JAVA注解和Spring中对注解的增强

1. 什么是注解?

JAVA中的注释大家都很熟悉了，注释是给开发者看的，可以提升代码的可读性，帮助读者理解代码的意思，当然，真正编译时，编译器是不会处理注释的，虚拟机和不需要注释。注解和注释虽然只有一字之差，但是很不一样，注解是给编译器和虚拟机看的。javac和JVM会获取注解信息根据不同的需求处理注解。比如JDK中有一个自带的注解@Override，当编译器编译代码时，就会对@Override标注的方法进行验证名字是否时父类的方法名字，不一致，提示错误。

总体上，我们可以把注解看作是JAVA代码中携带的给编译器和虚拟机看的一些信息，我们可以在编译期或者运行期通过获取这些信息，完成我们额外的一些需求，实现对程序功能的扩展和增强。

2. 注解如何定义和使用?

我们使用注解一般分为三个步骤:

1. 定义注解本身。
2. 使用注解并携带信息。
3. 在编译或者运行期获取注解信息并使用。

2.1. 定义注解

就像定义类使用class,定义接口使用interface一样，定义注解也有对应的关键字：@interface。基本规则如下:

```
public @interface 注解名称{
    [public] 参数类型 参数名称1() [default 参数默认值];
    [public] 参数类型 参数名称2() [default 参数默认值];
    [public] 参数类型 参数名称n() [default 参数默认值];
}
```

需要注意的是:

1. 注解所有的属性都必须是public的。
2. 注解中元素可以是基本数据类型【包装类不允许】、String、枚举【常见】、其他注解等。
3. 如果注解中的成员名字如果是value，则赋值时可以省略。
4. 可以给注解的成员添加默认值使用default定义，如果有默认值，使用注解时，可以不用赋值。

举个栗子

```
public @interface MyController {
```

```
    //属性名字是value时，赋值时可以不写
    public String value() default "";

    //注解中有基本数据类型，不是包装类
    public int age();

    ///注解中可以有枚举成员 RequestMethod是一个枚举
    public RequestMethod[]  method();
}
```

2.2. 使用注解

注解一旦定义就可以在类上、方法上、属性上等规定的地方去使用注解。

> 在实际中，我们一般是用“别人”【比如：spring框架】写好的注解，所以我们只需要了解、学习注解的含义和如何配置，一般框架都提供的框架的对应的处理程序。

2.3. 获取注解信息

获取注解信息，可以使用反射提供的getAnnotations或者getDeclaredAnnotations()来获取指定类或者方法上的所有的注解，也可以使用getDeclaredAnnotation获取某个注解信息。

3. 元注解--用在注解上的注解?

在java.lang.annotation包中定义了几个用在注解上的注解，我们称之为元注解[这些注解只能用在其他注解上]：@Target 、@Rentention、 @Documented、 @Inherit、 @Repeatable。

---

a. @Target注解

```
package java.lang.annotation;

 /*
 * @since 1.5
 * @jls 9.6.4.1 @Target
 * @jls 9.7.4 Where Annotations May Appear
 */
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
    /**
     * Returns an array of the kinds of elements an annotation type
     * can be applied to.
     * @return an array of the kinds of elements an annotation type
     * can be applied to
     */
    ElementType[] value();
```

```
}
```

@Target注解用来表示注解使用的程序元素的种类，其中有一个属性value,类型是枚举类型ElementType.

```
package java.lang.annotation;

/*
 * @author  Joshua Bloch
 * @since 1.5
 * @jls 9.6.4.1 @Target
 * @jls 4.1 The Kinds of Types and Values
 */
public enum ElementType {
    /** Class, interface (including annotation type), or enum declaration */
    TYPE,

    /** Field declaration (includes enum constants) */
    FIELD,

    /** Method declaration */
    METHOD,

    /** Formal parameter declaration */
    PARAMETER,

    /** Constructor declaration */
    CONSTRUCTOR,

    /** Local variable declaration */
    LOCAL_VARIABLE,

    /** Annotation type declaration */
    ANNOTATION_TYPE,

    /** Package declaration */
    PACKAGE,

    /**
     * Type parameter declaration
     *
     * @since 1.8
     */
    TYPE_PARAMETER,

    /**
     * Use of a type
     *
     * @since 1.8
     */
```

```
    TYPE_USE
}
```

b. @Retention注解

@Rentention注解表示注解的保留策略，其中也有属性value，类型是枚举类型RententionPolicy.CLASS、RetentionPolicy.RUNTIME、RetentionPolicy.SOURCE

```
package java.lang.annotation;
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Retention {
    RetentionPolicy value();
}


package java.lang.annotation;
public enum RetentionPolicy {
   //注解保留到源码中，编译之后的class文件中没有
    SOURCE,
   //注解保留到class文件中，运行期不加载
    CLASS,
   //注解保留到运行期，可以通过反射获取。
    RUNTIME
}
```

c. @Documented注解

@Documented注解用来标注注解是否可以生成到文档中，如果添加了@Documented则注解可以生成到文档中。

d. @Inherit注解

该注解只能用在其他注解上.表示子类可以获取父类中继承的注解且只能获取父类中的注解【不能是接口中的】。

```
package com.etoak.test;

import java.lang.annotation.*;

public class TestAnnotation {
    @Target(ElementType.TYPE)
    @Retention(RetentionPolicy.RUNTIME)
    @Inherited
    @interface A{ }

    @Target(ElementType.TYPE)
    @Retention(RetentionPolicy.RUNTIME)
    @Inherited
```

```
    @interface B{ }

    @Target(ElementType.TYPE)
    @Retention(RetentionPolicy.RUNTIME)
    @interface C{ }

    @A
    interface I1{}

    @B
    @C
    static class C1{}

    static class C2 extends C1 implements I1{}

    public static void main(String[] args) {
        for (Annotation annotation : C2.class.getAnnotations()) {
            System.out.println(annotation);
        }
    }
}
```

e. @Repeatable注解【1.8之后提出的注解】

如果一个注解需要重复使用，则需要在该注解上添加@Repeatable注解，其中有一个value的值指定重复注解的父注解，或者叫容器注解。

- 定义父容器

```
package com.etoak.rst.utils;

import java.lang.annotation.*;

@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface MyMappers {
    MyMapper[] value();
}
```

- 为注解指定父容器

```
package com.etoak.rst.utils;

import java.lang.annotation.*;

@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface MyMappers {
```

```
    MyMapper[] value();
}
```

- 使用注解

```
package com.etoak.rst.utils;

import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

@MyMapper("abc")
@MyMapper("def")
public class TestRepeatable {

    @MyMappers(value={@MyMapper("aaa"),@MyMapper("bbb")})
    public static void main(String[] args) throws NoSuchMethodException {
        Annotation[] annotations = TestRepeatable.class.getAnnotations();
        for (Annotation annotation : annotations) {
            System.out.println(annotation);
        }
        System.out.println("-------------");
        Method main = TestRepeatable.class.getDeclaredMethod("main", String[
        Annotation[] declaredAnnotations = main.getDeclaredAnnotations();
        for (Annotation declaredAnnotation : declaredAnnotations) {
            System.out.println(declaredAnnotation);
        }
    }
}
```

4. Spring中spring-core包中的@AliasFor注解

@AliasFor注解是用来给注解中的属性设置别名的。

```
package org.springframework.web.bind.annotation;

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;

import org.springframework.core.annotation.AliasFor;
import org.springframework.stereotype.Controller;


@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
@Controller
@ResponseBody
public @interface RestController {
//这里的意思是设置的RestController中的value值相当于设置的@Controller中的value
   @AliasFor(annotation = Controller.class)
   String value() default "";

}
package org.springframework.stereotype;

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;

import org.springframework.core.annotation.AliasFor;


@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Controller {

  //这里的意思是设置的Controller中的value值就相当于设置了Component中的value值
   @AliasFor(annotation = Component.class)
   String value() default "";

}
```