注解与注释

注释

单行注释://

多行注释:/* */

文档注释:/** */

注释的作用

传递额外信息, 给程序员看的, 不参与编译

只是规定语法 内容随便写 只要能看懂

```
// 年龄在18-25
// [18,25]   18<=age<=25
public boolean judgeAge(){

}
```

什么是注解

Annotation其实是代码里的特殊标记，这些标记可以在编译、类加载、运行时被读取，并执行相应的处理

注解的作用

通过使用Annotation，程序开发人员可以在不改变原有逻辑的情况下，在源文件嵌入一些补充信息

Annotation就像修饰符一样被使用，可用于修类、构造器、方法、成员变量、参数...，这些信息被存储在Annotation的“属性名=属性值”对中

理解为一个标签

注解 VS 注释

- 相同点
  - 都是用来传递额外信息的
- 不同点
  - 注解可以参与编译,注释不行
  - 注解有使用范围,注释没有(想咋写咋写)
  - 注解作为一种数据类型,跟class interface具有同等地位

注解定义

```
权限修饰符 @interface 注解名{
    // 注解体
```

```
    属性类型 属性名();
    属性类型 属性名();
    属性类型 属性名();
    .....
}

属性类型的范围
java基本数据类型
String类型
Class类型
枚举类型
注解类型
以及以上类型的数组
```

注意:

不允许继承

元注解

元注解：描述注解的注解（注解的注解） 元数据 meta data

常用元注解:

@Retention元注解，来定义我们自己定义的注解的保留级别.

- RetentionPolicy.RUNTIME
- RetentionPolicy.CLASS 默认
- RetentionPolicy.SOURCE

@Target元注解，注解可以作用的目标

对于注解而言，可以作用的目标:

1. 整个类 ElementType.TYPE
2. 成员变量 ElementType.FIELD
3. 构造方法 ElementType.CONSTRUCTOR
4. 成员方法 ElementType.METHOD

注解的使用(重点)

语法:

类对象使用

new创建对象的时候 , 通过构造方法完成赋值. 也可以不赋值, 但是有默认值

注解的使用

创建对象,完成赋值

```
@注解名(属性1=value,属性2=value...)

举例:
@Override
```

```
public class Dmeo {
    public static void main(String[] args) {

    }

    @MyAnno4("admin")
    @MyAnno3(names = {"zs","ls","ww"})
    @MyAnno2()
    @MyAnno(name = "zs",age = 20)
    public static void func() {

    }
}

@interface MyAnno{
    // 注解体
    String name();
    int age();
}

@interface MyAnno2{
    // 设置默认值
    String role() default "teacher";
}

@interface MyAnno3{
    String[] names();
}

@interface MyAnno4{
    String value();
}
```

注意事项:

- 每个属性都要赋值
- 也可以不赋值, 但是有默认值, 通过default去设置
- 数组形式的赋值, 使用{ }
- 如果属性只有1个 并且叫做value 可以简化赋值
- 引用类型不能是null

注解处理器

获取注解信息,根据信息进行处理

使用反射获取注解信息

```
package _25annotation.com.cskaoyan._04handle;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.reflect.Method;
```

```java
/**
 * @description:
 * @author: 景天
 * @date: 2022/8/2 9:55
 **/

public class Demo {
    public static void main(String[] args) throws Exception{
        // 获取字节码文件对象
        Class<?> c = Class.forName("_25annotation.com.cskaoyan._04handle.Demo");
        // 获取方法对象
        Method loginMethod = c.getDeclaredMethod("login");
        // 判读是否使用了注解
        boolean annotationPresent =
loginMethod.isAnnotationPresent(Login.class);
        if (annotationPresent) {
            // 获取注解实例
            Login annotation = loginMethod.getAnnotation(Login.class);
            // 获取属性值
            String name = annotation.name();
            String password = annotation.password();
            System.out.println(name);
            System.out.println(password);
        }else {
            System.out.println("没有使用注解");
        }
    }

    @Login()
    public static void login() {

    }
}
//使用元注解修饰
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface Login{
    // 注解体
    String name() default "root";

    String password() default "123456";
}
```

练习:

定义2个注解 , 修饰成员变量

@NameLimit　length 名字长度不能超过5

@AgeLimit　　maxAge　minAge　18-25

创建学生对象, 用2个注解修饰成员变量的值. 要求满足18-25, 并且名字长度不超过5 允许创建学生对象

```java
package _25annotation.com.cskaoyan._04handle;
```

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/2 10:16
 **/

public class StudentFactory {
    static Class stuCls;
    static {
        try {
            stuCls =
Class.forName("_25annotation.com.cskaoyan._04handle.Student");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    // 产生学生对象的方法
    public static Student getInstance(String name, int age) throws
NoSuchMethodException, IllegalAccessException, InvocationTargetException,
InstantiationException, NoSuchFieldException {
        // 判断名字
        judgeName(name);


        // 判断年龄
        judgeAge(age);


        // 获取构造方法
        Constructor declaredConstructor =
stuCls.getDeclaredConstructor(String.class, int.class);

        // 实例化
        declaredConstructor.setAccessible(true);
        Student student = (Student) declaredConstructor.newInstance(name, age);

        // 最终要返回学生对象
        return student;
    }

    private static void judgeAge(int age) throws NoSuchFieldException {
        // 获取年龄成员变量
        Field ageField = stuCls.getDeclaredField("age");
        // 判断是否使用了注解
        boolean annotationPresent =
ageField.isAnnotationPresent(AgeLimit.class);
        // 获取注解实例
        if (annotationPresent) {
            // 获取属性值
```

```
            AgeLimit ageLimit = ageField.getAnnotation(AgeLimit.class);
            int maxAge = ageLimit.maxAge();
            int minAge = ageLimit.minAge();
            // 判断是否合法
            if (age < minAge || age > maxAge) {
                throw new IllegalArgumentException("年龄不合法!");
            }

        }
    }

    private static void judgeName(String name) throws NoSuchFieldException {
        // 获取name成员变量
        Field nameField = stuCls.getDeclaredField("name");
        // 判断是否使用了注解
        boolean annotationPresent =
nameField.isAnnotationPresent(NameLimit.class);
        if (annotationPresent) {
            // 获取注解实例
            NameLimit nameLimit = nameField.getAnnotation(NameLimit.class);
            // 获取属性值
            int length = nameLimit.length();
            // 做判断
            if (name.length() > length) {
                // 抛出异常
                throw new IllegalArgumentException("名字不合法!");
            }

        }
    }
}
```

注解VS配置文件

注解 VS 配置文件
配置文件
优点：可配置，不用改源码。管理方便
缺点：不直观，开发效率低

注解
优点：直观开发效率高
缺点：硬编码，修改之后需要重新编译运行
　难以和代码分开独立管理

注解的使用场景

SE: @Test @Override @Deperate

EE : @WebServlet

框架: @AutoWired @Service @RequestMapping @Data