# 注解和自定义注解

大家好，我是一个爱举铁的程序员Shr。

本文介绍Java中的注解，实现一个自定义的注解。阅读本篇文章可能需要10分钟。

# 一、注解的起源

注解是从JDK 1.5开始引入的新特性。与类，接口，枚举在同一个层次。

# 二、常用的注解

在学习Java语言的时候就接触到了注解。

例如@Override，IDE工具实现编译检查，最早遇到这个注解是在重写类的toString()方法时。

@Deprecated，标记为过时，可用在方法和字段上，最早遇到这个注解是java.lang.Date类中的getDate()方法。

# 三、使用注解的体会

我第一次感受到注解的方便是在使用Spring注解的时候，上一篇文章写的是Spring AOP的使用，程序例子中用的都是配置文件，并没有讲注解，其实用注解使用Spring框架很方便，看几篇文章就会了。当项目中有很多类的时候，要在Spring配置文件中将类注入到容器中，需要写大量的<bean></bean>标签。但是后来用@Controller，@Service，@Repository标签来实现注入。

注解的使用确实提高了开发效率，为什么在一个类上加上注解就能代替原来的XML配置呢，这是我是在学习Spring Boot的时候才开始考虑到的。

所以我特意去看了一下注解。

# 四、元注解

元注解可以理解为注解的注解。在Eclipse中按住Ctrl再加鼠标左击，我们来看看@Override和@Deprecated注解的源代码。

## 4.1 @Override注解源代码

**package** java.lang;

**import** java.lang.annotation.\*;

/\*\*

\* Indicates that a method declaration is intended to override a

\* method declaration in a superclass. If a method is annotated with

\* this annotation type but does not override a superclass method,

\* compilers are required to generate an error message.

\*

\* **@author** Joshua Bloch

\* **@since** 1.5

\*/

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***SOURCE***)

**public** **@interface** Override {

}

## 4.2 @Deprecated注解源代码

**package** java.lang;

**import** java.lang.annotation.\*;

/\*\*

\* A program element annotated &#64;Deprecated is one that programmers

\* are discouraged from using, typically because it is dangerous,

\* or because a better alternative exists. Compilers warn when a

\* deprecated program element is used or overridden in non-deprecated code.

\*

\* **@author** Neal Gafter

\* **@since** 1.5

\*/

@Documented

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** Deprecated {

}

这两个注解都用了@Retention注解，用于指定被@Retention修饰的注解要保留多久。

@Retention注解中有个属性值是RetentionPolicy.SOURCE，表示编译器要丢弃的注解。

另外还有两个值是RetentionPolicy.CLASS，编译器将注解记录在class文件中，但是运行时不保留注解。

RetentionPolicy.RUNTIME，编译器将注解记录在class文件中，运行时也保留注解，因此可以通过反射读取该注解的信息。

注解默认的属性是value，而value属性可以省略不写。

@Override注解中的@Target注解，注解可以适用的程序元素种类，在哪个地方，这里定义是方法，所以@Override注解只能用在方法上。而@Target注解中的ElementType是一个枚举类型，ElementType中还定义了字段，构造方法，局部变量等枚举常量，也就是说自定义的注解要限制使用，可以用@Target注解来修饰自定义注解。

**package** java.lang.annotation;

/\*\*

\* A program element type. The constants of this enumerated type

\* provide a simple classification of the declared elements in a

\* Java program.

\*

\* <p>These constants are used with the {@link Target} meta-annotation type

\* to specify where it is legal to use an annotation type.

\*

\* **@author** Joshua Bloch

\* **@since** 1.5

\*/

**public** **enum** ElementType {

/\*\* Class, interface (including annotation type), or enum declaration \*/

***TYPE***,

/\*\* Field declaration (includes enum constants) \*/

***FIELD***,

/\*\* Method declaration \*/

***METHOD***,

/\*\* Parameter declaration \*/

***PARAMETER***,

/\*\* Constructor declaration \*/

***CONSTRUCTOR***,

/\*\* Local variable declaration \*/

***LOCAL\_VARIABLE***,

/\*\* Annotation type declaration \*/

***ANNOTATION\_TYPE***,

/\*\* Package declaration \*/

***PACKAGE***

}

如果注解类型声明中不存在@Target元注解，则声明的类型可以用在任一程序元素上。

@Deprecated注解中的@Document注解，用javadoc命令生成文档时，被@Document修饰的注解将一同被写入文档。

@Inherited，被它修饰的注解具有继承性，某个类使用了被@Inherited修饰的注解，则其子类将自动具有该注解。

# 五、自定义注解

语法就不在这里描述了，下面举一个简单的栗子，自定义一个数据源注解连接数据库。

## 5.1 准备工作

MySQL驱动包

在MySQL数据库中新建一个名为“annotation\_20180808”的数据库

## 5.2 MySQLDataSource

**package** com.shrmus.annotation;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

/\*\*

\* 自定义注解，连接MySQL的数据源

\* <p>Title:MySQLDataSource</p>

\* <p>Description:</p>

\* **@author** Shr

\* **@date** 2018年8月8日下午4:45:45

\* **@version**

\*/

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** MySQLDataSource {

String driverName();

String url();

String user();

String password();

}

定义了一个注解，还有4个属性，并且用元注解指示这个注解只能用在方法上。

## 5.3 测试类

**package** com.shrmus.annotation.test;

**import** java.lang.reflect.Method;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** com.shrmus.annotation.MySQLDataSource;

/\*\*

\* 测试类

\* <p>Title:AnnotationTest</p>

\* <p>Description:</p>

\* **@author** Shr

\* **@date** 2018年8月8日下午4:48:10

\* **@version**

\*/

**public** **class** AnnotationTest {

/\*\*

\* 获取数据库连接

\* **@return**

\* **@throws** SecurityException

\* **@throws** NoSuchMethodException

\* **@throws** ClassNotFoundException

\* **@throws** SQLException

\*/

@MySQLDataSource(

driverName = "com.mysql.jdbc.Driver",

url = "jdbc:mysql://localhost:3306/annotation\_20180808",

user = "root",

password = "shrmus")

**public** **static** Connection getConnection() **throws** NoSuchMethodException, SecurityException, ClassNotFoundException, SQLException {

// 获取该类的字节码

Class clazz = AnnotationTest.**class**;

// 获取此方法

Method method = clazz.getMethod("getConnection",**null**);

// 获取此方法上的注解

MySQLDataSource mySQLDataSource = method.getAnnotation(MySQLDataSource.**class**);

// 获取注解的属性值

String driverName = mySQLDataSource.driverName();

String url = mySQLDataSource.url();

String user = mySQLDataSource.user();

String password = mySQLDataSource.password();

// 加载数据库驱动

Class.*forName*(driverName);

// 获取数据源

Connection connection = DriverManager.*getConnection*(url, user, password);

**return** connection;

}

**public** **static** **void** main(String[] args) **throws** Exception{

Connection connection = *getConnection*();

System.***out***.println(connection);

connection.close();

}

}

getConnection()方法上加了刚刚自定义的注解，并且给这4个属性赋值，然后通过反射获取注解的属性值，再后来就是通过JDBC获取连接了。

运行程序后控制台打印结果：

com.mysql.jdbc.JDBC4Connection@46f7f36a

打印结果不是null，说明获取了连接。

# 总结

注解的使用使得开发的程序中没有大量的配置文件，提高开发速度。

不管是注解还是配置文件，最终都是通过反射获取属性值，所以反射需要掌握好。

源代码地址：

<https://github.com/ShrMus/Design-Pattern/tree/master/annotation_20180808>