链滴

注解和反射

作者：wenyl
原文链接：https://ld246.com/article/1640077404103
来源网站：链滴

1、注解

1.1、内置注解

- @Override

子类重写方法

- @Deprecated

标识这个方法或者类已经被废弃或有更好的选择

- @SuppressWarnings

抑制编译器告警

1.2、元注解

元注解是用来修饰注解的注解

- @Target

指定注释的使用范围

- @Retention

注释的生命周期，有SOURCE（编译时被抛弃），CLASS（class文件中存在，但是运行时被抛弃），UNTIME（运行时依然存在）

- @Documented

标识生成javadoc时，该注解修饰的注解也会在javadock中显示

- @Inherited

某个类使用了用@Inherited注解标识的注解，则他的子类也会继承这个注解

1.3、自定义注解

使用@interface定义注解

```
import java.lang.annotation.*;

@Target(value = {ElementType.METHOD,ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
public @interface MyAnnotation {
    String[] names() default {};
}

@MyAnnotation(names = {"wenyl","zhaomt"})
public class BasicAnnotation {
}
```

2、反射

对于运行时的类，动态获取类信息，动态调用类的方法及属性的功能成为反射

2.1、获取Class对象

```
public class Person {

    public static void main(String[] args) throws ClassNotFoundException {
        // 直接通过类.class获取
        Class<Person> personClass = Person.class;

        // 通过类名获取
        Class<?> aClass1 = Class.forName("cn.com.wyl.study.reflection.Person");

        // 通过对象实例获取
        Person person = new Person();
        Class<? extends Person> aClass = person.getClass();
    }

}
```

3、反射获取泛型参数

通过反射获取传入的类的泛型实现如下，先定义一个泛型类

这里将获取传入的泛型的实际类型的方法写在构造函数里

```
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.util.Calendar;

public abstract class BaseObject<T> {
    public BaseObject(){
        Type type = getClass().getGenericSuperclass();
        if( type instanceof ParameterizedType) {
            ParameterizedType parameterizedType = (ParameterizedType) type;
            type = parameterizedType.getActualTypeArguments()[0];
            if (type instanceof Class) {
                Class<T> type1 = (Class<T>) type;
                System.out.println(type1);
            }
        }


    }
    public abstract void insert(T t);
}
```

然后定义一个类来继承这个抽象类

```
public class TargetObject1 extends BaseObject<Integer>{

    public void insert(Integer integer) {

    }
}
```

最后我们创建对象

```
    public static void main(String[] args) throws ClassNotFoundException {
        BaseObject<Integer> baseObject = new TargetObject1();
    }
```

在初始化的时候会调用抽象类的构造函数获取传入的泛型的实际类型

4、反射获取注解

参照hibernate自定义两个注解

```
@Table(tableName = "person")
public class Person {
    @Column(columnName = "person_name",columnType = "varchar",length = 255)
    private String personName;

    public Person(){}
    public Person(String personName) {
        this.personName = personName;
    }

    public String getPersonName() {
        return personName;
```

```
    }

    public void setPersonName(String personName) {
        this.personName = personName;
    }

}

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@interface Table{
    String tableName();
}

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@interface Column{
    String columnName();
    int length() default 32;
    String columnType() default "varchar";
}
```

获取注解如下

```
    public static void main(String[] args) throws Exception{
        // 获取类注解
        Person person = new Person();
        Table annotation = person.getClass().getAnnotation(Table.class);
        System.out.println(annotation.tableName());

        // 获取属性注解
        Field field = person.getClass().getDeclaredField("personName");
        Column column = field.getAnnotation(Column.class);
        System.out.println(column.columnName());
        System.out.println(column.columnType());
        System.out.println(column.length());
    }
```

这里都是获取指定注解，需要获取所有注解使用getAnnotations()方法，获取方法的注解可以先获取法在获取方法上的注解（参照获取属性）