Java基础加强-注解Annotation

# 注解概述

## 注解介绍

注解（Annotation），也叫元数据。一种代码级别的说明。它是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以用于创建文档，跟踪代码中的依赖性，甚至执行基本编译时检查。注解是以‘@注解名’在代码中存在的

它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。你可以在编译时选择代码里的注解是否只存在于源代码级，或者它也能在class文件、或者运行时中出现（SOURCE/CLASS/RUNTIME）。

## 注解作用

如果要对于元数据的作用进行分类，还没有明确的定义，不过我们可以根据它所起的作用，大致可分为三类：

编写文档：通过代码里标识的元数据生成文档。

代码分析：通过代码里标识的元数据对代码进行分析。

编译检查：通过代码里标识的元数据让[编译器](http://baike.baidu.com/view/487018.htm)能实现基本的编译检查

在现在开发中使用注解，一般是用于将注解替换配置文件。（Xml配置文件）

# Java中基本内置注解

## @Override

它的作用是对覆盖超类中方法的方法进行标记，如果被标记的方法并没有实际覆盖超类中的方法，则编译器会发出错误警告。

注意事项:

对于接口中的方法重写，[在jdk1.5时@Override](mailto:在jdk1.5时@Override)它是会报错.

在jdk1.6后的版本就可以描述接口与类之间的重写

## @Deprecated

它的作用是对不应该再使用的方法添加注解，当编程人员使用这些方法时，将会在编译时显示提示信息

问题:什么时候方法是过时的?

当前版本中方法它存在隐患，在后续版本中对其进行了补充，这时前一个版本中的方法就会标注成过时的。

## @SuppressWarnings

它的作用是去掉程序中的警告.

其参数有：

deprecation，使用了过时的类或方法时的警告

unchecked，执行了未检查的转换时的警告

fallthrough，当 switch 程序块直接通往下一种情况而没有 break 时的警告

path，在类路径、源文件路径等中有不存在的路径时的警告

serial，当在可序列化的类上缺少serialVersionUID 定义时的警告

finally ，任何 finally 子句不能正常完成时的警告

all，关于以上所有情况的警告

# 自定义注解

## 注解声明

通过上面的三个注解的源代码可以发现，要声明一个注解通过 @interface

声明一个注解格式 @interface 注解名{}

## 注解本质分析

分析一下注解的本质:将其.class文件找到，反编译. 可以使用javap命令或反编译工具。

@interface MyAnnoation{}

反编译后的结果

interface MyAnnotation extends Annotation

{

}

结论:注解本质上就是一个接口。它扩展了java.lang.annotation.Annotation接口;

在java中所有注解都是Annotation接口的子接口。

## 注解成员

注解本质上就是一个接口，那么它也可以有属性和方法。

但是接口中的属性是 public static final的，在注解中注解没有什么意义。

在开发中注解中经常存在的是方法。而在注解中叫做注解的属性.

## 自定义注解-属性操作

### 注解属性类型

1.基本类型 byte short int long float double char boolean

2.String

3.枚举类型

4.注解类型

5.Class类型

6.以上类型的一维数组类型

### 注解属性的使用

1.如果一个注解有属性，那么在使用注解时，要对属性进行赋值操作.

例如:@MyAnnotation(st = "aaa")

2.如果一个注解的属性有多个，都需要赋值，使用","分开属性.

@MyAnnotation(st = "aaa",i=10)

3.也可以给属性赋默认值

double d() default 1.23;

如果属性有默认值，在使用注解时，就可以不用为属性赋值。

4.如果属性是数组类型

1.可以直接使用 属性名={值1,值2,。。。}方式,例如

@MyAnnotation(st = "aaa",i=10,sts={"a","b"})

2.如果数组的值只有一个也可以写成下面方式

@MyAnnotation(st = "aaa",i=10,sts="a")

注意sts属性它是数组类型，也就是说，只有一个值时，可以省略"{}"

5.对于属性名称 value的操作.

1.如果属性名称叫value,那么在使用时，可以省略属性名称

@MyAnnotation("hello")

2.如果有多个属性，都需要赋值，其中一个叫value,这时，必须写属性名称

@MyAnnotation(value="hello",i=10)

3.如果属性名称叫value,它的类型是数组类型.

1.只有这个value属性

可以直接赋值，不能写属性名称,但是，如果只有一个值

@MyAnnotation({"abc"})或 @MyAnnotation("abc")

但是如果有多个值

@MyAnnotation({"abc","def"})

2.如果有多个属性，属性名称叫value

所有属性都需要赋值，那么必须写属性名称.

## 自定义注解-元注解

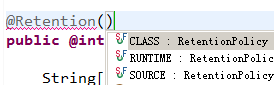
### 1.什么是元注解及其作用

用于修饰注解的注解，可以描述注解在什么范围及在什么阶段使用等

### 2.四个元注解介绍

#### @Retention

指定注解信息在哪个阶段存在 Source Class Runtime



SOURCE它对应着编译阶段，可以帮助我们进行检查。

CLASS 它对应解析执行阶段

RUNTIME 它对应着在JVM中

#### @Target

指定注解修饰目标对象类型 TYPE 类、接口 FIELD 成员变量 METHOD 方法



#### @Documented

使用该元注解修饰，该注解的信息可以生成到javadoc 文档中

#### @Inherited

如果一个注解使用该元注解修饰，应用注解目标类的子类会自动继承该注解

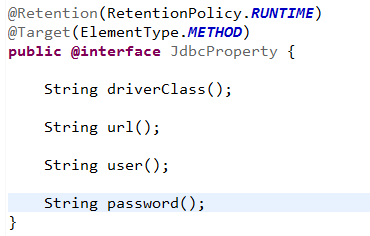
@Retention @Target 是自定义注解必须使用两个元注解，并且，@Retention它的值应该是RUNTIME,因为我们会结合反射技术来使用。 @Target我们一般使用TYPE或METHOD

## 案例-获取Connection连接数据库

目的：让注解具有功能，必须结合反射技术来应用。

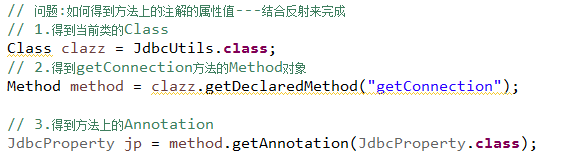
注解它可以替换配置文件。

第一步:创建注解



第二步:使用注解

我们可以通过getAnnotation()方法来获取注解对象



问题:使用注解可以替换配置文件，为什么要替换?

配置文件中的信息，它会随着程序的变大，配置信息越来越多，不利于开发。

而将配置信息通过注解来替换，便于开发与阅读。