

Spring

谈谈自己对于 Spring IoC 和 AOP 的理解

IoC

IoC (Inverse of Control:控制反转) 是一种**设计思想**, 就是 **将原本在程序中手动创建对象的控制权, 交由Spring框架来管理**。IoC 在其他语言中也有应用, 并非 Spring 特有。IoC 容器是 Spring 用来实现 IoC 的载体, IoC 容器实际上就是个 Map (key, value) ,Map 中存放的是各种对象。

将对象之间的相互依赖关系交给 IOC 容器来管理, 并由 IOC 容器完成对象的注入。这样可以很大程度上简化应用的开发, 把应用从复杂的依赖关系中解放出来。IOC 容器就像是一个工厂一样, 当我们需要创建一个对象的时候, 只需要配置好配置文件/注解即可, 完全不用考虑对象是如何被创建出来的。在实际项目中一个 Service 类可能有几百甚至上千个类作为它的底层, 假如我们需要实例化这个 Service, 你可能要每次都要搞清这个 Service 所有底层类的构造函数, 这可能会把人逼疯。如果利用 IOC 的话, 你只需要配置好, 然后在需要的地方引用就行了, 这大大增加了项目的可维护性且降低了开发难度。

Spring 时代我们一般通过 XML 文件来配置 Bean, 后来开发人员觉得 XML 文件来配置不太好, 于是 SpringBoot 注解配置就慢慢开始流行起来。

推荐阅读: <https://www.zhihu.com/question/23277575/answer/169698662>

Spring IOC的初始化过程:



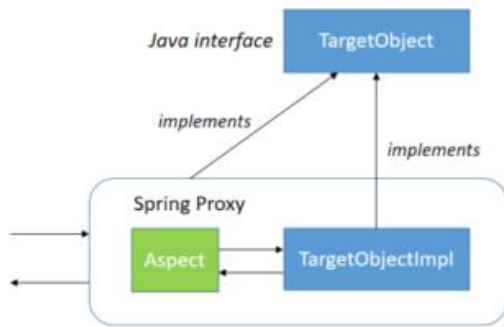
AOP

AOP(Aspect-Oriented Programming:面向切面编程)能够将那些与业务无关, **却为业务模块所共同调用的逻辑或责任 (例如事务处理、日志管理、权限控制等) 封装起来**, 便于**减少系统的重复代码, 降低模块间的耦合度**, 并有利于未来的可拓展性和可维护性。

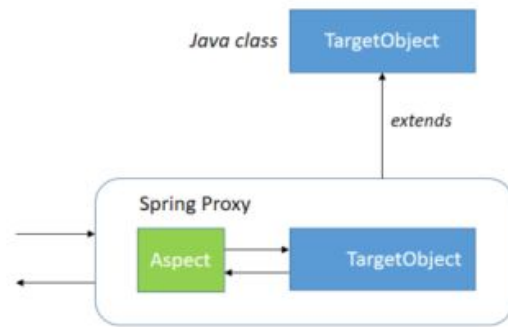
Spring AOP就是基于动态代理的, 如果要代理的对象, 实现了某个接口, 那么Spring AOP会使用**JDK Proxy**, 去创建代理对象, 而对于没有实现接口的对象, 就无法使用 JDK Proxy 去进行代理了, 这时候 Spring AOP会使用**Cglib**, 这时候Spring AOP会使用 **Cglib** 生成一个被代理对象的子类来作为代理, 如下图所示:

Spring AOP Process

JDK Proxy (interface based)



CGLib Proxy (class based)



当然你也可以使用 AspectJ ,Spring AOP 已经集成了AspectJ , AspectJ 应该算的上是 Java 生态系统中最完整的 AOP 框架了。

使用 AOP 之后我们可以把一些通用功能抽象出来，在需要用到的地方直接使用即可，这样大大简化了代码量。我们需要增加新功能时也方便，这样也提高了系统扩展性。日志功能、事务管理等等场景都用到了 AOP 。