1、课程名称：Annotation


魔乐科技JAVA课堂
www.mldnjava.cn
我们的课程 · 一切为了就业
Java SE基础课程
Annotation
北京MLDN软件教学研发中心
李兴华
培训咨询热线：010-51283346　院校合作：010-62350411
官方JAVA学习社区：bbs.mldn.cn

2、知识点

2.1、上次课程的主要知识点

1、 枚举的作用
2、 enum 关键字和 Enum 类的关系
3、 EnumSet 和 EnumMap 的操作

2.2、本次预计讲解的知识点

1、 理解三个系统的 Annotation 的作用
2、 了解自定义 Annotation 的语法
3、 了解反射机制与 Annotation 的使用

4、 了解@Documented、@Retention、@RetentionPolicy、@Target、@Inherited 这些注解的作用

3、具体内容

3.1、认识 Annotation

在之前曾经讲解过，对于程序来讲，最好的做法是做到程序与配置文件相分离。但是随着新的设计理论的出现，现在的最好做法是将程序中所需要的配置直接写到程序之中，这样看起来会更加的直观一些，那么此时就可以使用 Annotation 完成。

3.2、系统内建的三个 Annotation（理解）

在 JDK 1.5 之后，在系统中提供了三个 Annotation，分别是：@Override、@Deprecated、@SupressWarnings。

3.2.1、@Override

此 Annotation 在方法覆写的时候使用，此 Annotation 定义的格式如下：

```
@Target(value=METHOD)
@Retention(value=SOURCE)
public @interface Override
```

此方法是在进行正确的覆写时使用的。

```java
package org.lxh.demo01;
public class Person {
    public String toString() {// 覆写toString()方法
        return "Hello ";
    }
}
```

从之前的操作来将，以上的代码没有任何问题，但是如果现在由于不小心，单词写错了：

```java
package org.lxh.demo01;
public class Person {
    @Override
    public String toString() {// 覆写toString()方法
        return "Hello ";
    }
}
```

使用@Override 可以进行正确的方法覆写操作。

3.2.2、@Deprecated

此 Annotation 表示的是一个类或方法已经不再建议继续使用了。

此 Annotation 的定义语法如下：

```
@Documented
@Retention(value=RUNTIME)
public @interface Deprecated
```

使用此 Annotation 定义的方法或类在使用时会出现警告信息，表示此操作已经不建议继续使用了。

```java
package org.lxh.demo02;
public class Person {
    @Deprecated
    public String say() {// 此方法不建议使用了
        return "Hello ";
    }
}
```

以上表示 say()方法已经不建议使用了，在 Eclipse 中通过删除线进行表示，而在调用此方法处将出现警告信息。

```java
package org.lxh.demo02;
public class Test {
    public static void main(String[] args) {
        new Person().say() ;
    }
}
```

3.2.3、@SuppressWarnings

表示压制警告信息。

```java
package org.lxh.demo03;
import java.io.Serializable;
@SuppressWarnings("serial")
public class Person implements Serializable {
    @Deprecated
    public String say() {// 此方法不建议使用了
        return "Hello ";
    }
}
```

3.3、自定义 Annotation

在 Annotation 的使用中，除了可以使用系统中提供的 Annotation 之外，也可以进行自定义的 Annotation，自定义 Annotation 的语法如下：

```
访问控制权限 @interface Annotation 名称{}
```

而且在 Annotation 中也可以定义若干个变量，接收设置。

范例： 定义一个简单的 Annotation

```java
package org.lxh.demo04;
public @interface MyAnnotation {}
```

在此 Annotation 中并没有定义任何的其他操作，那么此时，就可以通过如下的方式访问 Annotation。

```java
package org.lxh.demo04;
@MyAnnotation
public class Demo {
}
```

范例： 在 Annotation 中定义变量

```java
package org.lxh.demo05;
public @interface MyAnnotation {
    public String name();
    public String info();
}
```

一旦变量定义之后，则在调用此 Annotation 的时候就必须设置变量的内容。

```java
package org.lxh.demo05;
@MyAnnotation(name = "HELLO", info = "MLDNJAVA")
public class Demo {
}
```

此时，在使用此 Annotation 的时候就必须设置两个变量的内容。

如果有时候一些变量希望可以自由设置，不设置的话，可以给出默认值，则此时可以通过 default 指定默认值。

```java
package org.lxh.demo06;
public @interface MyAnnotation {
    public String name() default "hello";
    public String info() default "world";
}
```

则此时在使用 MyAnnotation 的时候即使不指定变量的内容，程序也不会出现任何的错误。

```java
package org.lxh.demo06;
@MyAnnotation
public class Demo {
}
```

也可以定义一个变量的数组，接收一组参数。

```java
package org.lxh.demo07;
public @interface MyAnnotation {
    public String[] name();
}
```

以上的 name 变量将以一个数组的形式出现，所以在设置的时候必须设置上一个数组。

```java
package org.lxh.demo07;
@MyAnnotation(name = { "HELLO", "World" })
public class Demo {
}
```

当然，如果要想指定一个 Annotation 的变量的取值范围，也可以通过枚举操作完成。

```java
package org.lxh.demo08;
public enum Color {
    RED, GREEN, BLUE
}
```

可以将以上的类型定义在 Annotation 之中，但是此时，其取值范围只能有三个。

```java
package org.lxh.demo08;
public @interface MyAnnotation {
    public Color color();
}
```

在使用的时候必须从 Color 中指定具体的类型。

```java
package org.lxh.demo08;
@MyAnnotation(color = Color.RED)
public class Demo {
}
```

3.4、Retention 和 RetentionPolicy

一个 Annotation 如果要想决定其作用的范围，则必须通过 Retention 指定，而 Retention 指定的范围由 ReteiontPolicy 决定。

从 Retention 中可以发现，里面的数据都是 RetentionPolicy 指定的，那么 RetentionPolicy 指定了三种范围：

No.	范围	描述
1	public static final RetentionPolicy SOURCE	在 java 源程序中存在
2	public static final RetentionPolicy CLASS	在 java 生成的 class 中存在
3	public static final RetentionPolicy RUNTIME	在 java 运行的时候存在

所以，如果要想让一个 Annotation 在程序中起作用，则必须指定 RUNTIME 类型

```java
package org.lxh.demo08;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
}
```

3.5、反射与 Annotation

之前学习过了很多的语法，而如果要想一个 Annotation 真正起作用的话，则必须结合反射机制，在反射中提供了以下的操作方法：java.lang.reflect.AccessibleObject

No.	方法名称	类型	描述
1	public boolean isAnnotationPresent(Class<? extends Annotation> annotationClass)	普通	判断是否是指定的 Annotation
2	public Annotation[] getAnnotations()	普通	得到全部的 Annotation

Constructor、Field、Method 都是 AccessibleObject 的子类，所以在构造方法中，成员变量中，方法中都可以使用 Annotation。

范例： 定义一个 Annotation，里面存在两个变量

```java
package org.lxh.demo10;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
```

```java
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
    public String info();
}
```

之后定义一个类，此类使用三种 Annotation 声明。

```java
package org.lxh.demo10;
public class Demo {
    @SuppressWarnings("unchecked")
    @Override
    @Deprecated
    @MyAnnotation(name = "hello", info = "world")
    public String toString() {
        return "hello";
    }
}
```

以上的方法中有四个 Annotation。但是只有 MyAnnotation、Deprecated 两个属于 RUNTIME 范围的 Annotation。所以程序最终只能够取得两个 Annotation。

```java
package org.lxh.demo10;
import java.lang.annotation.Annotation;
import java.lang.reflect.Method;
public class ClassDemo {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Class.forName("org.lxh.demo10.Demo");
        Method met = cls.getMethod("toString"); // 找到toString()方法
        Annotation ann[] = met.getAnnotations(); // 得到全部的Annotation
        for (int x = 0; x < ann.length; x++) {
            System.out.println(ann[x]);
        }
    }
}
```

既然已经得到了全部的 Annotation，那么下面呢，既然在使用 MyAnnotation 的时候为里面设置了内容，就应该可以取得这样的功能。

```java
package org.lxh.demo11;
import java.lang.reflect.Method;
public class ClassDemo {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Class.forName("org.lxh.demo11.Demo");
        Method met = cls.getMethod("toString"); // 找到toString()方法
        if (met.isAnnotationPresent(MyAnnotation.class)) {
            MyAnnotation my = (MyAnnotation) met
                    .getAnnotation(MyAnnotation.class);
            String name = my.name();
            String info = my.info();
```

```java
            System.out.println("name = " + name);
            System.out.println("info = " + info);
        }
    }
}
```

如果一个 Annotation 要想起作用，则必须结合反射机制完成。

3.6、@Documented 注释

此注释表示的是文档化，可以在生成 doc 文档的时候添加注释。

```java
package org.lxh.demo12;
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
@Documented
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
    public String info();
}
```

则在使用 Demo 类的时候，可以增加一些 DOC 注释。

```java
package org.lxh.demo12;
public class Demo {
    /**
     * 这个方法是从Object类中覆写而来的
     */
    @MyAnnotation(name = "hello", info = "world")
    public String toString() {
        return "hello";
    }
}
```

```
javadoc -d . Demo.java
```

3.7、@Target 注释

@Target 注释表示的是一个 Annotation 的使用范围，例如：之前定义的 MyAnnotation 可以在任意的位置上使用。

如果这个时候需要指定其使用范围的话，则就必须通过@Target 进行指定。

No.	范围	描述
1	public static final ElementType TYPE	只能在类或接口上使用
2	public static final ElementType FIELD	在成员变量使用
3	public static final ElementType METHOD	在方法中使用
4	public static final ElementType PARAMETER	在参数上使用
5	public static final ElementType CONSTRUCTOR	在构造中使用
6	public static final ElementType LOCAL_VARIABLE	局部变量上使用
7	public static final ElementType ANNOTATION_TYPE	只能在 Annotation 中使用
8	public static final ElementType PACKAGE	只能在包中使用

下面定义一个 Annotation 只能在类上使用。

```java
package org.lxh.demo13;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Target(value=ElementType.TYPE)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
    public String info();
}
```

3.8、@Inherited

@Inherited 表示一个 Annotation 是否允许被其子类继承下来。

```java
package org.lxh.demo14;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Inherited
@Target(value = ElementType.TYPE)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    public String name();
    public String info();
}
```

此 Annotation 在使用的时候允许被其子类所继承。

4、总结

1、 只需要理解三种系统内建的 Annotation 即可
2、 了解 Annotation 的基本语法即可
3、 如果一个 Annotation 要想起作用，则必须依靠反射完成