# Spring，java等注解annotation

**Spring注解**

**1.Annotation的一般形式是 :**

public @interface MyAnnotation {

String value() default "hahaha";

}

**2.@component：**

把普通pojo实例化到spring容器中，相当于配置文件中的 <bean id="" class=""/>

**java注解**

1. **元注解（meta-annotation）**

元注解的作用就是负责注解其他注解。Java5.0定义了4个标准的meta-annotation类型，它们被用来提供对其它 annotation类型作说明。Java5.0定义的元注解：这些类型和它们所支持的类在java.lang.annotation包中可以找到

1.@Target  
 2.@Retention  
　　　3.@Documented  
　　　4.@Inherited

1. **@Target：**

作用：用于描述注解的使用范围（即：被描述的注解可以用在什么地方）。

取值(ElementType)有：

　1.CONSTRUCTOR:用于描述构造器  
　　　　2.FIELD:用于描述域  
　　　　3.LOCAL\_VARIABLE:用于描述局部变量  
　　　　4.METHOD:用于描述方法  
　　　　5.PACKAGE:用于描述包  
　　　　6.PARAMETER:用于描述参数  
　　　　7.TYPE:用于描述类、接口(包括注解类型) 或enum声明

例如：

@Target(ElementType.FIELD)

public @interface NoDBColumn {}  
**2.@Retention：**

**作用**：表示需要在什么级别保存该注释信息，用于描述注解的生命周期（即：被描述的注解在什么范围内有效）。

@Retention定义了该Annotation被保留的时间长短：某些Annotation仅出现在源代码中，而被编译器丢弃；而另一些却被编译在class文件中；编译在class文件中的Annotation可能会被虚拟机忽略，而另一些在class被装载时将被读取（请注意并不影响class的执行，因为Annotation与class在使用上是被分离的）。使用这个meta-Annotation可以对 Annotation的“生命周期”限制。

**取值（RetentionPoicy）有**：

1.SOURCE:在源文件中有效（即源文件保留）  
　　2.CLASS:在class文件中有效（即class保留）  
　　3.RUNTIME:在运行时有效（即运行时保留）

**例子：**

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)public @interface Column {

public String name() default "fieldName";

public String setFuncName() default "setField";

public String getFuncName() default "getField";

public boolean defaultDBValue() default false;

}

Column注解的的RetentionPolicy的属性值是RUTIME,这样注解处理器可以通过反射，获取到该注解的属性值，从而去做一些运行时的逻辑处理  
**3.@Inherited：**

**4.@Documented**

**@**Documented用于描述其它类型的annotation应该被作为被标注的程序成员的公共API，因此可以被例如javadoc此类的工具文档化。Documented是一个标记注解，没有成员。

Documented 注解表明这个注解应该被 javadoc工具记录. 默认情况下,javadoc是不包括注解的. 但如果声明注解时指定了 @Documented,则它会被 javadoc 之类的工具处理, 所以注解类型信息也会被包括在生成的文档中. 示例6进一步演示了使用 @Documented

***注释：javadoc是Sun公司提供的一个技术，它从程序[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81/3969" \t "https://baike.baidu.com/item/javadoc/_blank)中抽取类、方法、成员等注释形成一个和源代码配套的API帮助文档。也就是说，只要在编写程序时以一套特定的标签作注释，在程序编写完成后，通过Javadoc就可以同时形成程序的开发文档了。javadoc命令是用来生成自己[API](https://baike.baidu.com/item/API/10154" \t "https://baike.baidu.com/item/javadoc/_blank)文档的，使用方式：使用命令行在目标文件所在目录输入javadoc +文件名.java。***

***在注释中出现以@开头东东被称之为Javadoc文档标记，是JDK定义好的，如@author、@version、@since、@see、@link、@code、@param、@return、@exception、@throws等。***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **标签** | **说明** | **JDK 1.1 doclet** | **标准doclet** | **标签类型** |
| **@author 作者** | **作者标识** | **√** | **√** | **包、 类、接口** |
| **@version 版本号** | **版本号** | **√** | **√** | **包、 类、接口** |
| **@param 参数名 描述** | **方法的入参名及描述信息，如入参有特别要求，可在此注释。** | **√** | **√** | **构造函数、 方法** |
| **@return 描述** | **对函数返回值的注释** | **√** | **√** | **方法** |
| **@deprecated 过期文本** | **标识随着程序版本的提升，当前API已经过期，仅为了保证兼容性依然存在，以此告之开发者不应再用这个API。** | **√** | **√** | **包、类、接口、值域、构造函数、 方法** |
| **@throws异常类名** | **构造函数或方法所会抛出的异常。** |  | **√** | **构造函数、 方法** |
| **@exception 异常类名** | **同@throws。** | **√** | **√** | **构造函数、 方法** |
| **@see 引用** | **查看相关内容，如类、方法、变量等。** | **√** | **√** | **包、类、接口、值域、构造函数、 方法** |
| **@since 描述文本** | **API在什么程序的什么版本后开发支持。** | **√** | **√** | **包、类、接口、值域、构造函数、 方法** |
| **{@link包.类#成员 标签}** | **链接到某个特定的成员对应的文档中。** |  | **√** | **包、类、接口、值域、构造函数、 方法** |
| **{@value}** | **当对常量进行注释时，如果想将其值包含在文档中，则通过该标签来引用常量的值。** |  | **√(JDK1.4)** | **静态值域** |