注解

概念

注释: 给人看的提示信息

注解: 给程序看的提示信息

可以根据 有没有注解 或 注解上的属性值的不同 决定程序按照不同的方式去运行
本质上是对程序进行的配置,是对配置文件的一种替代方案
相对于配置文件,它更加轻量级,简单 方便 易用,便于管理
现代开发中,很多时候,配置文件用于配置核心的选项,而注解用于配置其它轻量级配置.

JDK原生注解

> @Override - 声明重写
> @Deprecated - 声明过时
> @SupressWarnings - 压制警告

自定义注解

定义注解类

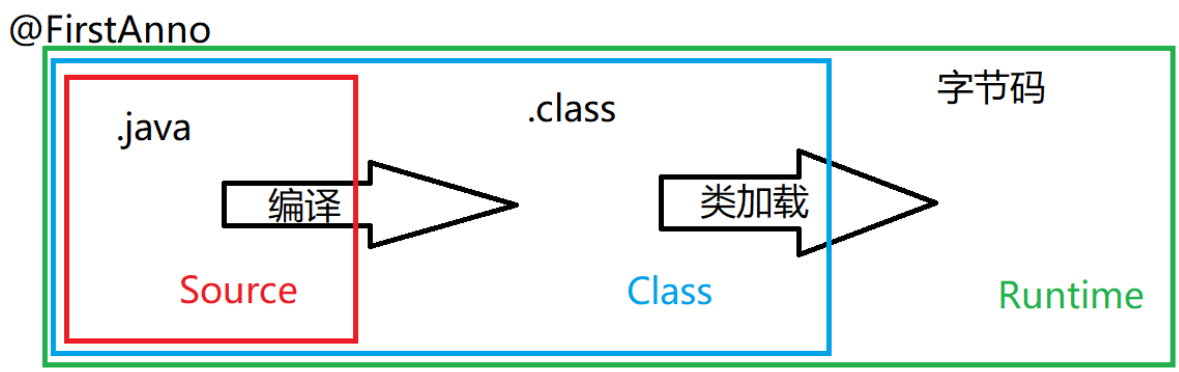
定义一个注解非常类似于定义一个接口,不同的是,需要通过 @interface 来声明

```
public @interface MyAnno{}
```

元注解标注

所谓元注解,是java官方提供的,用于定义注解特性的注解,所以称为"元"注解.
@Target - 限定当前注解可以用在哪些地方
@Retention - 限定当前注解能够被保留到什么阶段
RetentionPolicy.SOURCE
RetentionPolicy.CLASS
RetentionPolicy.RUNTIME

只有配置为RUNTIME,注解信息才会被保留到运行阶段,注解才能在运行时控制程序运行



声明属性

声明注解属性的过程非常类似于为接口定义方法

访问权限只能是public的,不写默认就是public

类型只能是八种基本数据类型 String类型 class类型 枚举类型 其它注解类型 及 以上类型的一维数组

可以在属性的声明之后,通过default关键字为属性声明默认值.

有默认值的属性可以不赋值,不赋值则取默认值

为注解声明的属性,如无默认值,必须在使用注解时,明确指定属性的值

在注解使用时,后跟小括号,在小括号内通过属性名=属性值的方式为注解属性赋值

如果属性是数组类型,则需要通过{}包裹所有值

如果数组中值只有一个,则包裹数组的大括号可以省略

如果需要赋值的值只有一个,且名字叫value,则赋值时,value=可以省略

反射注解

通过反射注解可以获取组件上注解的信息，从而控制程序按照不同方式运行

```
boolean  isAnnotationPresent(Class anno) -- 判断当前类上是否有指定类型的注解
Annotation [] getAnnotations(); -- 获取当前类上声明的所有注解
Annotation getAnnotation(Class annoClz); -- 获取当前类上的指定类型的注解
```

案例1

```
package anno;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
enum PL{
    协警,交警,刑警,警督
}

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@interface Level{
    PL value() default PL.协警;
}

@Level(PL.警督)
class Police{
    public static void faKuan(){
        System.out.println("罚款200元..");
    }
}

public class Demo02 {
    public static void main(String[] args) {
```

```
        Police.faKuan();
        if(Police.class.isAnnotationPresent(Level.class)){
            Level anno = Police.class.getAnnotation(Level.class);
            PL pl = anno.value();
            if(pl == PL.协警){
                System.out.println("大哥,抽根烟,少罚点行不行,给你50得了~~~");
            }else if(pl == PL.交警){
                System.out.println("交200走人..");
            }else if(pl == PL.刑警){
                System.out.println("给了钱赶紧跑,别查出别的事~~~");
            }else{
                System.out.println("首长好,200够不够,不够还有~~~");
            }

        }else{
            System.out.println("你个假警察!打一顿,扭送派出所~~~");
        }
    }
}
```

案例2

```
public class Demo03 {
    public static void main(String[] args) {
        MyService service = new MyService();
        service.test();
    }
}
```

```
package anno.service;

import anno.dao.MyDao;

public class MyService {
    private MyDao dao = MyDaoFactory.getDao();

    public void test(){
        dao.add();
        dao.get();
    }
}
```

```
package anno.service;

import anno.UseThis;
import anno.dao.MyDao;

import java.io.File;
import java.io.FileInputStream;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
```

```
public class MyDaoFactory {
    private static List<Class> list = new ArrayList<>();
    static{
        try {
            //获取anno/dao目录下所有的.java文件
            URL url =
MyDaoFactory.class.getClassLoader().getResource("./anno/dao");
            File dir = new File(url.toURI());
            File[] files = dir.listFiles();
            //遍历这些java文件
            for(File file : files){
                if(file.isFile() && file.getName().endsWith(".class")){
                    //截取它们文件名,拼上报名,得到类的全路径名
                    String shortName = file.getName().split("\\.")[0];
                    String fullName = "anno.dao."+shortName;
                    //反射加载这些类
                    Class<?> clz = Class.forName(fullName);
                    //存入集合
                    list.add(clz);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
    public static MyDao getDao(){
        try {
            //遍历dao包下所有类
            for(Class clz : list){
                //如果发现某个类有@UseThis,返回这个类的对象
                if(clz.isAnnotationPresent(UseThis.class)){
                    return (MyDao) clz.newInstance();
                }
            }
            return null;
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
}
```

```
package anno.dao;

public interface MyDao {
    public void add();
    public void get();
}
```

```
package anno.dao;
```

```java
import anno.UseThis;


public class MySqlDao implements MyDao{
    @Override
    public void add() {
        System.out.println("MySql..add..");
    }

    @Override
    public void get() {
        System.out.println("MySql..get..");
    }
}
```

```java
package anno.dao;

import anno.UseThis;

@UseThis
public class OracleDao implements MyDao{
    @Override
    public void add() {
        System.out.println("Oracle..add..");
    }

    @Override
    public void get() {
        System.out.println("Oracle..get..");
    }
}
```