目标

- [ ] 理解掌握 JavaBean 规范
- [ ] 了解 Lombok 插件的安装
- [ ] 掌握 Lombok 工具的使用
- [ ] 掌握内省的基本操作(属性的操作)
- [ ] 理解注解的使用
- [ ] 掌握注解的基本定义和使用

1_JavaBean

Javabean 是 Java 中最重要的一个可重用的**组件**(减少代码重复,可重用,封装业务逻辑,封装数据).

组件:一些符合某种规范的类,可以完成特定的功能.

JavaBean 的规范要求:

1. 使用 public 修饰.
2. 字段私有化.
3. 提供 get/set 方法.
4. 公共的无参数的构造器.(使用反射,使用字节码对象.newInstance去创建对象.)

三大成员:

1. 事件.
2. 方法.
3. **属性**(最重要的一个概念).

叮丁狼教育

什么是属性(property)呢?:

JavaBean 可以封装数据,就是将数据保存到一个 bean 对象的属性中的.**属性不是字段,属性是通过get/set方法推导出来的.**

规范的get方法/获取方法/读方法: public修饰,无参数,有返回,get开头. **规范的set方法/设置方法/写方法**: public修饰,有参数,无返回,set开头.

```
// 1.public 修饰. 2.无参数. 3. 有返回. 4.get开头.---> 标准的get方法/读方法.
// 属性的概念就出现了: 把get去掉,首字母小写.
// 属性:name
public String getName() {
    return name;
}
// 1.public 修饰. 2.有一个参数. 3.无返回. 4. set开头.--->标准的set方法/写方法.
// 属性的概念也出现了: 把set去掉,首字母小写.
// 属性:name
public void setName(String name) {
    this.name = name;
}
```

注意:

1. 只要是标准的get/set方法,就存在属性.不一定非得是通过工具自动生成的规范的写法.

```
// getStartIndex 方法也是一个标准的get方法,属性为:startIndex
public int getStartIndex(){
    return (currentPage - 1) * pageSize;
}
```

以上代码,并不存在字段 startIndex, 但是因为 getStartIndex 方法是规范的 get 方法,所以就存在属性 startIndex.

2. 字段是 boolean 的,读方法不是 get 开头,是 is 开头.

```
private boolean man;
public boolean isMan() {
    return man;
}
public void setMan(boolean man) {
    this.man = man;
}
```

2_LomBok 使用

lombok是开源的代码生成库，是一款非常实用的小工具，我们在编辑实体类时可以通过 lombok 注解减少getter、setter等方法的编写，在更改实体类时只需要修改属性即可，减少了很多重复代码的编写工作。

2.1_Lombok 插件安装

1. 点击菜单栏中的 File-->Settings，或者使用快捷键 Ctrl+Alt+S进入到设置页面。
2. Plugins -> 选择Browse repositories -> 搜索页面输入lombok->点击 Install 按钮
3. Settings设置页面->Build，Execution，Deployment-->选择Compiler-->选中Annotation Processors，然后在右侧勾选 Enable annotation processing即可。
4. 重启工具

2.2_Lombok 工具包导入

1. 下载拷贝 lombok 包到项目中
2. 引用 lombok 包

2.3_Lombok 的使用(掌握)

1. @Getter : 生成 getter 方法
2. @Setter : 生成 setter 方法
3. @Data : 生成类中所有支持的方法(不推荐)
4. @ToString: 生成 toString 方法

叮丁狼教育

5. @NoArgsConstructor : 无参构造器
6. @AllArgsConstructor : 全参构造器

3_内省

3.1_内省介绍

Javabean是一个非常常用的组件,无外乎就是操作里面的属性.而之前咱们要获取JavaBean中的方法,如果使用反射非常麻烦.SUN公司专门提供了一套操作 Javabean 属性的API: **内省**.

目标:记住内省的核心类 **Introspector**,熟练操作 JavaBean 的属性.

内省的入口: Introspector

3.2_内省的作用:

1. 获取到属性名和属性类型等相关状态信息.
2. 获取属性对应的读写方法操作属性的值等操作方式.

3.3_内省常用的API

1. 通过字节码对象,获取到 Javabean 的描述对象. Introspector类: public static BeanInfo getBeanInfo(Class beanClass, Class stopClass):返回 Javabean的描述对象
2. 通过 Javabean 描述对象获取属性描述器. BeanInfo类: PropertyDescriptor[] getPropertyDescriptors(): 获取属性描述器.
3. 通过属性描述器,获取到属性名,属性类型,读写方法. PropertyDescriptor类: public String getName(): 获取属性名 public Class<?> getPropertyType(): 获取属性类型 public Method getReadMethod():获取读方法 public Method getWriteMethod():获取写方法

3.4_实战

```
@Test
public void testIntrospector() throws Exception {
    // 创建对象来调用方法
    Person obj = Person.class.newInstance();
    // 把 JavaBean 转成 beanInfo
    BeanInfo beanInfo = Introspector.getBeanInfo(Person.class);
    // 2 通过beanInfo 获取所有的属性
    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
    // 3 遍历属性描述器数组,获取到每个属性描述器
    for (PropertyDescriptor pd: pds) {
        // 获取属性名
        System.out.println("属性名:" + pd.getName());
        // 获取属性类型
        System.out.println("属性类型:" + pd.getPropertyType());
```

中软狼教育

```
        // 获取属性的getter 和setter 方法
        Method getMethod = pd.getReadMethod();
        Method setMethod = pd.getWriteMethod();
        System.out.println("属性getter:" + getMethod);
        System.out.println("属性setter:" + setMethod);
        // 调用属性的 getter 和 setter方法
        // 调用name属性的setter 方法
        if("username".equals(pd.getName())) {
            setMethod.invoke(obj, "小狼");
        }
        // 调用所有属性的getter 方法
        System.out.println(getMethod.invoke(obj));
    }
}
```

通过字节码对象来获取BeanInfo对象的时候,默认会内省当前字节码对象以及其所有的父类的信息.比如,getBeanInfo(A.class),其实它也会内省A的父类,如Object的信息.一般来说,我们不关心父类的属性相关信息,此时可以调用getBeanInfo的重载方法.getBeanInfo(beanClass,stopClass)

实战:

```
BeanInfo beanInfo = Introspector.getBeanInfo(Person.class,Object.class);
```

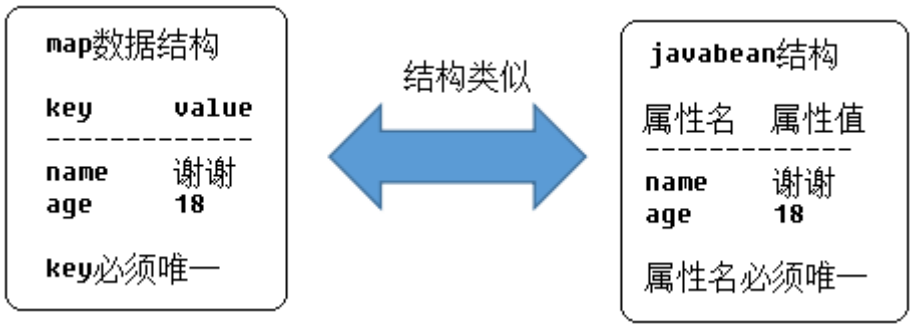
3.5_JavaBean 和 Map 之间的转化

目标: 掌握 JavaBean 和 map 的结构,通过分析结构的相似,锻炼需求分解的能力.

1. 为什么需要将 JavaBean 和 Map 进行转换? 在很多应用场景中,需要将 key=value 形式的数据(ResultSet)与 JavaBean 对象相互转换.

2. 为什么具备可以转换的条件?

并不是说,任何两种结构的数据都是可以相互转换的.之所以map和JavaBean可以转换,是因为它们在数据结构上就极其相似.



所以,我们可以将 map 和 JavaBean 相互转换.将key和属性名一一对应起来.

实战:

Javabean 转 map

```
// Javabean 转 map
public static Map<String,Object> bean2map(Object obj) throws Exception {
    // 创建要给Map 对象
    Map<String,Object> map = new HashMap<>();
    // 1 把 obj 通过内省去得到所有的属性
    BeanInfo beanInfo = Introspector.getBeanInfo(obj.getClass(), Object.class);
    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
    for (PropertyDescriptor pd : pds) {
        // 2 获取属性名作为map 的key
        String key = pd.getName();
        // 3 获取属性的getter方法并调用,得到属性值作为map 的value
        Object value= pd.getReadMethod().invoke(obj);
        map.put(key,value);
    }
    return  map;
}
```

测试方法:

```
@Test
public  void testBeanToMap() throws Exception {
    Person p = new Person(1L,"小码",true);
    Map<String, Object> map = BeanToMapUtil.bean2map(p);
    System.out.println(map);
}
```

map转javabean

```
// map 转为 JavaBean
public static Object map2bean(Map<String, Object> map, Class clz) throws
Exception {
    // 创建JavaBean对象
    Object obj = clz.newInstance();
    // 遍历属性,获取属性名作为map 的key 去获取value值,再设置给setter 方法
    // 1 获取所有的属性
```

叮丁猿教育

```
    BeanInfo beanInfo = Introspector.getBeanInfo(clz, Object.class);
    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
    for (PropertyDescriptor pd : pds) {
        // 2 获取属性名作为map 的key 去获取属性值
        String key = pd.getName();
        // 3 获取到属性值,调用属性的setter方法去设置属性值
        Object value = map.get(key);
        pd.getWriteMethod().invoke(obj,value);
    }
    return obj;
}
```

测试代码:

```
    @Test
    public  void testBeanToMap() throws Exception {
        Person p = new Person(1L,"小码",true);
        Map<String, Object> map = BeanToMapUtil.bean2map(p);
        System.out.println(map);

        // 测试 map2bean
        Person p2 = (Person)BeanToMapUtil.map2bean(map, Person.class);
        System.out.println(p2);
    }
```

问题: 调用者已经告诉工具方法要把map转为 Person,而拿到数据之后任然需要做强转,不合理.

优化方式: 使用泛型 (了解,可不用写)

```
// map 转为 JavaBean
public static <T> T map2bean(Map<String, Object> map, Class<T> clz) throws
Exception {
    // 创建JavaBean对象
    T obj = clz.newInstance();
    // 遍历属性,获取属性名作为map 的key 去获取value值,再设置给setter 方法
    // 1 获取所有的属性
    BeanInfo beanInfo = Introspector.getBeanInfo(clz, Object.class);
    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
    for (PropertyDescriptor pd : pds) {
        // 2 获取属性名作为map 的key 去获取属性值
        String key = pd.getName();
        // 3 获取到属性值,调用属性的setter方法去设置属性值
        Object value = map.get(key);
        pd.getWriteMethod().invoke(obj,value);
    }
    return obj;
}
```

测试:

叮丁狼教育

```
    @Test
    public  void testBeanToMap() throws Exception {
        Person p = new Person(1L,"小码",true);
        Map<String, Object> map = BeanToMapUtil.bean2map(p);
        System.out.println(map);

        Person p2 = BeanToMapUtil.map2bean(map, Person.class);
        System.out.println(p2);
    }
```

3.6_ 小结

1. 理解和掌握 JavaBean 的规范
2. 理解和掌握什么是JavaBean属性 (写 JavaBean)
3. 理解内省的作用
4. 掌握使用内省去查看和操作 JavaBean 属性
5. 了解 JavaBean 和Map 的转换(写下代码,感受下就OK)

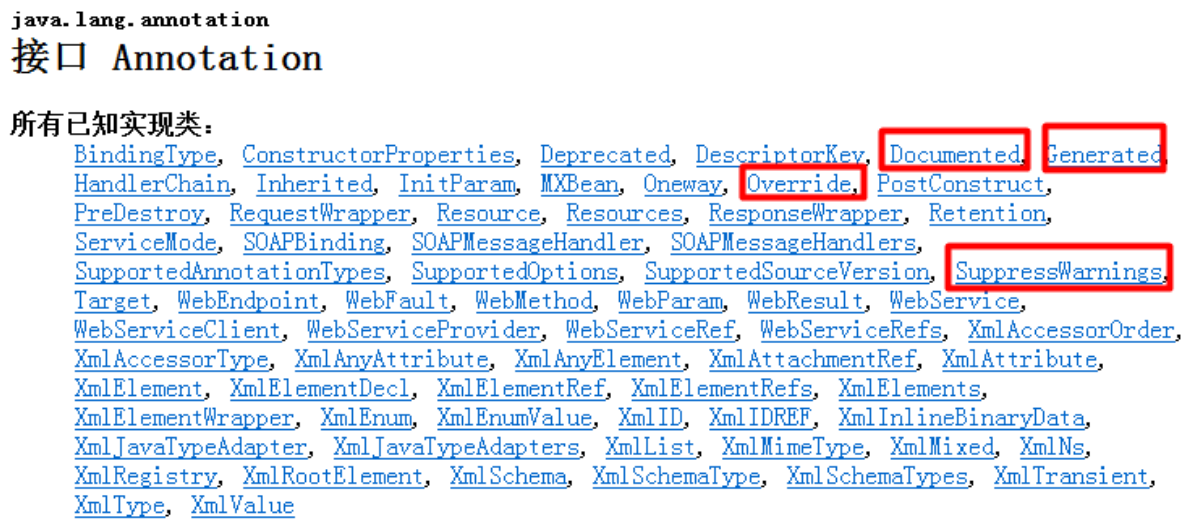
拓展:不做要求,有时间有精力做

1. 使用内省去完成 打印任何 JavaBean 的数据. (拓展)
2. 使用内省去实现 JDBC 中结果集的统一处理(使用工具方法去处理)

4 注解

4.1_注解介绍

到目前为止,其实我们已经用过很多注解了,比如@Test, @Override等等.从 Java5 开始, Java开始对元数据进行支持,这个就是注解.可以使用注解来修饰类中的成员信息.注解其实就是Annotation.


java.lang.annotation
接口 Annotation

所有已知实现类：
BindingType, ConstructorProperties, Deprecated, DescriptorKey, Documented, Generated, HandlerChain, Inherited, InitParam, MXBean, Oneway, Override, PostConstruct, PreDestroy, RequestWrapper, Resource, Resources, ResponseWrapper, Retention, ServiceMode, SOAPBinding, SOAPMessageHandler, SOAPMessageHandlers, SupportedAnnotationTypes, SupportedOptions, SupportedSourceVersion, SuppressWarnings, Target, WebEndpoint, WebFault, WebMethod, WebParam, WebResult, WebService, WebServiceClient, WebServiceProvider, WebServiceRef, WebServiceRefs, XmlAccessorOrder, XmlAccessorType, XmlAnyAttribute, XmlAnyElement, XmlAttachmentRef, XmlAttribute, XmlElement, XmlElementDecl, XmlElementRef, XmlElementRefs, XmlElements, XmlElementWrapper, XmlEnum, XmlEnumValue, XmlID, XmlIDREF, XmlInlineBinaryData, XmlJavaTypeAdapter, XmlJavaTypeAdapters, XmlList, XmlMimeType, XmlMixed, XmlNs, XmlRegistry, XmlRootElement, XmlSchema, XmlSchemaType, XmlSchemaTypes, XmlTransient, XmlType, XmlValue

定义格式 :@interface 注解名

叩丁狼教育

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface 注解名 {
}
```

使用格式 :@注解名(属性名=属性值, 属性名=属性值).

注解的原理: 想要理解注解的原理,我们不得不提到另外一个词语:**标签.** 注解是用来修饰类中的成员信息的,这个修饰的过程非常像贴标签.所以我们也叫注解为标签.日常生活中我们见过的标签:暖男,宅男,渣男,好人.....比如说你被贴上了好人标签,那么每个人心里都知道好人标签就等价于"不合适".之所以能够让每个人都有一样的感觉,是因为,所有人都达成了统一的协议.为好人赋予了另外一层含义:好人=不合适.

回归到代码上来,感觉上,只要使用了 @Override 就表示被贴的方法是从父类或者接口继承过来的.是吗?同理,根据上面的分析,注解也是标签,想要让一个标签具有特殊的含义,也必须有人给它赋予功能.

注解贴在程序元素上,想要拥有某一些功能,.必须有三个角色去参与. 1.注解本身 2.被贴的程序元素 3.第三方程序,使用反射给注解赋予功能.(在注解的背后,一定有一段代码给注解赋予功能).

记住: 注解要有功能,必须要有三方参与:

4.2_内置注解:

@Override 限定覆写父类方法 @Deprecated 标记已过时,不推荐使用.在JDK1.5之前,使用文档注释来标记过时 @SuppressWarings 抑制编译器发出的警告,@SuppressWarings(value="all") @SafeVarargs 抑制堆污染警告(Java7开始出现的)
@FunctionalInterface 标记该接口是一个函数接口(Java8开始出现的)

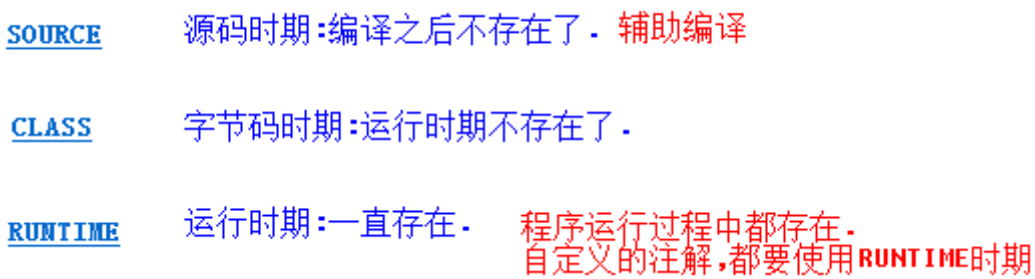
4.3_元注解(理解)

通过对内置注解的操作,发现了一些问题: 1.有的注解可以贴在局部变量上,有的只能贴在方法上.(元注解的约束) 2.有的注解可以有属性名和属性值,但是有的注解没有(注解的定义的内容).

注解：用来贴在类/方法/变量等之上的一个标记，第三方程序可以通过这个标记赋予一定功能 **元注解**：在定义注解的时候用来贴在注解上的注解，用来限定注解的用法

@Target: 表示注解可以贴在哪些位置(类,方法上,构造器上等等).位置的常量封装在ElementType 枚举类中: ElementType.ANNOTATION_TYPE只能修饰Annotation ElementType.CONSTRUCTOR只能修饰构造方法 ElementType.FIELD只能修饰字段(属性),包括枚举常量 ElementType.LOCAL_VARIABLE只能修饰局部变量 ElementType.**METHOD** 只能修饰方法 ElementType.PACKAGE只能修饰包(极少使用) ElementType.PARAMETER只能修饰参数 ElementType.**TYPE**只能修饰类，接口，枚举

@Retention: 表示注解可以保存在哪一个时期. 表示时期的值,封装在RetentionPolicy枚举类中:



@Documented: 使用@Documented标注的标签会保存到API文档中.

@Inherited: @Inherited标注的标签可以被子类所继承.

举例:

```
@Target(RetentionPolicy=RUNTIME)
@Target(ElementType.TYPE)
@Inherited
@interface A{}
@A
class SuperClass{}
class SubClass extends SuperClass{}
//使用反射来检测SubClass,发现SubClass类上也有A标签.因为A标签是可以被继承的.
```

4.4 注解的语法(重点)

目标:对于注解的语法,一个个尝试一下.注解掌握程度暂时不高.只要在有注解源码的情况下,知道贴在哪(Target),有哪些属性(注解的基本语法)即可.

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface VIP {
    // 1.注解中,也是抽象方法,不能有方法体.
    // 2.注解中,抽象方法不叫抽象方法,方法名叫做属性名.方法的返回值是属性值的类型.
    // 3.给属性设置默认值. default
    // 4.如果必须要写的属性只有一个,并且属性名叫做value,可以省略属性名
    // 5.属性值的类型,只能是基本类型, String, Class, annotation, 枚举以及其一维数组
    // 6.数组,如果值只有一个,可以用{}, 也可以不用,如果值有多个,一定要使用{}
    String name() default "碧君";
    int age();
    String[] hobby();
}
```

4.5_ 注解的使用(了解)

```
public static void main(String[] args) {
    // 来获取注解上的内容,然后做业务操作,给贴上的注解赋予功能
    Class<UserInfo> clz = UserInfo.class;
    // 在类上去获取注解
    Annotation[] anns = clz.getAnnotations();
    for (Annotation ann: anns) {
        System.out.println(ann);
    }
    // 直接获取注解
    VIP vip = clz.getAnnotation(VIP.class);
    // 获取数据中数据,然后赋予贴的注解特殊的功能
    System.out.println(vip.name());
    System.out.println(Arrays.toString(vip.hobby()));
    // 把VIP数据保存到当前用户上
}
```

叮丁狼教育