# 注解概览

注解功能使得代码中的声明部分都可以添加结构化、机器可读的元数据， 注解的目标可以是类、方法、函数、参数、属性、类常量。

通过 [反射 API](mk:@MSITStore:C:\\Users\\张卫航\\Desktop\\php_enhanced_zh.chm::/res/book.reflection.html) 可在运行时获取注解所定义的元数据。 因此注解可以成为直接嵌入代码的配置式语言。

通过注解的使用，在应用中实现功能、使用功能可以相互解耦。

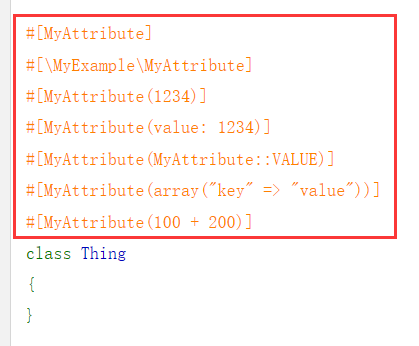
某种程度上讲，它可以和接口（interface）与其实现（implementation）相比较。 但接口与实现是代码相关的，注解则与声明额外信息和配置相关。

接口可以通过类来实现，而注解也可以声明到方法、函数、参数、属性、类常量中。 因此它们比接口更灵活。

# 注解语法

注解语法包含以下几部分。 首先，注解声明总是以 #[ 开头，以 ] 结尾来包围。

内部则是一个或以逗号包含的多个注解。 注解的名称按 [使用命名空间：基础](mk:@MSITStore:C:\\Users\\张卫航\\Desktop\\php_enhanced_zh.chm::/res/language.namespaces.basics.html) 章节中描述，可以是非限定、限定、完全限定的名称。 注解的参数是可以选的，以常见的括号()包围。 注解的参数可以是字面值或者常量表达式。 它同时接受位置参数和命名参数两种语法。



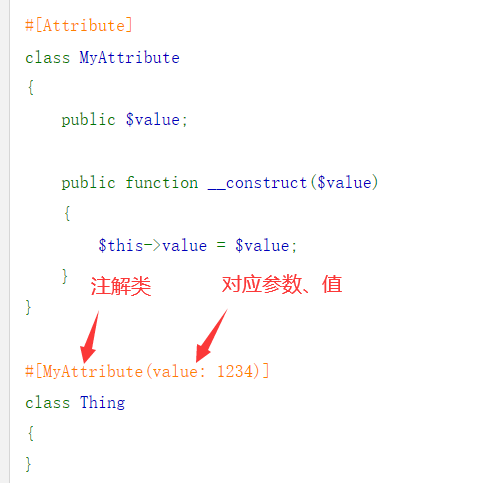
# 使用反射 API 读取注解

反射 API 提供了 **getAttributes()** 方法， 类、方法、函数、参数、属性、类常量的反射对象可通过它获取相应的注解。

该方法返回了 **ReflectionAttribute** 实例的数组， 可用于查询注解名称、参数、也可以实例化一个注解。

实例和反射注解的分离使得程序员增加了在丢失反射类、类型错误、丢失参数等情况下的处理能力，也能处理错误。 只有调用 **newInstance()** 后，注解类的对象才会以验证过匹配的参数来实例化。

## 例子

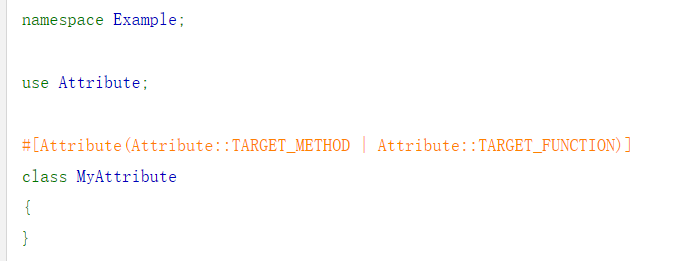




# 声明注解类

虽然没有严格要求，推荐为每个注解创建一个实际的类。

## 简单的 Attribute 类



# 总结

## 注释和注解的区别

注解 ：参与代码编译，以#[开头的。它是给应用程序看的，单独使用注解毫无意义，一定要跟工具一起使用，这个所谓的工具实际就是能读懂注解的应用程序。  
注释 ：对代码没有影响。对代码起到解释、说明的作用；

在laravel框架中有使用到，感觉业务层基本不会使用到，无需深究。