注解

1 能够读懂别人写的代码，特别是框架相关的代码
2 让编程更加整洁，代码更加清晰
3 让别人高看一眼

。注解（Annotation） 为我们在代码中添加信息提供了一种形式化的方法，是我们可以在稍后某个时刻方便地使用这些数据（通过 解析注解 来使用这些数据），常见的作用有以下几种：

- 生成文档。这是最常见的，也是java 最早提供的注解。常用的有@see @param @return 等
- 跟踪代码依赖性，实现替代配置文件功能。比较常见的是spring 2.5 开始的基于注解配置。作用就是减少配置。现在的框架基本都使用了这种配置来减少配置文件的数量。
- 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。

包 **Java.lang.annotation** 中包含所有定义自定义注解所需用到的原注解和接口。

全面解析注解

java的常见注解

注解分类

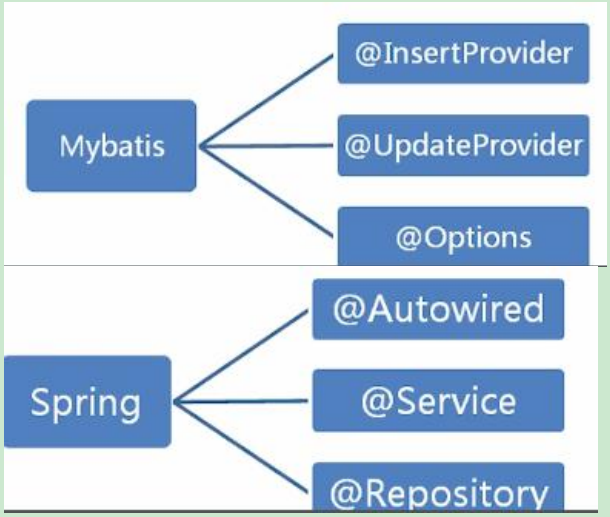
自定义注解

@Override

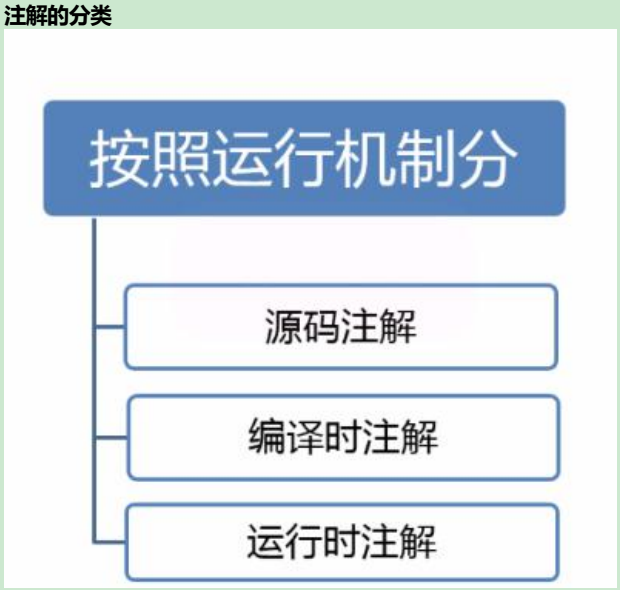
@Deprecated

@SuppressWarnings

常见的第三方注解



注解的分类



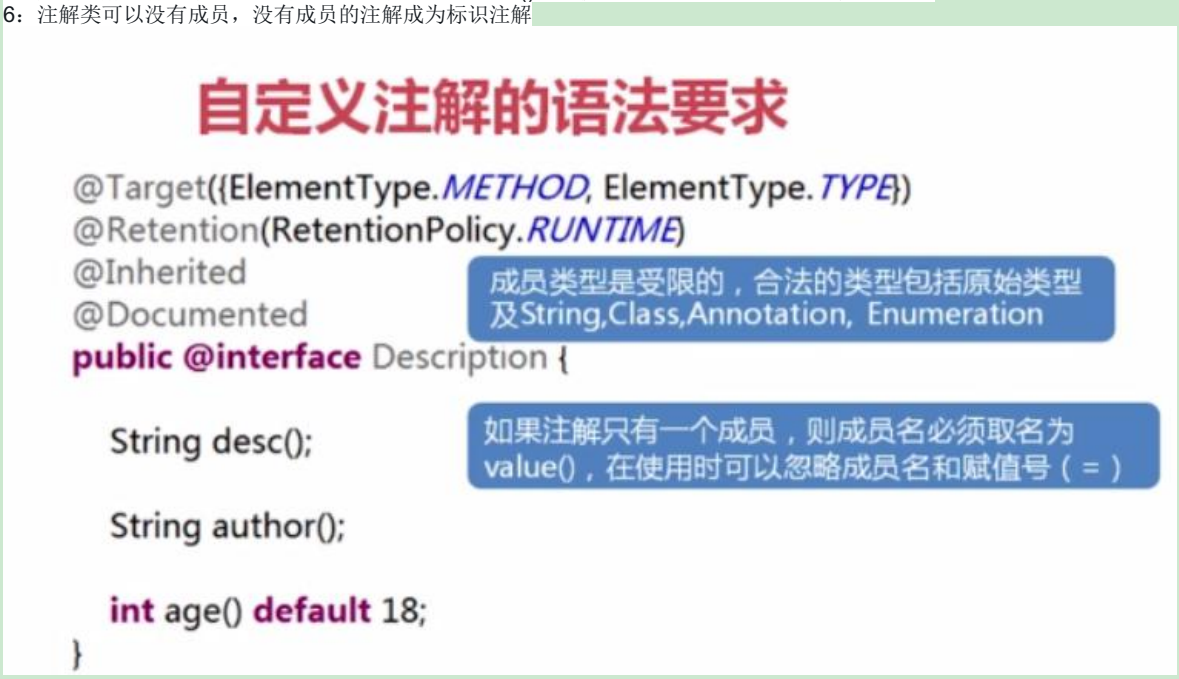
源码注解	注解只在源码中存在，编译成class文件就不存在了
编译时注解	注解在源码和.class文件中都存在

运行时注解	在运行阶段还起作用的，甚至会影响逻辑的运行

还有一类是**元注解** 就是注解的注解 (注解的注解)

自定义注解的语法要求

@自定义注解的语法要求：
1：使用@interface关键字定义注解
2：成员方法以无异常的方式声明
3：可以使用default为成员方法指定一个默认值
4：成员的类型是有限制的，合法的成员类型包括原始类型/String/Class/Annotation/Enumeration
5：如果注解只有一个成员，则成员名必须取名为value()，在使用时可以忽略成员名和赋值号（=）
6：注解类可以没有成员，没有成员的注解成为标识注解



注解的注解（元注解）

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

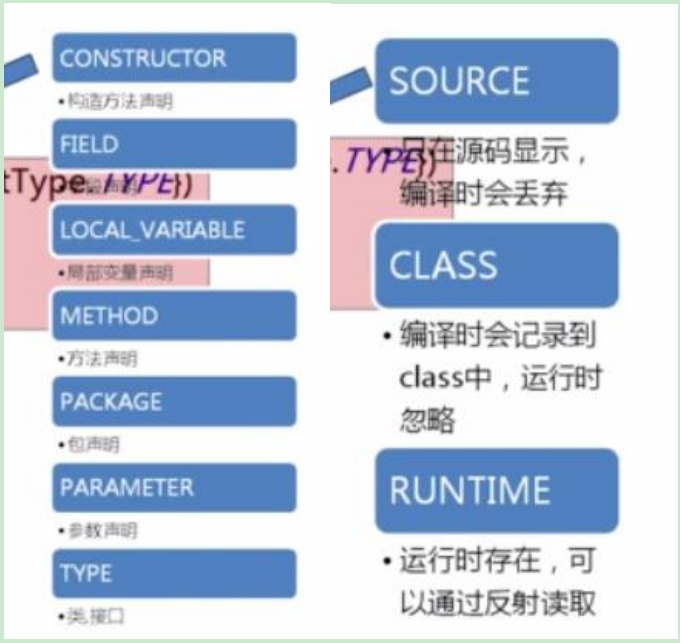
@Target说明了Annotation所修饰的对象范围：Annotation可被用于 packages、types（类、接口、枚举、Annotation类型）、类型成员（方法、构造方法、成员变量、枚举值）、方法参数和本地变量（如循环变量、catch参数）。在Annotation类型的声明中使用了target可更加明晰其修饰的目标。

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）

@Deprecated

这个注释是一个标记注释。所谓标记注释，就是在源程序中加入这个标记后，并不影响程序的编译，但有时编译器会显示一些警告信息。

编译器会告诉你这个方法已经淘汰了 在eclipse中会出现一道横线滑过这个方法名



可以用在包的声明上 还有参数上 类或接口上

使用自定义注解

解析注解

java 注解annotation的使用，以及反射如何获取注解

主要还是通过

反射加载时用getAnnotation方法来获取注解

概念：
通过反射获取类，函数或成员上的运行时的注解信息，实现动态控制程序运行的逻辑
@Deprecated注解的程序元素，不鼓励程序员使用这样的元素，

@Inherited对implements不起作用，对extends起作用（只会 继承类上面注解，而不会继承该类方法中的注解）。
通过注解可以写出

1. 要用好注解，必须熟悉java 的反射机制，从上面的例子可以看出，注解的解析完全依赖于反射。

2. 不要滥用注解。平常我们编程过程很少接触和使用注解，只有做设计，且不想让设计有过多的配置时。