

最近我的助理，“娃娃”问了我一次注解应该怎么写，我想注解应该太简单了吧，从我刚开始看java代码就能看到的@Override,到后来经常挺行我加上的@SuppressWarnings，当然这些注解我们用起来，看起来都是非常简单的。并且我本人是做安卓开发的，经常用各种注解框架做界面的绑定，我就简单的说了几句，利用反射啊，不改变代码逻辑，可以起到拦截的作用啊等等。但是当我自己想从头想写一个的时候，就发现并非那么简单了，经过了几个小时的研究，打算写这么一篇文章记录一下。

想了解注解标签的原理，我们最好先搞懂，java的反射机制。所以可能这篇文章的篇幅较长，希望大家可以认真的读完。

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为java语言的反射机制。

以上内容，来自百度文库。

用我的话说java的反射机制就是，可以通过程序的编写，让程序在运行时加载使用一个全新的Classes。

Java反射机制主要提供了以下功能：在运行时判断任意一个对象所属的类，在运行时构造任意一个类的对象，在运行时判断任意一个类所具有的成员变量和方法，在运行时调用任意一个对象的方法，生成动态代理。

在java中，所有类的基类是Object类，在这个函数中为我们提供了getClass()的方法。这个Class是一个非常特殊的类，由于我对这部分知识理解的也不是很到位，下面借用一下百度百科的解释吧：

Class 类十分特殊。它和一般类一样继承自Object，其实体用以表达Java程序运行时的classes和interfaces，也用来表达enum、array、primitive Java types（boolean, byte, char, short, int, long, float, double）以及关键词void。当一个class被加载，或当加载器（class loader）的defineClass()被JVM调用，JVM 便自动产生一个Class 对象。如果您想藉由“修改Java标准库源码”来观察Class 对象的实际生成时机（例如在Class的constructor内添加一个println()），这样是行不通的！因为Class并没有public constructor。

下面我们便可以开始手动实现一个注解了：

```
package annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Reflect {
    String name() default "sunguoli";
}
```

```
package annotation;

import java.lang.reflect.Method;

public class ReflectProcessor {
    public void parseMethod(final Class<?> clazz) throws Exception {
        final Object obj = clazz.getConstructor(new Class[] {} ).newInstance(new Object[] {});
        final Method[] methods = clazz.getDeclaredMethods();
        for (final Method method : methods) {
            final Reflect my = method.getAnnotation(Reflect.class);
            if (null != my) {
                method.invoke(obj, my.name());
            }
        }
    }
}
```

```

package annotation;

public class ReflectTest {


    @Reflect
    public static void sayHello(final String name) {
        System.out.println("==> Hi, " + name + " [sayHello]");
    }

    @Reflect(name = "AngelaBaby")
    public static void sayHelloToSomeone(final String name) {
        System.out.println("==> Hi, " + name + " [sayHelloToSomeone]");
    }

    public static void main(final String[] args) throws Exception {
        final ReflectProcessor relectProcessor = new ReflectProcessor();
        relectProcessor.parseMethod(ReflectTest.class);
    }
}

```

以上我们的接口就写完了~下面看一下运行结果~



```

<terminated> ReflectTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java (2016年10月1
==> Hi, lin [sayHello]
==> Hi, AngelaBaby [sayHelloToSomeone]

```

这样我们就测试了我们的注解，主要利用的是java的反射，在反射过程中调用了装载的方法就可以啦~

当然啊，我知道我们这里面调用的反射只能算是反射的原理和最基本的概念，我写出来的注解，都是相当的水的注解，和各种框架中的注解几乎不可能相提并论，我最近也很留意这方面的源码，有别的收货还是会分享出来的~~