**功能**

**IOC**

**AOP**

定义

Spring是一个轻量级的控制反转和面向切面的容器框架

Spring 能帮我们做什么

①Spring 能根据配置文件创建及组装对象之间的依赖关系

②Spring 面向切面编程能无耦合的实现日志记录，性能统计，安全控制

③Spring 能帮我们管理数据库事务

④Spring 还提供了与第三方数据访问框架（如Hibernate、JPA）无缝集成

Spring 中常用术语：

框架：是能完成一定功能的半成品

框架能够帮助我们完成的是：项目的整体框架、一些基础功能、规定了类和对象如何创建，如何协作等，当我们开发一个项目时，框架帮助我们完成了一部分功能，我们自己再完成一部分，那这个项目就完成了

非侵入式设计：

从框架的角度可以理解为：无需继承框架提供的任何类

这样我们在更换框架时，之前写过的代码几乎可以继续使用

轻量级和重量级：

轻量级是相对于重量级而言的，轻量级一般就是非入侵性的、所依赖的东西非常少、资源占用非常少、部署简单等等，其实就是比较容易使用，而重量级正好相反

容器：

从程序设计角度看就是装对象的的对象，因为存在放入、拿出等操作，所以容器还要管理对象的生命周期

注意：bean 的各种名称——虽然 Spring 用 bean 或者 JavaBean 来表示应用组件，但并不意味着 Spring 组件必须遵循 JavaBean 规范，一个 Spring 组件可以是任意形式的 POJO

**IoC**：Inverse of Control（控制反转）

读作“反转控制”，更好理解，不是什么技术，而是一种设计思想，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**

正控：若要使用某个对象，需要自己去负责对象的创建（new）

反控：若要使用某个对象，只需要从 Spring 容器中获取需要使用的对象，不关心对象的创建过程，也就是把创建对象的控制权反转给了Spring框架

**DI**：Dependency Injection（依赖注入）

指 Spring 创建对象的过程中，将对象依赖的属性（简单值，集合，对象）通过配置设值给该对象

实现IOC思想需要DI做支持

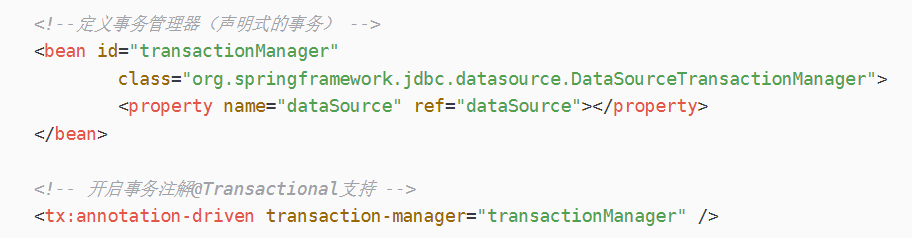
注入方式: 1.set方式注入 2.构造方法注入 3.接口注入

注入类型: 1.值类型注入 2.引用类型注入

依赖注入不能单独存在，需要在ioc的基础之上完成操作

静态变量/类变量是类的属性,spring则是基于对象层面上的依赖注入

**开启事务**



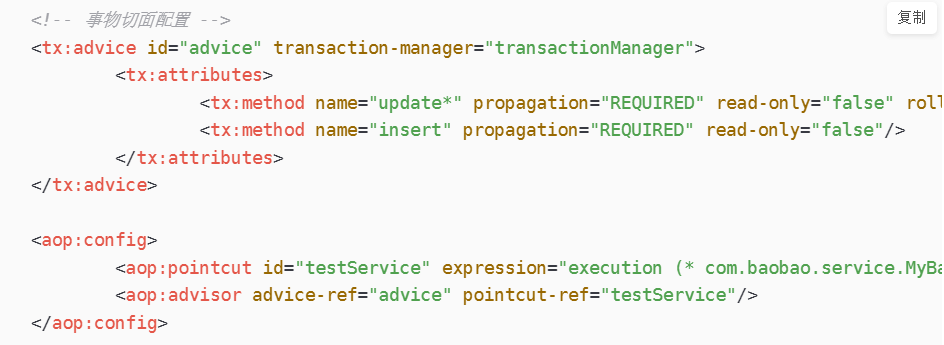
Spring 事务管理分为编程式和声明式的两种方式

编程式事务指的是通过编码方式实现事务；

声明式事务基于 AOP,将具体业务逻辑与事务处理解耦

声明式事务有两种方式

一种是在配置文件（xml）中做相关的事务规则声明



一种是基于 @Transactional 注解的方式

默认配置下

只会回滚运行时、未检查异常（继承自 RuntimeException 的异常）或者 Error

@Transactional 注解只能应用到 public 方法才有效

**AOP**

面向切面编程的思想里面，把功能分为核心业务功能，和周边功能

核心业务，比如登陆，增加数据，删除数据都叫核心业务

周边功能(横切关注功能)，比如性能统计，日志，事务管理等等

* 周边功能被定义为切面
* 在面向切面编程AOP的思想里面，核心业务功能和切面功能分别独立进行开发，然后把切面功能和核心业务功能 "编织" 在一起，这就叫 AOP

AOP 的目的

将那些与业务无关，却为业务模块所共同调用的逻辑（例如事务处理、日志管理、权限控制等）封装成一个可重用模块，减少重复代码，降低模块间的耦合度

AOP 当中的概念：

* 切入点（Pointcut横切关注点）

在哪些类，哪些方法上切入（where）

* 通知（Advice）

在方法执行的什么时间（when:方法前/方法后/方法前后）做什么（what:增强的功能）

* 切面（Aspect）

切面 = 切入点 + 通知，通俗点就是：在什么时机，什么地方，做什么增强！

* 织入（Weaving）

将切面应用到目标对象并导致代理对象创建的过程

* 连接点（Joinpoint） ：程序执行过程中的某一行为，

例如，调用UserService.get或者UserService.delete抛出异常等行为

* 目标对象（Target Object） ：被一个或者多个切面所通知的对象

例如，AServcieImpl和BServiceImpl，在实际运行时，Spring AOP采用代理实现，实际AOP操作的是目标对象的代理对象

实操

两种配置方式，一是xml配置的方式，二是自动注解的模式

Xml:



注解：

1. <aop:aspectj-autoproxy/><!-- 开启aop的注解方式 -->
2. 编写增强类

环绕通知=前置+目标方法执行+后置通知, 可以控制目标方法何时执行

**Spring IoC 容器**

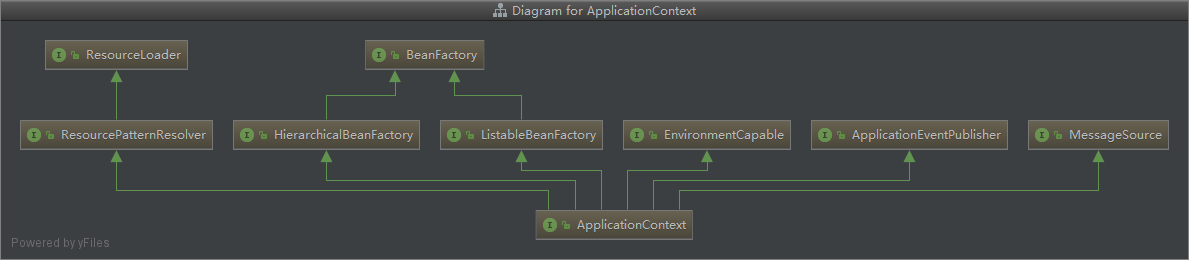
IOC容器简单理解就是Spring实现IOC的一套框架程序

管理和容纳我们所开发的 Bean,并且我们可以从中获取各种发布在 Spring IoC 容器里的 Bean

Spring IoC 容器的设计主要是基于以下两个接口：

BeanFactory、ApplicationContext

BeanFactory 是 Spring IoC 容器所定义的最底层接口，而 ApplicationContext 是其最高级接口之一，并对 BeanFactory 功能做了许多的扩展，所以在绝大部分的工作场景下，都会使用 ApplicationContext 作为 Spring IoC 容器



BeanFactory只提供了最简单的IoC功能

负责读取bean配置文档，管理bean的加载，实例化，维护bean之间的依赖关系，负责bean的声明周期

BeanFactroy采用的是延迟加载形式来注入Bean的, ApplicationContext则相反，它是在容器启动时，一次性创建了所有的Bean, 占用内存空间

ApplicationContext

1.继承了 BeanFactory，拥有了基本的 IoC 功能；

2.除此之外，ApplicationContext 还提供了以下功能：

① 支持国际化；

② 支持消息机制；

③ 支持统一的资源加载；

④ 支持AOP功能；

ApplicationContext 的子类——ClassPathXmlApplicationContext

ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");

springmvc和spring它俩都是容器，容器就是管理bean对象的地方

例如Tomcat，就是管理servlet对象

springmvc就是管理controller对象的容器，spring就是管理service和dao的容器

所以我们在springmvc的配置文件里配置的扫描路径就是controller的路径，而spring的配置文件里自然配的就是service和dao的路径

spring容器是父容器，springmvc是子容器。在子容器里可以访问父容器里的对象，但是在父容器里不可以访问子容器的对象

说的通俗点就是，在controller里可以访问service对象，但是在service里不可以访问controller对象

web容器(Tomcat)是管理servlet，以及监听器(Listener)和过滤器(Filter)的。 这些都是在web容器的掌控范围里。但他们不在spring和springmvc的掌控范围里 。因此，我们无法在这些类中直接使用Spring注解的方式来注入我们需要的对象，是无效的