Spring注解 IOC

1.简介

在曾经的岁月里，Java和XML是完美契合，许多人认为Java是跨平台的语言，而XML是跨平台的数据交换格式，所以JAVA和XML应该是天作之合。在这种潮流下，以前的JAVA框架不约而同的选择了XML作为配置文件

xml 。目前 web 应用中几乎都使用 xml 作为配置项，例如我们常用的框架 Struts、spring、hibernate、IBatis 等等都采用 xml 作为配置。xml 之所以这么流行，是因为它的很多优点是其它技术的配置所无法替代的。

XML优点

1. xml 作为可扩展标记语言最大的优势在于开发者能够为软件量身定制适用的标记，使代码更加通俗易懂。
2. 利用 xml 配置能使软件更具扩展性。例如 Spring 将 class 间的依赖配置在 xml 中，最大限度地提升应用的可扩展性。
3. 具有成熟的验证机制确保程序正确性。利用 Schema 或 DTD 可以对 xml 的正确性进行验证，避免了非法的配置导致应用程序出错。
4. 修改配置而无需变动现有程序

XML缺点

1. 需要解析工具或类库的支持。
2. **解析 xml 势必会影响应用程序性能，占用系统资源。**
3. 配置文件过多导致管理变得困难。
4. 编译期无法对其配置项的正确性进行验证，或要查错只能在运行期。
5. IDE 无法验证配置项的正确性无能为力。
6. 查错变得困难。往往配置的一个手误导致莫名其妙的错误。
7. 开发人员不得不同时维护代码和配置文件，开发效率变得低下。
8. 配置项与代码间存在潜规则。改变了任何一方都有可能影响另外一方。

新的技术出现一定会为了解决当下的问题才会出现的。

2.基于注解的依赖注入

基于注解（Annotation）的配置有越来越流行的趋势，Spring 3顺应这种趋势，提供了完全基于注释配置 Bean、装配 Bean 的功能，开发人员可以使用基于注解的 Spring IOC 替换原来基于 XML 的配置。

注解配置相对于 XML 配置具有很多的优势：

注解本身没有功能的，就和xml一样。注解和xml都是一种元数据，元数据即解释数据的数据，这就是所谓配置。

Spring注解方式减少了配置文件内容，更加便于管理，并且使用注解可以大大提高了开发效率！

导入AOP依赖JAR

```
aspectjweaver.jar
spring-aop-4.3.6.RELEASE.jar
spring-aspects-4.1.6.RELEASE.jar
```

3 注解依赖注入

既然我们不再使用Spring配置文件来配置任何Bean实例，

那么我们只能指望Spring会自动搜索某些路径下的Java类，并将这些类注册成Bean实例。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
         http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        ">
     <!-- 指定Spring IOC扫描器
     base-package 指要需要扫描的包
     use-default-filters表示使用默认的扫描过滤器 会扫描包含
Service,Component,Repository,Controller注解
     resource-pattern="**/*.class" 扫描指定包内下的所有的(子)类
      -->
    <context:component-scan base-package="com.qfjy" use-default-filters="true"
    resource-pattern="**/*.class">

    </context:component-scan>
  </beans>
```

Ant风格:

通配符	说明
?	匹配任何单字符
*	匹配0或者任意数量的字符
**	匹配0或者更多的目录

4 声明Bean注解

@Component 标注一个普通的Spring Bean类 **@Service** 标注一个业务逻辑组件类 **@Controller** 标注一个控制器组件类 **@Repository** 标注一个DAO组件类

@Component是所有受Spring管理组件的通用形式；而@Repository、@Service和 @Controller则是@Component的细化，用来表示更具体的用例(例如，分别对应了持久化层、服务层和表现层)。 也就是说，你能用@Component来注解你的组件类，但如果用@Repository、@Service 或@Controller来注解它们，你的类也许能更好地被工具处理，或与切面进行关联。例如，这些典型化注解可以成为理想的切入点目标。也许还能携带更多语义。

备注：以上注解作用在类上。都是负责声明一个bean到Spring上下文中，bean 的ID默认为类的名称开头的小写字母。如果想自定义，可以@Service(“aaaaa”)这样来指定。

可通过 context:exclude-filter 排除指定扫描的某个注解(不扫描@Service注解)

```
<context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Service"/>
```

可通过 context:include-filter 只包含某个指定扫描的注解(仅扫描带@Service注解)

```
<context:component-scan base-package="com.qfjy" use-default-filters="false">
    <context:include-filter type="annotation"
expression="org.springframework.stereotype.Service"/>
    </context:component-scan>
```

如果改变对象的作用域，可通过 @Scope(“prototype”)来改变 注解来改变。（作用于类上）

5 组件装配(依赖注入)

简要

自动装配是指Spring 在装配 Bean 的时候，根据指定的自动装配规则，将某个 Bean 所需要引用类型的 Bean 注入进来。可以在类的成员变量上,构造方法,setter方法使用

context:component-scan 元素还会自动注册 AutowiredAnnotationBeanPostProcessor 实例,

该实例可以自动装配具有 **@Autowired 和 @Resource 、@Inject**注解的属性.

@Autowired

Spring通过@Autowired实现依赖注入： Spring 2.5 引入了 @Autowired 注释，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。（是根据类型进行自动装配）

```
@Service
public class UserServiceImpl  implements UserService{
    @Autowired
    private UserDao userDao;
```

1. 属性加上@Autowired后不需要getter()和setter()方法，Spring会自动注入。
2. 默认情况下required为true,即一定要找到匹配bean，否则报异常
3. 或者 @Autowired(required=false) 如果根据类型找不到，不报错。显示为null
4. 如下：如果spring 上下文中没有找到该类型的bean 时， 才会使用

```
    @Autowired(required=false)
    private UserDao userDao=new UserDao();
```

@Autowired 根据bean 类型从spring 上线文中进行查找，注册类型必须唯一，否则报异常

在接口前面标上@Autowired注释使得接口可以被容器注入，如：当接口存在两个实现类的时候必须使用@Qualifier指定注入哪个实现类，否则可以省略，

@Qualifier("userDao") //（是根据名称进行自动装配的）

```
    @Autowired
    @Qualifier("userDaoImpl1")
    private UserDao userDao;
```

@Value("qfjy") //对属性进行赋值

@Resource

@Resource（这个注解属于J2EE的），默认按照名称进行装配，名称可以通过name属性进行指定， 如果没有指定name属性，当注解写在字段上时，默认取字段名进行按照名称查找，如果注解写在setter方法上默认取属性名进行装配。 当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

```
    @Resource   //默认会根据字段名 查找 UserDao接口下的userDaoImpl1名称的BEAN进行注入
    private UserDao userDaoImpl1;
```

```
   @Resource(name="userDaoImpl") //查找userDaoImpl名称进行注入
   private UserDao userDaoImpl1;
```

6 Bean生命周期

通过在目标方法上标注@PostConstruct和@PreDestroy注解指定初始化或销毁方法，可以定义任意多个方法

```
@PostConstruct
    public void init(){
        System.out.println("2--init");
    }
@PreDestroy
    public void destory(){
        System.out.println("3--销毁");
    }
```

7 完整总结

	基于XML配置	基于注解配置	基于Java类配置
Bean定义	在XML文件中通过元素定义Bean，如:	在Bean实现类处通过标注@Compoment或衍型类@Repository、@Service及@Controller定义Bean	在标注了@Configuration的Java类中，通过在类方法上标注@Bean定义一个Bean。方法必须提供Bean的实例化逻辑
Bean名称	通过的id或name属性定义，如:	通过注解的value属性定义，如@Component(“userDao”)。默认名称为小写字母打头的类名（不带包名）:userDao	通过@Bean的name属性定义，如@Bean(“userDao”)，默认名称为方法名
Bean注入	通过子元素或通过p命名空间的动态属性，如p:userDao-ref=”userDao”进行注入	通过在成员变量或方法入参处标注@Autowired，按类型匹配自动注入。还可以配合使用@Qualifier按名称匹配方式注入	比较灵活，可以通过在方法处通过@Autowired方法入参绑定Bean，然后在方法中通过代码进行注入，还可以通过调用配置类的@Bean方法进行注入
Bean生命过程方法	通过的init-method和destory-method属性指定Bean实现类的方法名。最多只能指定一个初始化方法和一个销毁方法。	通过在目标方法上标注@PostConstruct和@PreDestroy注解指定初始化或销毁方法，可以定义任意多个方法	通过@Bean的initMethod或destoryMethod指定一个初始化或销毁方法。 对于初始化方法来说，你可以直接在方法内部通过代码的方式灵活定义初始化逻辑

	基于XML配置	基于注解配置	基于Java类配置
Bean作用范围	通过的scope属性指定，如：	通过在类定义处标注@Scope指定，如@Scope(“prototype”)	通过在Bean方法定义处标注@Scope指定
适合场景	1）Bean实现类来源于第三方类库，如DataSource，JdbcTemplate等，因无法在类中标注注解，通过XML配置方式较好 2）命名空间的配置，如aop、context等，只能采用基于XML的配置	Bean的实现类是当前项目开发的，可以直接在Java类中使用基于注解的配置	基于Java类配置的优势在于可以通过代码方式控制Bean初始化的整体逻辑。所以如果实例化Bean的逻辑比较复杂，则比较适合用基于Java类配置的方式

8 Java类配置

Spring1.x时代

在Spring1.x时代，都是通过xml文件配置bean，随着项目的不断扩大，需要将xml配置分放到不同的配置文件中，需要频繁的在java类和xml配置文件中切换。

Spring2.x时代

随着JDK 1.5带来的注解支持，Spring2.x可以使用注解对Bean进行声明和注入，大大的减少了xml配置文件，同时也大大简化了项目的开发。

那么，问题来了，究竟是应该使用xml还是注解呢?

最佳实践：

1、 应用的基本配置用xml，比如：数据源、资源文件等； 2、 业务开发用注解，比如：Service中注入bean等；

Spring3.x到Spring4.x到Spring5.x

从Spring3.x开始提供了Java配置方式，使用Java配置方式可以更好的理解你配置的Bean，现在我们就处于这个时代，并且Spring4.x和Springboot都推荐使用java配置的方式。

Java配置是Spring4.x推荐的配置方式，可以完全替代xml配置