详解Java编程中Annotation注解对象的使用方法

更新时间：2016年03月05日 08:56:00 作者：xyz_lmn 我要评论

这篇文章主要介绍了Java编程中Annotation注解对象的使用方法,注解以"@注解名"的方式被编写,与类、接口、枚举是在同一个层次，需要的朋友可以参考下

注解(也被称为元数据)为我们在代码中添加信息提供了一种形式化的方法，使我们可以在稍后某个时刻非常方便地使用这些数据。

1.基本语法

Java SE5内置三种标准注解

@Override:表示当前的方法定义将覆盖超类中的方法.如果你不小心拼写错误,或者方法签名对不上被覆盖的方法,编译器就会发出错误提示

@Deprecated:如果程序员使用了注解为它的元素,那么编译器就会发出警告信息

@SupperessWarnings:关闭不当的编译器警告信息.

大家感兴趣的内容

1 java使double保留两位小数的多方
2 JAVA8 十大新特性详解
3 java.net.SocketException: Conn
4 java写入文件的几种方法分享
5 Java环境变量的设置方法(图文教程
6 JAVA 十六进制与字符串的转换
7 java list用法示例详解
8 java中File类的使用方法
9 JavaWeb实现文件上传下载功能实例
10 Java中HashMap和TreeMap的区别深

最近更新的内容

Java SE5内置四种元注解

@Target:表示该注解可以用于什么地方.可能的ElementType参数包括:

1)CONSTRUCTOR:构造器的声明

2)FIELD:域声明(包括enum实例)

3)LOCAL_VARIABLE:局部变量声明

4)METHOD:方法声明

5)PACKAGE:包声明

6)PARAMETER:参数声明

7)TYPE:类、接口(包括注解类型)或enum声明

@Retention:表示需要在什么级别保存该注解信息.可选的RetentionPolicy参数包括:

1)SOURCE:注解将被编译器丢弃

2)CLASS:注解在class文件中可用,但会被VM丢弃

3)RUNTIME:VM将在运行期也保留注解,因此可以通过反射机制读取注解的信息

@Documented:将此注解包含在Javadoc中

@Inherited:允许子类继承父类中的注解

大多数时候,程序员主要是定义自己的注解，并编写自己的处理器来处理它们.

UseCase.java

```
1  package com;
2
3  import java.lang.annotation.ElementType;
4  import java.lang.annotation.Retention;
5  import java.lang.annotation.RetentionPolicy;
6  import java.lang.annotation.Target;
7
8  @Target(ElementType.METHOD)//用来定义你的注解将应用在什么地方,本处应用为方法
9  //用来定义该注解在哪一个级别可用,在源代码中(source)类文件中(class)或者运行时(runti
10 @Retention(RetentionPolicy.RUNTIME)
11 public @interface UseCase {
12  public int id();
13  public String description()default "no description";
14 }
```

spring Data jpa简介_动力节点Java学...
Spring MVC 关于controller的字符编...
java实现菜单滑动效果
初次使用IDEA创建maven项目的教程
利用Socket.io 实现消息实时推送功能
SQL Server 2000 Driver for JDBC Se...
工厂模式_动力节点Java学院整理
Java Thread多线程详解及用法解析
详解JavaEE 使用 Redis 数据库进行内...
利用HttpUrlConnection 上传 接收文...

```
 PasswordUtils .java
package com;

public class PasswordUtils {
 @UseCase(id=47,description="Passwords must contain at least one numeric")
 public boolean validatePassword(){
 return true;
 }

 @UseCase(id=48)
 public String encryptPassword(String password){
 return password;
 }

 @UseCase(id=49,description="Jong_Cai")
 public void showName(){
 System.out.println("Jong_Cai");
 }
}
```

2.编写注解处理器

如果没有用来读取注解的工具,那注解也不会比注释更有用.使用注解的过程中.很重要的一个部分就是创建与使用注解处理器.Java SE5扩展了反射机制的API,以帮助程序员构造这类工具.同时,它还提供了一个外部工具apt帮助程序员解析带有注解的Java源代码.下面是一个非常简单的注解处理器,我们将用它来读取PasswordUtils类,并使用反射机制查找@UseCase标记.我们为其提供了一组id值得,然后它会列出在PasswordUtils中找到的用例,以及缺失的用例.

```
UseCaseTracker.java
package com;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class UseCaseTracker {
 public static void trackUseCases(List<Integer> list, Class<?> cl) {
 for (Method m : cl.getDeclaredMethods()) {
```

```
    UseCase us = m.getAnnotation(UseCase.class);
    if (us != null) {
    System.out.println("Found Use Case:" + us.id() + " "
     + us.description());
    list.remove(new Integer(us.id()));
    }
   }
   for (int i : list) {
    System.out.println("Warning:Missing use case-" + i);
   }
   }

   public static void main(String[] args) {
   List<Integer> list = new ArrayList<Integer>();
   Collections.addAll(list, 47,48,49,50,51);
   trackUseCases(list, PasswordUtils.class);
   }
  }
```

这个程序用到了两个反射的方法:getDeclaredMethods()和getAnnotation(),它们都属于AnnotatedElement接口(class,Method与Field等类都实现了该接口).getAnnotation()方法返回指定类型的注解对象,在这里就是UseCase,如果被注解的方法上没有该类型的注解,则返回null值.然后我们通过调用id()和description()方法从返回的UseCase对象中提取元素的值.其中encryptPassword()方法在注解的时候没有指定description的值,因此处理器在处理它对应的注解时,通过description()方法取得的是默认值no description.

Annotation在java的世界正铺天盖地展开,有空写这一篇简单的annotations的文章,算是关于Annotation入门的文章吧,希望能各位们能抛砖,共同学习……

不讲废话了,实践才是硬道理.

3.实例

下面讲一下annotation的概念先，再来讲一下怎样设计自己的annotation.

首先在jdk自带的java.lang.annotation包里,打开如下几个源文件:

源文件Target.java

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
  ElementType[] value();

@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
  ElementType[] value();
}
```

其中的@interface是一个关键字，在设计annotations的时候必须把一个类型定义为@interface，而不能用class或interface关键字(会不会觉得sun有点吝啬,偏偏搞得与interface这么像).

源文件Retention.java

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Retention {
  RetentionPolicy value();
}

@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Retention {
  RetentionPolicy value();
}
```

看到这里，大家可能都模糊了,都不知道在说什么，别急，往下看一下. 在上面的文件都用到了RetentionPolicy，ElementType这两个字段,你可能就会猜到这是两个java文件.的确，这两个文件的源代码如下：

源文件RetentionPolicy.java

```
public enum RetentionPolicy {
 SOURCE,
 CLASS,
 RUNTIME
}

public enum RetentionPolicy {
 SOURCE,
 CLASS,
 RUNTIME
}
```

这是一个enum类型,共有三个值，分别是SOURCE,CLASS 和 RUNTIME.

SOURCE代表的是这个Annotation类型的信息只会保留在程序源码里，源码如果经过了编译之后，Annotation的数据就会消失,并不会保留在编译好的.class文件里面。

CIASS的意思是这个Annotation类型的信息保留在程序源码里,同时也会保留在编译好的.class文件里面,在执行的时候，并不会把这一些信息加载到虚拟机(JVM)中去.注意一下，当你没有设定一个Annotation类型的Retention值时，系统默认值是CLASS.

第三个,是RUNTIME,表示在源码、编译好的.class文件中保留信息，在执行的时候会把这一些信息加载到JVM中去的.

举一个例子，如@Override里面的Retention设为SOURCE,编译成功了就不要这一些检查的信息;相反,@Deprecated里面的Retention设为RUNTIME,表示除了在编译时会警告我们使用了哪个被Deprecated的方法,在执行的时候也可以查出该方法是否被Deprecated.

源文件ElementType.java

```
public enum ElementType {
 TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR,
 LOCAL_VARIABLE, ANNOTATION_TYPE,PACKAGE
}

public enum ElementType {
 TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR,
 LOCAL_VARIABLE, ANNOTATION_TYPE,PACKAGE
}
```

@Target里面的ElementType是用来指定Annotation类型可以用在哪一些元素上的.说明一下：TYPE(类型), FIELD(属性), METHOD(方法), PARAMETER(参数), CONSTRUCTOR(构造函数),LOCAL_VARIABLE(局部变量), ANNOTATION_TYPE,PACKAGE(包),其中的TYPE(类型)是指可以用在Class,Interface,Enum和Annotation类型上.

另外,从1的源代码可以看出,@Target自己也用了自己来声明自己,只能用在ANNOTATION_TYPE之上.

如果一个Annotation类型没有指明@Target使用在哪些元素上,那么它可以使用在任何元素之上,这里的元素指的是上面的八种类型.

举几个正确的例子:

@Target(ElementType.METHOD)

@Target(value=ElementType.METHOD)

@Target(ElementType.METHOD,ElementType.CONSTRUCTOR)

具体参考一下javadoc文档

源文件它们都使用了@Documented,@Documented的目的就是让这一个Annotation类型的信息能够显示在javaAPI说明文档上;没有添加的话，使用javadoc生成API文档的时候就会找不到这一个类型生成的信息.

另外一点，如果需要把Annotation的数据继承给子类，那么就会用到@Inherited这一个Annotation类型.

下面讲的设计一个最简单的Annotation例子,这一例子共用四个文件;

Description.java

```
1  package lighter.javaeye.com;
2
3  import java.lang.annotation.Documented;
4  import java.lang.annotation.ElementType;
5  import java.lang.annotation.Retention;
6  import java.lang.annotation.RetentionPolicy;
7  import java.lang.annotation.Target;
8
9  @Target(ElementType.TYPE)
10 @Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
public @interface Description {
  String value();
}

package lighter.javaeye.com;

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;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Description {
  String value();
}
```

说明:所有的Annotation会自动继承java.lang.annotation这一个接口,所以不能再去继承别的类或是接口.

最重要的一点,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,Clas

s,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:上面的例子就只有一个参数成员.

Name.java

```
package lighter.javaeye.com;

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;

//注意这里的@Target与@Description里的不同,参数成员也不同
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
```

```
@Documented
public @interface Name {
  String originate();
  String community();
}

package lighter.javaeye.com;

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;

//注意这里的@Target与@Description里的不同,参数成员也不同
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Name {
  String originate();
  String community();
}
```

JavaEyer.java

```
package lighter.javaeye.com;

@Description("javaeye,做最棒的软件开发交流社区")
public class JavaEyer {
  @Name(originate="创始人:robbin",community="javaEye")
  public String getName()
  {
    return null;
  }

  @Name(originate="创始人:江南白衣",community="springside")
  public String getName2()
  {
    return "借用两位的id一用,写这一个例子,请见谅!";
  }
}


package lighter.javaeye.com;
```

```
@Description("javaeye,做最棒的软件开发交流社区")
public class JavaEyer {
  @Name(originate="创始人:robbin",community="javaEye")
  public String getName()
  {
    return null;
  }

  @Name(originate="创始人:江南白衣",community="springside")
  public String getName2()
  {
    return "借用两位的id一用,写这一个例子,请见谅!";
  }
}
```

写一个可以运行提取JavaEyer信息的类TestAnnotation

```
package lighter.javaeye.com;

 import java.lang.reflect.Method;
 import java.util.HashSet;
 import java.util.Set;

 public class TestAnnotation {
  /**
   * author lighter
   * 说明:具体关天Annotation的API的用法请参见javaDoc文档
   */
    public static void main(String[] args) throws Exception {
    String CLASS_NAME = "lighter.javaeye.com.JavaEyer";
    Class test = Class.forName(CLASS_NAME);
    Method[] method = test.getMethods();
    boolean flag = test.isAnnotationPresent(Description.class);
    if(flag)
    {
      Description des = (Description)test.getAnnotation(Description.class);
      System.out.println("描述:"+des.value());
      System.out.println("-----------------");
    }

    //把JavaEyer这一类有利用到@Name的全部方法保存到Set中去
    Set<Method> set = new HashSet<Method>();
    for(int i=0;i<method.length;i++)
    {
      boolean otherFlag = method[i].isAnnotationPresent(Name.class);
```

```
      if(otherFlag) set.add(method[i]);
    }
    for(Method m: set)
    {
      Name name = m.getAnnotation(Name.class);
      System.out.println(name.originate());
      System.out.println("创建的社区:"+name.community());
    }
   }
}

 package lighter.javaeye.com;

 import java.lang.reflect.Method;
 import java.util.HashSet;
 import java.util.Set;

 public class TestAnnotation {
  /**
   * author lighter
   * 说明:具体关天Annotation的API的用法请参见javaDoc文档
   */
    public static void main(String[] args) throws Exception {
    String CLASS_NAME = "lighter.javaeye.com.JavaEyer";
    Class test = Class.forName(CLASS_NAME);
    Method[] method = test.getMethods();
    boolean flag = test.isAnnotationPresent(Description.class);
    if(flag)
    {
      Description des = (Description)test.getAnnotation(Description.class);
      System.out.println("描述:"+des.value());
      System.out.println("-----------------");
    }

    //把JavaEyer这一类有利用到@Name的全部方法保存到Set中去
    Set<Method> set = new HashSet<Method>();
    for(int i=0;i<method.length;i++)
    {
      boolean otherFlag = method[i].isAnnotationPresent(Name.class);
      if(otherFlag) set.add(method[i]);
    }
    for(Method m: set)
    {
      Name name = m.getAnnotation(Name.class);
      System.out.println(name.originate());
      System.out.println("创建的社区:"+name.community());
    }
   }
}
```

运行结果：

```
描述:javaeye,做最棒的软件开发交流社区
创始人:robbin
创建的社区:javaEye
创始人:江南白衣
创建的社区:springside
```

您可能感兴趣的文章：

- java教程之java注解annotation使用方法
- Java Annotation(Java 注解)的实现代码
- 基于Java注解(Annotation)的自定义注解入门介绍
- 基于Java 注解(Annotation)的基本概念详解
- JavaWeb Spring注解Annotation深入学习
- Java注解Annotation解析
- Java Annotation详解及实例代码
- 深入理解Java注解类型(@Annotation)

Java 注解

相关文章

详解 maven的pom.xml用<exclusion>解决版本问题

这篇文章主要介绍了详解 maven的pom.xml用<exclusion>解决版本问题的相关资料,希望通过本文能帮助到大家，需要的朋友可以参考下

2017-09-09

Mybatis分页插件PageHelper的配置和简单使用方法(推荐)

在使用Java Spring开发的时候，Mybatis算是对数据库操作的利器了。这篇文章主要介绍了Mybatis分页插件PageHelper的配置和使用方法,需要的朋友可以参考下

2017-12-12

举例讲解Java的Spring框架中AOP程序设计方式的使用

这篇文章主要介绍了Java的Spring框架中AOP程序设计方式的使用讲解,文中举的AOP下抛出异常的例子非常实用,需要的朋友可以参考下

2016-04-04

简单讲解奇偶排序算法及在Java数组中的实现

这篇文章主要介绍了奇偶排序算法及Java数组的实现,奇偶排序的时间复杂度为O(N^2),需要的朋友可以参考下

2016-04-04

接口隔离原则_动力节点Java学院整理

这篇文章主要介绍了接口隔离原则，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧

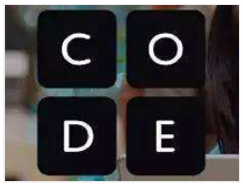
2017-08-08

springboot + devtools（热部署）实例教程

devtools是boot的一个热部署工具,当我们修改了classpath下的文件（包括类文件、属性文件、页面等）时，会重新启动应用。本文通过实例给大家介绍springboot+devtools热部... 2017-04-04

java集合map取key使用示例 java遍历map

这篇文章主要介绍了java集合map取key使用示例,需要的朋友可以参考下 2014-02-02



详解Spring框架之基于Restful风格实现的SpringMVC

这篇文章主要介绍了详解Spring框架之基于Restful风格实现的SpringMVC，具有一定的参考价值，感兴趣的小伙伴们可以参考一下 2017-05-05

Java 中Object的wait() notify() notifyAll()方法使用

这篇文章主要介绍了Java 中Object的wait() notify() notifyAll()方法使用的相关资料,需要的朋友可以参考下 2017-05-05

设计模式之模版方法模式_动力节点Java学院整理

这篇文章主要介绍了设计模式之模版方法模式，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧 2017-08-08

最新评论