自定义注解类编写的一些规则:

1. Annotation型定义为@interface, 所有的Annotation会自动继承java.lang.Annotation这一接口,并且不能再去继承别的类或是接口.

2. 参数成员只能用public或默认(default)这两个访问权修饰

3. 参数成员只能用基本类型byte,short,char,int,long,float,double,boolean八种基本数据类型和String、Enum、Class、annotations等数据类型,以及这一些类型的数组.

4. 要获取类方法和字段的注解信息，必须通过Java的反射技术来获取 Annotation对象,因为你除此之外没有别的获取注解对象的方法

5. 注解也可以没有定义成员, 不过这样注解就没啥用了

自定义注解类时, 可以指定目标 (类、方法、字段, 构造函数等) , 注解的生命周期(运行时,class文件或者源码中有效), 是否将注解包含在javadoc中及是否允许子类继承父类中的注解, 具体如下:

1. @Target 表示该注解目标,可能的 ElemenetType 参数包括:

ElemenetType.CONSTRUCTOR 构造器声明

ElemenetType.FIELD 域声明(包括 enum 实例)

ElemenetType.LOCAL\_VARIABLE 局部变量声明

ElemenetType.METHOD 方法声明

ElemenetType.PACKAGE 包声明

ElemenetType.PARAMETER 参数声明

ElemenetType.TYPE 类，接口(包括注解类型)或enum声明

2. @Retention 表示该注解的生命周期,可选的 RetentionPolicy 参数包括

RetentionPolicy.SOURCE 注解将被编译器丢弃

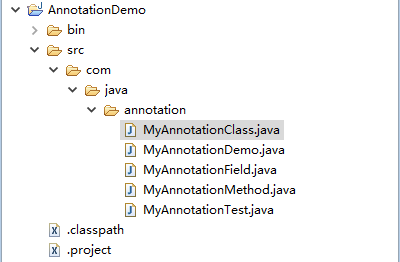
RetentionPolicy.CLASS 注解在class文件中可用，但会被VM丢弃

RetentionPolicy.RUNTIME VM将在运行期也保留注释，因此可以通过反射机制读取注解的信息

3. @Documented 指示将此注解包含在 javadoc 中

4. @Inherited 指示允许子类继承父类中的注解

源代码



1 MyAnnotationClass.java

package com.java.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* 类注解

\* \*/

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

public @interface MyAnnotationClass {

public String msg();

}

2 MyAnnotationMethod.java

package com.java.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* 方法注解

\*\*/

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

public @interface MyAnnotationMethod {

public String common();

}

3 MyAnnotationField.java

package com.java.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.FIELD)

public @interface MyAnnotationField {

boolean request();

}

4 MyAnnotationDemo.java

**package** com.java.annotation;

@MyAnnotationClass(msg = "这是一个类注解")

**public** **class** MyAnnotationDemo {

**public** MyAnnotationDemo() {

}

**public** MyAnnotationDemo(String text) {

**this**.text = text;

}

@MyAnnotationMethod(common = "这是一个方法注解")

**public** **void** method() {

}

@MyAnnotationField(request = **true**)

**private** String text;

}

5 MyAnnotationTest.java

**package** com.java.annotation;

**import** java.lang.reflect.Field;

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

**public** **class** MyAnnotationTest {

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

MyAnnotationDemo demo = **new** MyAnnotationDemo("hello rollen");

MyAnnotationClass annotationClass = demo.getClass().getAnnotation(MyAnnotationClass.**class**);

System.*out*.println(annotationClass.msg());

Method method = **null**;

**try** {

method = demo.getClass().getMethod("method",**new** Class[0]);

} **catch** (SecurityException e) {

e.printStackTrace();

} **catch** (NoSuchMethodException e) {

e.printStackTrace();

}

MyAnnotationMethod annotationMethod = method.getAnnotation(MyAnnotationMethod.**class**);

System.*out*.println(annotationMethod.common());

Field field = **null**;

**try** {

field = demo.getClass().getDeclaredField("text");

} **catch** (SecurityException e) {

e.printStackTrace();

} **catch** (NoSuchFieldException e) {

e.printStackTrace();

}

MyAnnotationField annotationField = field.getAnnotation(MyAnnotationField.**class**);

System.*out*.println(annotationField.request());

}

}

运行结果

