Spring 框架学习—控制反转（IOC）

原创 温柔狠角色 于 2016-10-28 16:04:57

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson创建。简单来说，Spring是一个分层的JavaSE/EEfull-stack(一站式) 轻量级开源框架，主要用于降低模块之间耦合度的框架，实际上Spring除了能够通过IoC降低模块之间的耦合度外还提供了其它功能。

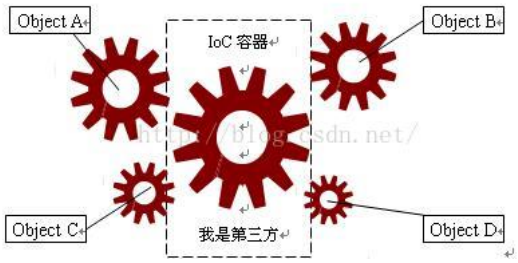
1、IoC的基础知识以及原理：

IoC理论的背景：在采用面向对象方法设计的软件系统中，底层实现都是由N个对象组成的，所有的对象通过彼此的合作，最终实现系统的业务逻辑。即软件系统中对象之间的耦合，对象A和对象B之间有关联，对象B又和对象C有依赖关系，这样对象和对象之间有着复杂的依赖关系，所以才有了控制反转这个理论。

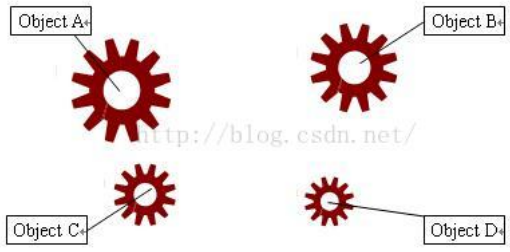
2、什么是控制反转(IoC)：

（1）IoC是Inversion of Control的缩写，有的翻译成“控制反转”，还有翻译成为“控制反向”或者“控制倒置”。

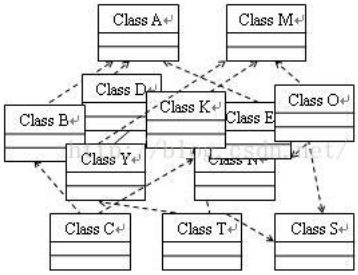
（2）1996年，Michael Mattson在一篇有关探讨面向对象框架的文章中，首先提出了IoC 这个概念。简单来说就是把复杂系统分解成相互合作的对象，这些对象类通过封装以后，内部实现对外部是透明的，从而降低了解决问题的复杂度，而且可以灵活地被重用和扩展。IoC理论提出的观点大体是这样的：借助于“第三方”实现具有依赖关系的对象之间的解耦，如下图所示：



即把各个对象类封装之后，通过IoC容器来关联这些对象类。这样对象与对象之间就通过IoC容器进行联系，但对象与对象之间并没有什么直接联系。如果去掉IoC容器后系统中的对象A与对象B就有了直接关系，如下图所示：



比如好多的对象类要关联起来的话，就会变得很复杂，如下图所示：



所以提出IoC控制反转是很有必要的。

3、为什么要把这种方式叫做控制反转呢?

（1） 软件系统在没有引入IoC容器之前，对象A依赖对象B，那么A对象在实例化或者运行到某一点的时候，自己必须主动创建对象B或者使用已经创建好的对象B，其中不管是创建还是使用已创建的对象B，控制权都在我们自己手上。

（2）如果软件系统引入了Ioc容器之后，对象A和对象B之间失去了直接联系，所以，当对象A实例化和运行时，如果需要对象B的话，IoC容器会主动创建一个对象B注入到对象A所需要的地方。

（3）通过前面的对比，可以看到对象A获得依赖对象B的过程，由主动行为变成了被动行为，即把创建对象交给了IoC容器处理，控制权颠倒过来了，这就是控制反转的由来！

4、IoC的别名：依赖注入（DI）

（1）2004年，Martin Fowler探讨了同一个问题，既然IoC是控制反转，那么到底是“哪些方面的控制被反转了呢？”，经过详细地分析和论证后，他得出了答案：“获得依赖对象的过程被反转了”。控制被反转之后，获得依赖对象的过程由自身管理对象变为由IoC容器主动注入。于是，他给“控制反转”取了一个更合适的名字叫做“依赖注入（Dependency Injection，DI）”。他的这个答案，实际上给出了实现IoC的方法：注入。

（2）所谓依赖注入，就是由IoC容器在运行期间，动态地将某种依赖关系注入到对象之中。

（3）所以，依赖注入（DI）和控制反转（IoC）是从不同的角度描述的同一件事情，就是指通过引入IoC容器，利用依赖关系注入的方式，实现对象之间的解耦。

5、使用IoC的好处：

（1）可维护性比较好，非常便于进行单元测试，便于调试程序和诊断故障。代码中的每一个Class都可以单独测试，彼此之间互不影响，只要保证自身的功能无误即可，这就是组件之间低耦合或者无耦合带来的好处。

（2）每个开发团队的成员都只需要关注自己要实现的业务逻辑，完全不用去关心其他人的工作进展，因为你的任务跟别人没有任何关系，你的任务可以单独测试，你的任务也不用依赖于别人的组件，再也不用扯不清责任了。所以，在一个大中型项目中，团队成员分工明确、责任明晰，很容易将一个大的任务划分为细小的任务，开发效率和产品质量必将得到大幅度的提高。

（3）可复用性好，我们可以把具有普遍性的常用组件独立出来，反复应用到项目中的其它部分，或者是其它项目，当然这也是面向对象的基本特征。显然，IoC更好地贯彻了这个原则，提高了模块的可复用性。符合接口标准的实现，都可以插接到支持此标准的模块中。

（4）IoC生成对象的方式转为外置方式，也就是把对象生成放在配置文件里进行定义，这样，当我们更换一个实现子类将会变得很简单，只要修改配置文件就可以了，完全具有热插拔的特性。

6、IoC的原理：

控制反转是spring框架的核心。其原理是基于面向对象(OO)设计原则的The Hollywood Principle：Don't call us, we'll call you（别找我，我会来找你的）。也就是说，所有的组件都是被动的，所有的组件初始化和调用都由容器负责。组件处在一个容器当中，由容器负责管理。简单的来讲，就是由容器控制程序之间的关系，而非传统实现中，由程序代码直接操控，即在一个类中调用另外一个类。这也就是所谓“控制反转”的概念所在：控制权由应用代码中转到了外部容器，控制权的转移，即所谓反转。

看到这里，相信大家已经对IoC有了初步的了解，那我们接着看看Spring框架。

轻量——从大小与开销两方面而言Spring都是轻量的。完整的Spring框架可以在一个大小只有1MB多的JAR文件里发布。并且Spring所需的处理开销也是微不足道的。此外，Spring是非侵入式的：典型地，Spring应用中的对象不依赖于Spring的特定类。

控制反转——Spring通过一种称作控制反转（IoC）的技术促进了低耦合。当应用了IoC，一个对象依赖的其它对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象。你可以认为IoC与JNDI相反——不是对象从容器中查找依赖，而是容器在对象初始化时不等对象请求就主动将依赖传递给它。

面向切面——Spring提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务（例如审计（auditing）和事务（transaction）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们并不负责（甚至是意识）其它的系统级关注点，例如日志或事务支持。

容器——Spring包含并管理应用对象的配置和生命周期，在这个意义上它是一种容器，你可以配置你的每个bean如何被创建——基于一个可配置原型（prototype），你的bean可以创建一个单独的实例或者每次需要时都生成一个新的实例——以及它们是如何相互关联的。然而，Spring不应该被混同于传统的重量级的EJB容器，它们经常是庞大与笨重的，难以使用。

框架——Spring可以将简单的组件配置、组合成为复杂的应用。在Spring中，应用对象被声明式地组合，典型地是在一个XML文件里。Spring也提供了很多基础功能（事务管理、持久化框架集成等等），将应用逻辑的开发留给了你。

MVC——Spring的作用是整合，但不仅仅限于整合，Spring 框架可以被看做是一个企业解决方案级别的框架。客户端发送请求，服务器控制器（由DispatcherServlet实现的)完成请求的转发，控制器调用一个用于映射的类HandlerMapping，该类用于将请求映射到对应的处理器来处理请求。HandlerMapping 将请求映射到对应的处理器Controller（相当于Action）在Spring 当中如果写一些处理器组件，一般实现Controller 接口，在Controller 中就可以调用一些Service 或DAO 来进行数据操作 ModelAndView 用于存放从DAO 中取出的数据，还可以存放响应视图的一些数据。 如果想将处理结果返回给用户，那么在Spring 框架中还提供一个视图组件ViewResolver，该组件根据Controller 返回的标示，找到对应的视图，将响应response 返回给用户。

所有Spring的这些特征使你能够编写更干净、更可管理、并且更易于测试的代码。它们也为Spring中的各种模块提供了基础支持。