概述
Java5.0注解可以看做Javadoc和Xdoclet标签的延伸和发展，在Java5.0中可以自定义这些标签，并通过Java语言的反射机制获取类中标注的注解，完成特定的功能。

注解是代码的附属信息，它遵循一个基本的原则：注解不能直接干扰程序代码的运行，无论增加或者删除注解，代码都能正常运行。

Java语言解释器会忽略这些注解，而由第三方工具负责对注解进行处理。 第三方工具可以利用代码中的注解间接控制程序代码的运行，它们通过Java反射机制读取注解的信息，并根据这些信息更改目标程序的逻辑。

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：

- @Target
- @Retention
- @Documented
- @Inherited

这几个类都在java.lang.annotation包中
@Target
@Target主要说明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： @Target(ElementType.XXX)取值如下

- TYPE:类、接口、注解类、Enum声明处，相应的注解称为类型注解
- FIELD:类成员变量或者常量声明处，相应的注解被称为域值注解
- METHOD：方法处声明，相应的注解称为方法注解
- PARAMETER:参数声明处，相应的注解称为参数注解
- CONSTRUCTOR:构造函数声明处，相应的注解称为构造函数注解
- LOCAL_VARIABLE:局部变量声明处，相应的注解称为局域比纳凉注解
- PACKAGE:包声明处，相应的注解被称为包注解

```
/**
 * @Description: 注解@DataSource既可以加在方法上，也可以加在接口或者接口的实现类上
 */

@Target({ ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DataSource {
   // 和配置文件中 dynamicDatasourceMap中的key保持一致
   public static String PR_RB = "dataSourcePR";

   public static String DR_RB = "dataSourceDR";

   public static String PR_CC = "dataSourceCC";

   /**
    *
    *
    * @Title: name
    *
    * @Description: 如果仅标注@DataSource 默认为PR_RB数据库实例
    *
    * @return
    *
    * @return: String
    */
   String name() default DataSource.PR_RB;
}
```

@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（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）

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。

```
 * @author  Joshua Bloch
 * @since 1.5
 */
public enum RetentionPolicy {
    /**
     * Annotations are to be discarded by the compiler.
     */
    SOURCE,

    /**
     * Annotations are to be recorded in the class file by the compiler
     * but need not be retained by the VM at run time.  This is the default
     * behavior.
     */
    CLASS,

    /**
     * Annotations are to be recorded in the class file by the compiler and
     * retained by the VM at run time, so they may be read reflectively.
     *
     * @see java.lang.reflect.AnnotatedElement
     */
    RUNTIME
}
```


取值（RetentionPoicy）有：

- SOURCE:在源文件中有效（即源文件保留）,单对应的字节码文件将不再保留
- CLASS:在class文件中有效（即class保留），但类加载器加载字节码文件时不会将注解加载到JVM中，即运行期不能获取注解信息
- RUNTIME:在运行时有效（即运行时保留），注解信息在目标类加载到JVM后依然保留，在运行期可以通过反射机制读取类中的注解信息

比如：

```
@Target({ ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DataSource {
   .......
}
```

@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。

@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。

@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。

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

- 所有基本数据类型（int,float,boolean,byte,double,char,long,short)
- String类型
- Class类型
- enum类型
- Annotation类型
- 以上所有类型的数组

Annotation类型里面的参数设定规则：

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.

所有的注解都隐式继承与java.lang.annotation.Annotation，但注解不允许显示继承其他的接口。

如果自定义的注解类和目标类不在同一个包中，需要通过improt + 全类名 引用注解类。

在标注注解时，可以通过以下格式对注解成员进行赋值

@<注解名>(<成员名1>=<成员值1>,<成员名2>=<成员值2>,.....)

如果成员是数组类型，这可以通过{}进行赋值，比如 boolean数组的成员可以设置为{true,false,true}