ex：setter方法）
* 接口注入（例如，Avalon）：注入通过接口完成。

**Q．**你用过其他依赖注入框架吗？

**A．**用过Guice，Hivemind和Seam。