java 注解学习

作者：flowaters
原文链接：https://ld246.com/article/1531619553137
来源网站：链滴

背景

注解(Annontation)是Java5中引入的新特征。其用途有：

- 生成文档：如　@param, @return
- 配置管理：如spring中的注解配置
- 格式检查：编译时进行格式检查，如　@override

第一个例子

VehicleInfo

VehicleInfo.java

```
package testAnnotation;

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

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface VehicleInfo {
    int id() default 0;

    String type() default "";
}
```

Car

Car.java

```
package testAnnotation;

public class Car {
    @VehicleInfo(id = 1, type = "汽车")
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Train

Train.java

```
package testAnnotation;

public class Train {
    @VehicleInfo(id = 2, type = "火车")
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

VehicleUtil

VehicleUtil.java

```
package testAnnotation;

import java.lang.reflect.Field;

public class VehicleUtil {

    public static void getTrafficInfo(Class<?> clazz) {
        Field[] fields = clazz.getDeclaredFields();

        for (Field field : fields) {
            if (field.isAnnotationPresent(VehicleInfo.class)) {
                VehicleInfo info = (VehicleInfo) field.getAnnotation(VehicleInfo.class);
                System.out.println("编号: " + info.id());
                System.out.println("类别: " + info.type());
            }
        }
    }
}
```

MainVehicle

MainVehicle.java

```
package testAnnotation;
```

```
public class MainVehicle {
    public static void main(String[] args) {
        System.out.println("Car info: ");
        VehicleUtil.getTrafficInfo(Car.class);

        System.out.println("Train info: ");
        VehicleUtil.getTrafficInfo(Train.class);

        System.out.println("BMW info: ");
        Car car = new Car();
        car.setName("BMW");
        VehicleUtil.getTrafficInfo(car.getClass());
        System.out.println("Name: " + car.getName());
    }
}
```

执行结果

```
Car info:
编号: 1
类别: 汽车

Train info:
编号: 2
类别: 火车

BMW info:
编号: 1
类别: 汽车
Name: BMW
```

其中的技术细节：

1. VehicleUtil.getTrafficInfo中，通过反射方法(SoftReference)拿到fields.
2. 查找field中匹配的Annotation
3. 注解是作用在类上，而不是类实例出的对象上。

原理

注解定义时使用了@interface，这是一个继承了java.lang.annotation.Annotation接口的特殊接口。

VehicleUtil.getTrafficInfo中的info.id()是怎么调用的呢?

如果是通常的类实例调用过程，info是类实例，id()是类方法。

在本示例中，info是$Proxy1类的实例，这个proxy类是Java运行时动态生成的代理类。

代理类的类方法调用都会实现了java.lang.reflect.InvocationHandler接口，具体到注解代理类中，方法的调用实际执行了sun.reflect.annotation.AnnotationInvocationHandler类。

id()类方法的执行过程为：

1. 调用了 AnnotationInvocationHandler.invoke(Object, Method, Object[])方法
2. 在 memberValues中，本例的值为{type=汽车, id=1}，找到id对应的值1
3. 返回 1

所以本质上，注解是一个继承了Annotation的特殊接口的内置的特殊接口。

元注解

在自定义注解时，需要用到一些内置的注解。这些内置的注解即为元注解，用来即注解其它的注解。

元注释有4个：

Documented

如果有@Documented注解，即做为了public contract的一部分，可以被工具调用，如javadoc。

反之，如果没有@Documented注解，即不做为public contract的一部分，不可以被工具调用。

Retention

注解作用的阶段，其类型为RetentionPolicy，值为

- SOURCE: 会被编译器抛弃
- CLASS: 会编译在class文件中，但是不需要VM运行时加载。这个为默认值。
- RUNTIME: 会编译在class文件中，同时需要在VM运行时加载。

Target

注解作用的范围，即在哪个位置用这个注解是合法的。常见的值为：

- TYPE: 类，接口，枚举类型
- FIELD: 变量
- METHOD: 类
- PARAMETER: 形式参数
- CONSTRUCTOR: 构造方法
- LOCAL_VARIABLE: 局部变量
- ANNOTATION_TYPE: 注解
- PACKAGE: 包
- TYPE_PARAMETER: @Since 1.8, 类型参数
- TYPE_USE: @Since 1.8, 类型使用
- MODULE: @Since 9, 模块

Inherited

类型是被继承的。本文不展开解释。

Native

@Since 1.8

常量字段中的值可以被native代码使用。本文不展开解释。

Repeatable

@Since 1.8

注解类型是可以重复的。本文不展开解释。

常见的标准注解

- Override
- Deprecated
- SuppressWarnings
- SafeVarargs: Since 1.7
- FunctionalInterface: Since 1.8

参考

- 注解Annotation实现原理与自定义注解例子
- java注解:如何实现和使用一个自定义注解?
- javadoc