**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：  
　　　　1.@Target,  
　　　　2.@Retention,  
　　　　3.@Documented,  
　　　　4.@Inherited  
　　这些类型和它们所支持的类在java.lang.annotation包中可以找到。下面我们看一下每个元注解的作用和相应分参数的使用说明。

**@Target：**

　　　@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）**

**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

　　使用实例：

@Target(ElementType.TYPE)

public @interface Table {

/\*\*

\* 数据表名称注解，默认值为类名称

\* @return

\*/

public String tableName() default "className";

}

@Target(ElementType.FIELD)

public @interface NoDBColumn {

}

**@Retention：**

**@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。

**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）**

**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）

　　Retention meta-annotation类型有唯一的value作为成员，它的取值来自java.lang.annotation.RetentionPolicy的枚举类型值。具体实例如下：

@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

**@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。

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface Column {

public String name() default "fieldName";

public String setFuncName() default "setField";

public String getFuncName() default "getField";

public boolean defaultDBValue() default false;

}

**@Inherited：**

@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。

　　注意：@Inherited annotation类型是被标注过的class的子类所继承。类并不从它所实现的接口继承annotation，方法并不从它所重载的方法继承annotation。

　　当@Inherited annotation类型标注的annotation的Retention是RetentionPolicy.RUNTIME，则反射API增强了这种继承性。如果我们使用java.lang.reflect去查询一个@Inherited annotation类型的annotation时，反射代码检查将展开工作：检查class和其父类，直到发现指定的annotation类型被发现，或者到达类继承结构的顶层。

　　实例代码：

@Inherited

public @interface Greeting {

public enum FontColor{ BULE,RED,GREEN};

String name();

FontColor fontColor() default FontColor.GREEN;

}

**自定义注解：**

　　使用@interface自定义注解时，自动继承了java.lang.annotation.Annotation接口，由编译程序自动完成其他细节。在定义注解时，不能继承其他的注解或接口。@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型（返回值类型只能是基本类型、Class、String、enum）。可以通过default来声明参数的默认值。

**定义注解格式：**  
　　public @interface 注解名 {定义体}

**注解参数的可支持数据类型：**

　　　　1.所有基本数据类型（int,float,boolean,byte,double,char,long,short)  
　　　　2.String类型  
　　　　3.Class类型  
　　　　4.enum类型  
　　　　5.Annotation类型  
　　　　6.以上所有类型的数组

　　Annotation类型里面的参数该怎么设定:   
　　第一,只能用public或默认(default)这两个访问权修饰.例如,String value();这里把方法设为defaul默认类型；　 　  
　　第二,参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.例如,String value();这里的参数成员就为String;　　  
　　第三,如果只有一个参数成员,最好把参数名称设为"value",后加小括号.例:下面的例子FruitName注解就只有一个参数成员。

简单的自定义注解和使用注解实例：

package annotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* 水果名称注解

\* @author peida

\*

\*/

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface FruitName {

String value() default "";

}

package annotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* 水果颜色注解

\* @author peida

\*

\*/

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface FruitColor {

/\*\*

\* 颜色枚举

\* @author peida

\*

\*/

public enum Color{ BULE,RED,GREEN};

/\*\*

\* 颜色属性

\* @return

\*/

Color fruitColor() default Color.GREEN;

}

package annotation;

import annotation.FruitColor.Color;

public class Apple {

@FruitName("Apple")

private String appleName;

@FruitColor(fruitColor=Color.RED)

private String appleColor;

public void setAppleColor(String appleColor) {

this.appleColor = appleColor;

}

public String getAppleColor() {

return appleColor;

}

public void setAppleName(String appleName) {

this.appleName = appleName;

}

public String getAppleName() {

return appleName;

}

public void displayName(){

System.out.println("水果的名字是：苹果");

}

}

**注解元素的默认值：**

注解元素必须有确定的值，要么在定义注解的默认值中指定，要么在使用注解时指定，非基本类型的注解元素的值不可为null。因此, 使用空字符串或0作为默认值是一种常用的做法。这个约束使得处理器很难表现一个元素的存在或缺失的状态，因为每个注解的声明中，所有元素都存在，并且都具有相应的值，为了绕开这个约束，我们只能定义一些特殊的值，例如空字符串或者负数，一次表示某个元素不存在，在定义注解时，这已经成为一个习惯用法。例如：

package annotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* 水果供应者注解

\* @author peida

\*

\*/

@Target(ElementType.FIELD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface FruitProvider {

/\*\*

\* 供应商编号

\* @return

\*/

public int id() default -1;

/\*\*

\* 供应商名称

\* @return

\*/

public String name() default "";

/\*\*

\* 供应商地址

\* @return

\*/

public String address() default "";

}

定义了注解，并在需要的时候给相关类，类属性加上注解信息，如果没有响应的注解信息处理流程，注解可以说是没有实用价值。如何让注解真真的发挥作用，主要就在于注解处理方法，下一步我们将学习注解信息的获取和处理！

**二、注解的使用：**

 第一步：新建一个annotation，名字为：MyAnnotation.java。

package com.dragon.test.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* Created by gmq on 2015/9/10.

\*/

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface MyAnnotation

{

String hello () default "hello";

String world();

}

第二步：建立一个MyTest.java 来使用上面的annotation。

package com.dragon.test.annotation;

/\*\*

\* Created by gmq on 2015/9/10.

\*/

public class MyTest

{

@MyAnnotation(hello = "Hello,Beijing",world = "Hello,world")

public void output() {

System.out.println("method output is running ");

}

}

第三步：用反射机制来调用注解中的内容

package com.dragon.test.annotation;

import java.lang.annotation.Annotation;

import java.lang.reflect.Method;

/\*\*

\* 用反射机制来调用注解中的内容

\* Created by gmq on 2015/9/10.

\*/

public class MyReflection

{

public static void main(String[] args) throws Exception

{

// 获得要调用的类

Class<MyTest> myTestClass = MyTest.class;

// 获得要调用的方法，output是要调用的方法名字，new Class[]{}为所需要的参数。空则不是这种

Method method = myTestClass.getMethod("output", new Class[]{});

// 是否有类型为MyAnnotation的注解

if (method.isAnnotationPresent(MyAnnotation.class))

{

// 获得注解

MyAnnotation annotation = method.getAnnotation(MyAnnotation.class);

// 调用注解的内容

System.out.println(annotation.hello());

System.out.println(annotation.world());

}

System.out.println("----------------------------------");

// 获得所有注解。必须是runtime类型的

Annotation[] annotations = method.getAnnotations();

for (Annotation annotation : annotations)

{

// 遍历所有注解的名字

System.out.println(annotation.annotationType().getName());

}

}

}

输出：

Hello,Beijing  
Hello,world  
----------------------------------  
com.dragon.test.annotation.MyAnnotation