# 自定义注解

参考内容：<https://www.cnblogs.com/acm-bingzi/p/javaAnnotation.html>

# [注解Annotation实现原理与自定义注解例子](https://www.cnblogs.com/acm-bingzi/p/javaAnnotation.html)

### 什么是注解？

      对于很多初次接触的开发者来说应该都有这个疑问？***Annontation***是Java5开始引入的新特征，中文名称叫**注解**。它提供了一种安全的类似注释的机制，用来将任何的信息或元数据（metadata）与程序元素（类、方法、成员变量等）进行关联。为程序的元素（类、方法、成员变量）加上更直观更明了的说明，这些说明信息是与程序的业务逻辑无关，并且供指定的工具或框架使用。Annontation像一种修饰符一样，应用于包、类型、构造方法、方法、成员变量、参数及本地变量的声明语句中。  
　　Java注解是附加在代码中的一些元信息，用于一些工具在编译、运行时进行解析和使用，起到说明、配置的功能。注解不会也不能影响代码的实际逻辑，仅仅起到辅助性的作用。包含在 java.lang.annotation 包中。

### 注解的用处：

      1、生成文档。这是最常见的，也是java 最早提供的注解。常用的有@param @return 等  
      2、跟踪代码依赖性，实现替代配置文件功能。比如Dagger 2依赖注入，未来java开发，将大量注解配置，具有很大用处;  
      3、在编译时进行格式检查。如@override 放在方法前，如果你这个方法并不是覆盖了超类方法，则编译时就能检查出。

### 注解的原理：

　　注解本质是一个继承了Annotation的特殊接口，其具体实现类是Java运行时生成的动态代理类。而我们通过反射获取注解时，返回的是Java运行时生成的动态代理对象$Proxy1。通过代理对象调用自定义注解（接口）的方法，会最终调用AnnotationInvocationHandler的invoke方法。该方法会从memberValues这个Map中索引出对应的值。而memberValues的来源是Java常量池。

### 元注解：

java.lang.annotation提供了四种元注解，专门注解其他的注解（在自定义注解的时候，需要使用到元注解）：  
   @Documented –注解是否将包含在JavaDoc中  
   @Retention –什么时候使用该注解  
   @Target –注解用于什么地方  
   @Inherited – 是否允许子类继承该注解

  1.）@Retention– 定义该注解的生命周期  
  ●   RetentionPolicy.SOURCE : 在编译阶段丢弃。这些注解在编译结束之后就不再有任何意义，所以它们不会写入字节码。@Override, @SuppressWarnings都属于这类注解。  
  ●   RetentionPolicy.CLASS : 在类加载的时候丢弃。在字节码文件的处理中有用。注解默认使用这种方式  
  ●   RetentionPolicy.RUNTIME : 始终不会丢弃，运行期也保留该注解，因此可以使用反射机制读取该注解的信息。我们自定义的注解通常使用这种方式。

  2.）Target – 表示该注解用于什么地方。默认值为任何元素，表示该注解用于什么地方。可用的ElementType参数包括  
  ● ElementType.CONSTRUCTOR:用于描述构造器  
  ● ElementType.FIELD:成员变量、对象、属性（包括enum实例）  
  ● ElementType.LOCAL\_VARIABLE:用于描述局部变量  
  ● ElementType.METHOD:用于描述方法  
  ● ElementType.PACKAGE:用于描述包  
  ● ElementType.PARAMETER:用于描述参数  
  ● ElementType.TYPE:用于描述类、接口(包括注解类型) 或enum声明

 3.)@Documented–一个简单的Annotations标记注解，表示是否将注解信息添加在java文档中。

 4.)@Inherited – 定义该注释和子类的关系  
     @Inherited 元注解是一个标记注解，@Inherited阐述了某个被标注的类型是被继承的。如果一个使用了@Inherited修饰的annotation类型被用于一个class，则这个annotation将被用于该class的子类。

### 常见标准的Annotation：

  1.）Override  
      java.lang.Override是一个标记类型注解，它被用作标注方法。它说明了被标注的方法重载了父类的方法，起到了断言的作用。如果我们使用了这种注解在一个没有覆盖父类方法的方法时，java编译器将以一个编译错误来警示。  
  2.）Deprecated  
     Deprecated也是一种标记类型注解。当一个类型或者类型成员使用@Deprecated修饰的话，编译器将不鼓励使用这个被标注的程序元素。所以使用这种修饰具有一定的“延续性”：如果我们在代码中通过继承或者覆盖的方式使用了这个过时的类型或者成员，虽然继承或者覆盖后的类型或者成员并不是被声明为@Deprecated，但编译器仍然要报警。  
 3.）SuppressWarnings  
     SuppressWarning不是一个标记类型注解。它有一个类型为String[]的成员，这个成员的值为被禁止的警告名。对于javac编译器来讲，被-Xlint选项有效的警告名也同样对@SuppressWarings有效，同时编译器忽略掉无法识别的警告名。  
　　@SuppressWarnings("unchecked")

### 自定义注解：

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  
PS:自定义注解需要使用到元注解

### 自定义注解实例：

FruitName.java

[复制代码](javascript:void(0);)

1 import java.lang.annotation.Documented;

2 import java.lang.annotation.Retention;

3 import java.lang.annotation.Target;

4 import static java.lang.annotation.ElementType.FIELD;

5 import static java.lang.annotation.RetentionPolicy.RUNTIME;

6

7 /\*\*

8 \* 水果名称注解

9 \*/

10 @Target(FIELD)

11 @Retention(RUNTIME)

12 @Documented

13 public @interface FruitName {

14 String value() default "";

15 }

[复制代码](javascript:void(0);)

FruitColor.java

[复制代码](javascript:void(0);)

1 import java.lang.annotation.Documented;

2 import java.lang.annotation.Retention;

3 import java.lang.annotation.Target;

4 import static java.lang.annotation.ElementType.FIELD;

5 import static java.lang.annotation.RetentionPolicy.RUNTIME;

6

7 /\*\*

8 \* 水果颜色注解

9 \*/

10 @Target(FIELD)

11 @Retention(RUNTIME)

12 @Documented

13 public @interface FruitColor {

14 /\*\*

15 \* 颜色枚举

16 \*/

17 public enum Color{ BLUE,RED,GREEN};

18

19 /\*\*

20 \* 颜色属性

21 \*/

22 Color fruitColor() default Color.GREEN;

23

24 }

[复制代码](javascript:void(0);)

FruitProvider.java

[复制代码](javascript:void(0);)

1 import java.lang.annotation.Documented;

2 import java.lang.annotation.Retention;

3 import java.lang.annotation.Target;

4 import static java.lang.annotation.ElementType.FIELD;

5 import static java.lang.annotation.RetentionPolicy.RUNTIME;

6

7

8 /\*\*

9 \* 水果供应者注解

10 \*/

11 @Target(FIELD)

12 @Retention(RUNTIME)

13 @Documented

14 public @interface FruitProvider {

15 /\*\*

16 \* 供应商编号

17 \*/

18 public int id() default -1;

19

20 /\*\*

21 \* 供应商名称

22 \*/

23 public String name() default "";

24

25 /\*\*

26 \* 供应商地址

27 \*/

28 public String address() default "";

29 }

[复制代码](javascript:void(0);)

FruitInfoUtil.java

[复制代码](javascript:void(0);)

1 import java.lang.reflect.Field;

2

3 /\*\*

4 \* 注解处理器

5 \*/

6 public class FruitInfoUtil {

7 public static void getFruitInfo(Class<?> clazz){

8

9 String strFruitName=" 水果名称：";

10 String strFruitColor=" 水果颜色：";

11 String strFruitProvicer="供应商信息：";

12

13 Field[] fields = clazz.getDeclaredFields();

14

15 for(Field field :fields){

16 if(field.isAnnotationPresent(FruitName.class)){

17 FruitName fruitName = (FruitName) field.getAnnotation(FruitName.class);

18 strFruitName=strFruitName+fruitName.value();

19 System.out.println(strFruitName);

20 }

21 else if(field.isAnnotationPresent(FruitColor.class)){

22 FruitColor fruitColor= (FruitColor) field.getAnnotation(FruitColor.class);

23 strFruitColor=strFruitColor+fruitColor.fruitColor().toString();

24 System.out.println(strFruitColor);

25 }

26 else if(field.isAnnotationPresent(FruitProvider.class)){

27 FruitProvider fruitProvider= (FruitProvider) field.getAnnotation(FruitProvider.class);

28 strFruitProvicer=" 供应商编号："+fruitProvider.id()+" 供应商名称："+fruitProvider.name()+" 供应商地址："+fruitProvider.address();

29 System.out.println(strFruitProvicer);

30 }

31 }

32 }

33 }

[复制代码](javascript:void(0);)

Apple.java

[复制代码](javascript:void(0);)

1 import test.FruitColor.Color;

2

3 /\*\*

4 \* 注解使用

5 \*/

6 public class Apple {

7

8 **@FruitName**("Apple")

9 private String appleName;

10

11  **@FruitColor**(fruitColor=Color.RED)

12 private String appleColor;

13

14  **@FruitProvider**(id=1,name="陕西红富士集团",address="陕西省西安市延安路89号红富士大厦")

15 private String appleProvider;

16

17 public void setAppleColor(String appleColor) {

18 this.appleColor = appleColor;

19 }

20 public String getAppleColor() {

21 return appleColor;

22 }

23

24 public void setAppleName(String appleName) {

25 this.appleName = appleName;

26 }

27 public String getAppleName() {

28 return appleName;

29 }

30

31 public void setAppleProvider(String appleProvider) {

32 this.appleProvider = appleProvider;

33 }

34 public String getAppleProvider() {

35 return appleProvider;

36 }

37

38 public void displayName(){

39 System.out.println("水果的名字是：苹果");

40 }

41 }

[复制代码](javascript:void(0);)

FruitRun.java

[复制代码](javascript:void(0);)

1 /\*\*

2 \* 输出结果

3 \*/

4 public class FruitRun {

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

6 FruitInfoUtil.getFruitInfo(Apple.class);

7 }

8 }

[复制代码](javascript:void(0);)

运行结果是：

 水果名称：Apple  
 水果颜色：RED  
 供应商编号：1 供应商名称：陕西红富士集团 供应商地址：陕西省西安市延安路89号红富士大厦

# 二 Java关键字详解

1. default：
2. Class<?>与Class<T>

# 三、SpringMVC的拦截器（Interceptor）和过滤器（Filter）的区别与联系

博客地址：<https://blog.csdn.net/xiaoyaotan_111/article/details/53817918>

### 一 简介

**（1）过滤器：**

依赖于servlet容器。在实现上基于函数回调，可以对几乎所有请求进行过滤，但是缺点是一个过滤器实例只能在容器初始化时调用一次。使用过滤器的目的是用来做一些过滤操作，获取我们想要获取的数据，比如：在过滤器中修改字符编码；在过滤器中修改HttpServletRequest的一些参数，包括：过滤低俗文字、危险字符等

关于过滤器的一些用法可以参考我写过的这些[文章](http://www.07net01.com/2015/07/860262.html)：

* 继承HttpServletRequestWrapper以实现在Filter中修改HttpServletRequest的参数：https://www.zifangsky.cn/677.html
* 在SpringMVC中使用过滤器（Filter）过滤容易引发XSS的危险字符：https://www.zifangsky.cn/683.html

**（2）拦截器：**

依赖于web框架，在SpringMVC中就是依赖于SpringMVC框架。在实现上基于[Java](http://www.07net01.com/tags-Java-0.html)的反射机制，属于面向切面[编程](http://www.07net01.com/)（AOP）的一种运用。由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截，对其他的一些比如直接访问静态资源的请求则没办法进行拦截处理

关于过滤器的一些用法可以参考我写过的这些文章：

* 在SpringMVC中使用拦截器（interceptor）拦截CSRF攻击（修）：https://www.zifangsky.cn/671.html
* SpringMVC中使用Interceptor+[cookie](http://www.07net01.com/tags-cookie-0.html)实现在一定天数之内自动登录：https://www.zifangsky.cn/700.html

### 二 多个过滤器与拦截器的代码执行顺序

如果在一个[项目](http://www.07net01.com/tags-%E9%A1%B9%E7%9B%AE-0.html)中仅仅只有一个拦截器或者过滤器，那么我相信相对来说理解起来是比较容易的。但是我们是否思考过：如果一个项目中有多个拦截器或者过滤器，那么它们的执行顺序应该是什么样的？或者再复杂点，一个项目中既有多个拦截器，又有多个过滤器，这时它们的执行顺序又是什么样的呢？

### 三 配置说明