* 学习目标
 * 能够掌握自定义注解的语法
   * public @interface MyAnnotation-->默认是编译阶段，改成运行阶段
   * 注解的本质就是接口
     * 属性方法:int age();
     * 基本数据类型，String,Class,注解类型，枚举,以上一维数组
 * 能够掌握元注解
     * @Target（ElementType.Type,Method,Field,Package,Constructor,Paramter...）
     * @ Retention(RetentionPolicy.Source,Class,Runttime)
     * @Document
     * @Inherted
     * @Repeatable
 * 能够掌握注解的反射
     * AnnotatedElement:是接口，直接或者间接实现子类：Package,Class,Method,Field
     * Constructor,Paramter
     *AnnotatedElement
       * isAnntotationPresent
       * getAnnotations(),getAnnotation(Class)
       * getDeclaredAnnotations(),getDeclaredAnnotation(Class)
       * getAnnotationsByType().getDeclaredAnnotationsByType()
       * @Repeatable
     * 案例：@Value案例
 * 能够掌握常用自定义注解的案例一
     * @Test
 * 能够掌握常用自定义注解的案例二
    * @Table，@Colunm
    * 多线程概述

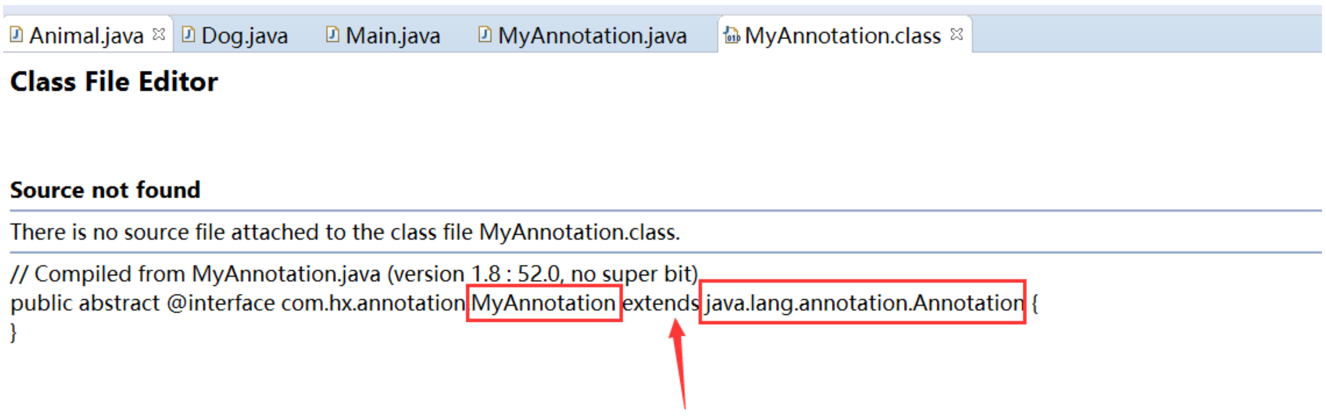
----

* 回顾
  * JSON：JavaScript Object Notation
    * 存储数据，数据交换（传输），独立语言
    * json 比 xml 更小，更快，更易解析
    * 语法：{“key”:"value",“key1”:[{}，{}]}
    * 在js中key表示一个对象，jsonStr---js的对象 eval
    * gson,fastjson,jackson
      * gson:toJson,fromJson
        * new TypeToken(List<Book>){}.getType();
      * fastjson:JSON.toString,JSON.parseObject,JSON.parseArray
  * 注解
    * JDK1.5
    * 注释和注解的区别
    * 现在开发是注解的天下
    * @Override,@Deprecated,@SupressWarning,@SafeVarArgs,@FunctionInterface
    * LomBok:简化开发，让代码更整洁，少写get/set,toString,equals,hashcode,构造器
      * @Data，@Setter,@Getter,@NoArgsConstructor,@AllArgsConstructor

        @ToString，....

        @Log,@Log4j,@Log4j2,@SLFJ,....

        @NonNull,@CleanUp
      * var,val
* 能够掌握自定义注解的语法
  * 声明一个注解 @interface 注解名 { }
    * 例如：public @interface MyAnnotation{}
  * 在eclipse查看class文件（在idea默认把class文件反编译出来了，不好观察）

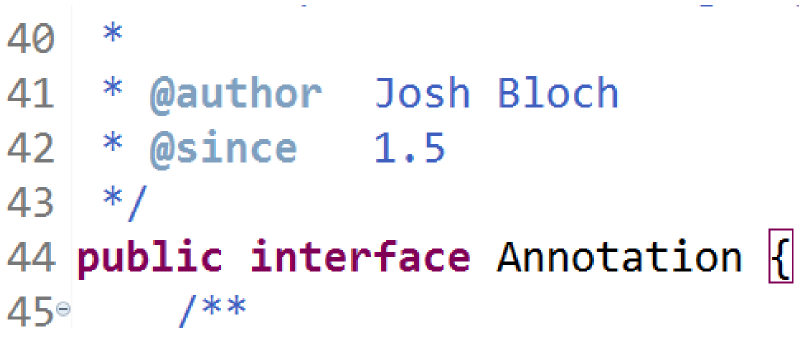

Animal.java Dog.java Main.java MyAnnotation.java MyAnnotation.class

Class File Editor

Source not found

There is no source file attached to the class file MyAnnotation.class.

```
// Compiled from MyAnnotation.java (version 1.8 : 52.0, no super bit)
public abstract @interface com.hx.annotation.MyAnnotation extends java.lang.annotation.Annotation {
}
```



```
 *
 * @author  Josh Bloch
 * @since   1.5
 */
public interface Annotation {
    /**
```

* 注解它的本质就是一个接口，这个接口需要继承 Annotation接口

* 接口中可以有属性方法

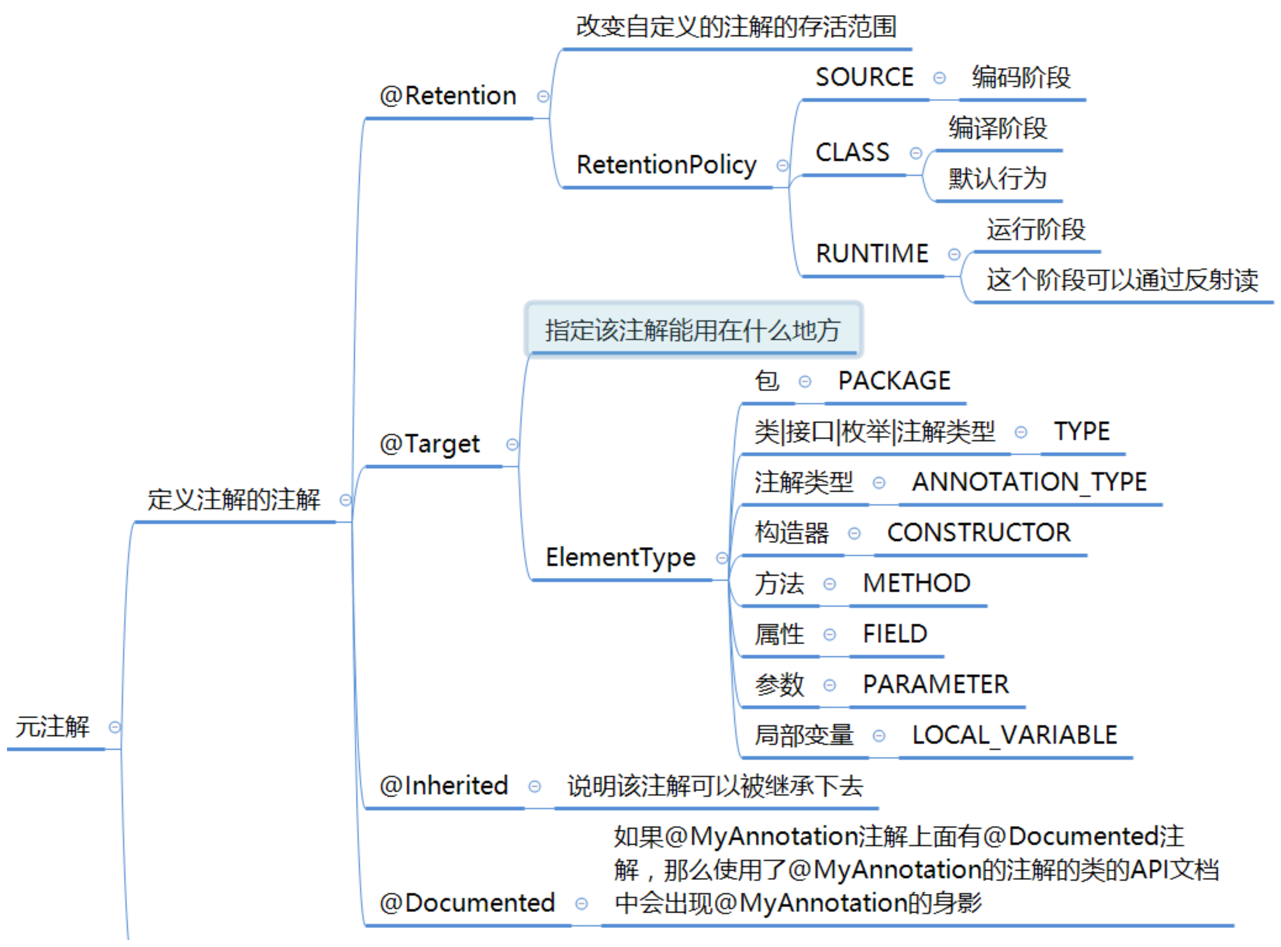
* 例：int age();

* 注解的属性类型只能用基本数据类型(char，boolean,byte、short、int、long、float、double)和String、Enum、Class、annotations数据类型,以及这一些类型的数组

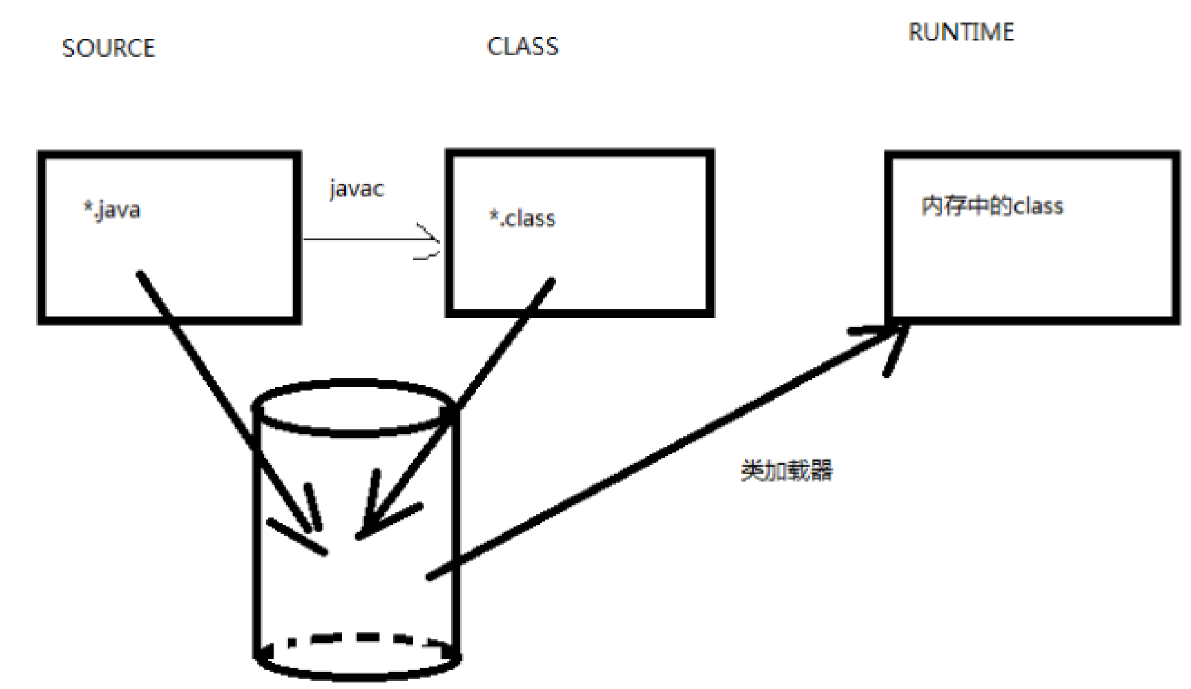
```
* 案例（演示语法糖）
* 自定义注解
public @interface MyAnnotation {
    int age() default 0;
    String name();
    String[] values() default {"1","2"};
}
* 使用注解
@MyAnnotation(name = "xiaohei1")
public class User {
    @MyAnnotation(name = "xiaohei2")
    private String name;
    @MyAnnotation(name = "xiaohei3")
    public void sayHello(){
    }
}

* 温馨提醒
  * 注解可以在类上，属性，方法上使用
  * 注解声明的值,可以通过在运行时，反射获取
```

* 能够掌握元注解
  * JDK1.5 的元注解



* 查看@Override,@SuppressWarnings,@Data源码和测试用法，理解上述概念
* 自定义的注解的存活范围（生命周期）：默认是CLASS。



* JDK1.8的元注解

- 元注解
  - 定义注解的注解
    - @Repeatable
      - 标记的注解可以多次应用于相同的声明或类型
  - 专门讲解

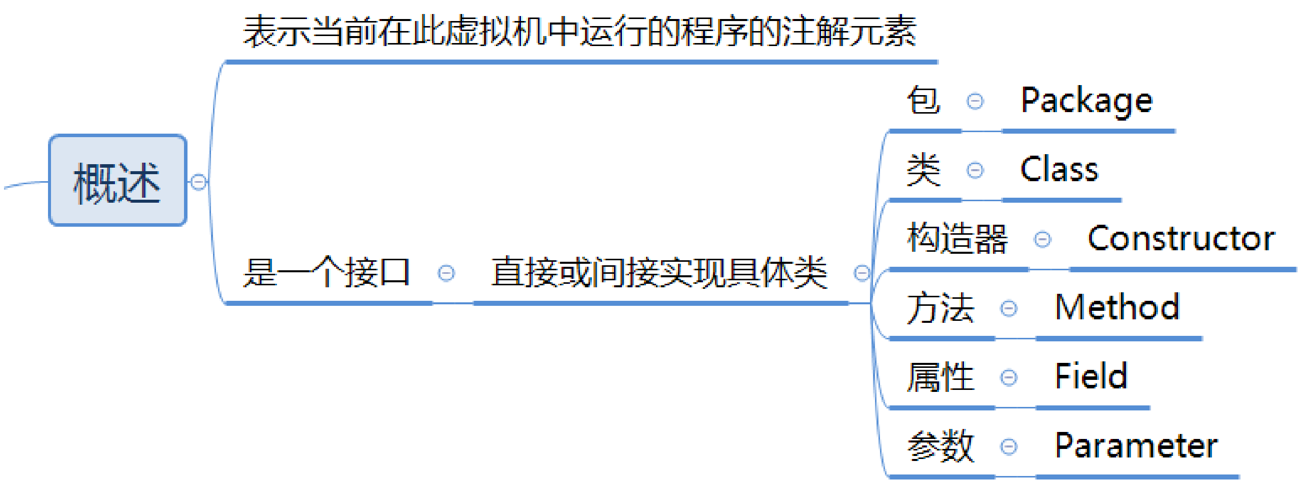
```java
 * 案例：一个人是具有多种身份，用注解表示他
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Persons {
    Person[] value();
}
@Repeatable(Persons.class)
public @interface Person {
    String role() default "";
}
@Person(role ="CEO")
@Person(role = "husband")
@Person(role = "father")
public class Man {
}
```

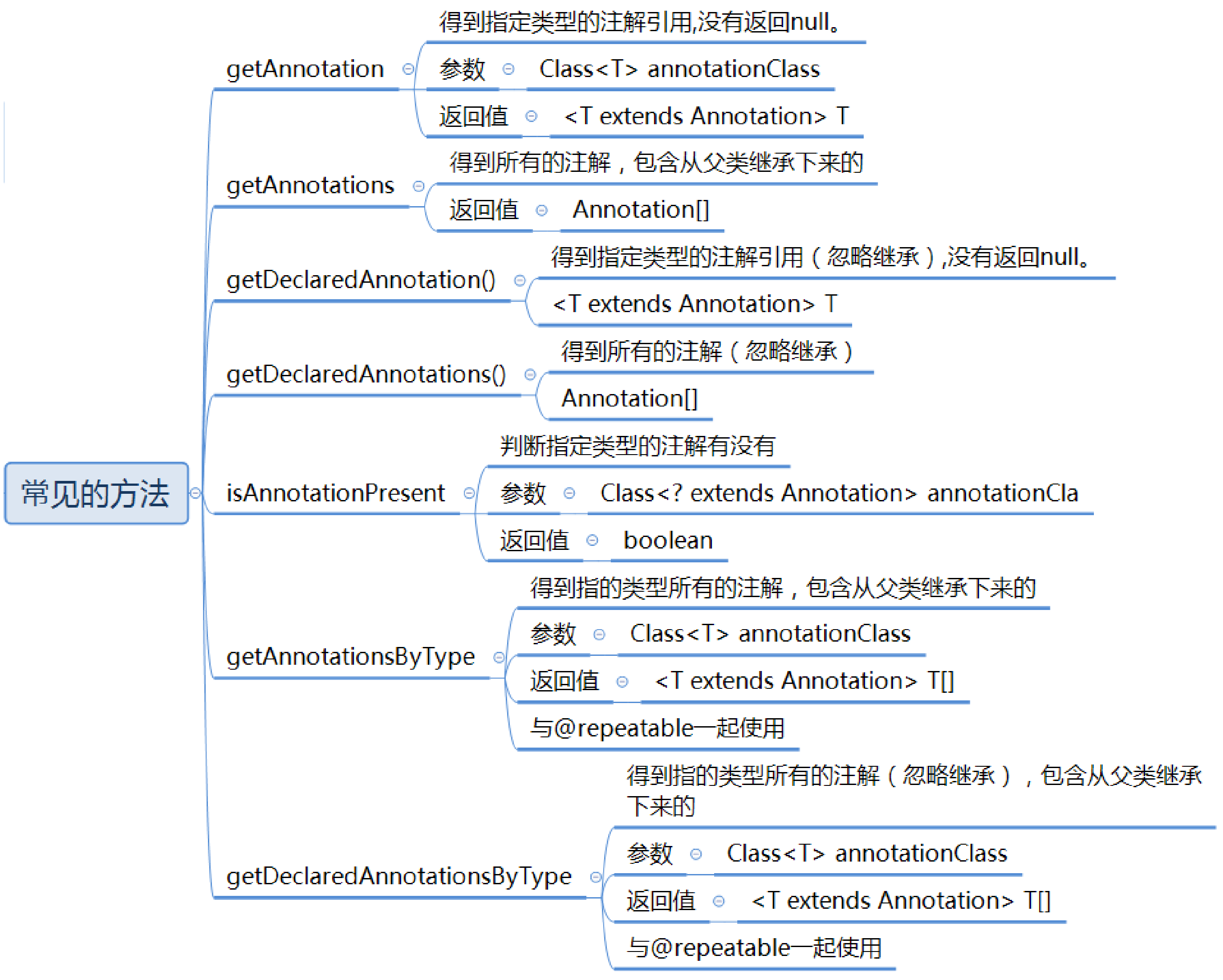
* 能够掌握注解的反射

  * 复习反射：

    * 参考09-JDBC高级新2

  * 反射注解类：AnnotatedElement




常见的方法
getAnnotation
得到指定类型的注解引用,没有返回null。
参数 Class<T> annotationClass
返回值 <T extends Annotation> T
getAnnotations
得到所有的注解，包含从父类继承下来的
返回值 Annotation[]
getDeclaredAnnotation()
得到指定类型的注解引用（忽略继承）,没有返回null。
<T extends Annotation> T
getDeclaredAnnotations()
得到所有的注解（忽略继承）
Annotation[]
isAnnotationPresent
判断指定类型的注解有没有
参数 Class<? extends Annotation> annotationCla
返回值 boolean
getAnnotationsByType
得到指的类型所有的注解，包含从父类继承下来的
参数 Class<T> annotationClass
返回值 <T extends Annotation> T[]
与@repeatable一起使用
getDeclaredAnnotationsByType
得到指的类型所有的注解（忽略继承），包含从父类继承下来的
参数 Class<T> annotationClass
返回值 <T extends Annotation> T[]
与@repeatable一起使用

```java
* 案例一
@Target({ElementType.TYPE,ElementType.METHOD,ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    int age() default 0;
    String name();
    String[] values() default {"1","2"};
}
@MyAnnotation(name = "xiaohei1")
public class User {
    @MyAnnotation(name = "xiaohei2")
    private String name;
    @MyAnnotation(name = "xiaohei3")
    public void sayHello(){
    }
}
 @Test
    public void test1() throws Exception {
       // 类上的注解
       Class clazz=User.class;
        if(clazz.isAnnotationPresent(MyAnnotation.class)){
            MyAnnotation annotation = (MyAnnotation) clazz.getDeclaredAnnotatio
            System.out.println(annotation.name());
            System.out.println(annotation.age());
            String[] values=annotation.values();
            for (String value : values) {
                System.out.println(value);
            }
        }
        System.out.println("----------------------");
        // 属性上
        Field field = clazz.getDeclaredField("name");
        field.setAccessible(true);
        if(field.isAnnotationPresent(MyAnnotation.class)){
            MyAnnotation annotation = field.getDeclaredAnnotation(MyAnnotation.
            System.out.println(annotation.name());
        }
        System.out.println("----------------------");
      // 方法上的注解
        Method method = clazz.getDeclaredMethod("sayHello");
        if(method.isAnnotationPresent(MyAnnotation.class)){
            MyAnnotation annotation=method.getDeclaredAnnotation(MyAnnotation.c
            System.out.println(annotation.name());
        }
    }
* 结果
xiaohei1
0
1
2
----------------------
xiaohei2
----------------------
xiaohei3

* 案例二：（定义框架--通过配置文件，配置默认属性的值）
  * 定义注解
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Value {
    String value();
}
  * 使用注解
    * 配置文件 params.properties
        name=xiaohei
        age=28
        mobile=15918747136
    * 使用注解
 @Data
public class User {
    @Value("name")
    private String name;
    @Value("age")
    private int age;
    private String mobile;
}
    * 框架的代码
  public class Main1 {
    public static void main(String[] args) throws Exception {
         Person person=new Person();
    System.out.println(person);
    executeParams(person);
    System.out.println(person);
  }

  // 写框架或者平台人写的
  private static void executeParams(@NonNull Object obj) throws Exception {
    Properties properties=new Properties();
    properties.load(Main1.class.getClassLoader().getResourceAsStream("param
    Class<?> clazz = obj.getClass();
    Field[] fields = clazz.getDeclaredFields();
    for (Field field : fields) {
      if(!field.isAnnotationPresent(Value.class)){
        continue;
      }
      Value v = field.getAnnotation(Value.class);
      String key = v.value();
      String property = properties.getProperty(key);
      field.setAccessible(true);
      if(field.getType()==int.class){
        field.set(obj,Integer.parseInt(property));
        continue;
      }
      field.set(obj,property);
    }
  }
}
* 案例三:
 // jdk1.5的实现方式
  @Test
  public void test2(){
    Class clazz=Man.class;
    if(clazz.isAnnotationPresent(Persons.class)){
      Annotation[] annotations = clazz.getAnnotations();
      Persons persons= (Persons) annotations[0];
      for (Person person : persons.value()) {
        System.out.println(person.role());
      }
    }
  }
```

```

    // jdk1.8的实现方式
    @Test
    public void test3(){
        Class clazz=Man.class;
        if(clazz.isAnnotationPresent(Persons.class)){
            Person[] persons = (Person[]) clazz.getDeclaredAnnotationsByType(Pe
            for (Person person : persons) {
                System.out.println(person.role());
            }
        }
    }
 * 结果
 CEO
 husband
 father

```

* 能够掌握常用自定义注解的案例一

```

* 模拟单元测试
 * 定义MTest注解
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MTest {
}
 * 编程测试类
 public class Test2 {
    @MTest
    public void test1(){
        System.out.println("执行test1...");
    }

    public void test2(){
        System.out.println("执行test2...");
    }

    @MTest
    public void test3(){
        System.out.println("执行test3...");
    }
}
 * 编写执行注解的类
 public class Main {
    public static void main(String[] args) throws Exception {
        executeTest();
    }
    public static void executeTest() throws Exception {
        Class clazz=Test2.class;
        Method[] methods = clazz.getDeclaredMethods();
        for (Method method : methods) {
            if(method.isAnnotationPresent(MTest.class)){
                method.invoke(clazz.newInstance());
            }
        }
    }
}
 * 执行结果
 执行test3...
 执行test1...
```

* 能够掌握常用自定义注解的案例二

```
* 案例：自定义Table、Column生成拼接SQL语句功能
* 定义注解
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Table {
    String value();
}
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Column {
    String value();
}

* 定义User类，使用注解
@Data
@NoArgsConstructor
```

```java
@AllArgsConstructor
@Table("user")
public class User {
    @Column("id")
    private int id;
    @Column("username")
    private String username;
    @Column("password")
    private String password;
    @Column("age")
    private int age;
    @Column("mobile")
    private String mobile;
}
* 定义Main1
public class Main1 {
    public static void main(String[] args) throws Exception {
        User user1=new User();
        user1.setId(1001);
        // select * from table_name where 1=1 and id=10001;
        String sql1 = query(user1);
        System.out.println(sql1);

        User user2=new User();
        user2.setUsername("xiaohei");
        user2.setPassword("123");
        // select * from table_name where 1=1 and username='xiaohei' and passwo
        String sql2 = query(user2);
        System.out.println(sql2);

        User user3=new User();
        user3.setMobile("15918747136,15815834856");
        // select * from table_name where 1=1 and mobile in("15918747136"."1591
        String sql3 = query(user3);
        System.out.println(sql3);
    }

    public static String query(Object obj) throws Exception {
        Class clazz=obj.getClass();
        //判断有没有Table注解
        if(!clazz.isAnnotationPresent(Table.class)){
            return null;
        }
        Table table = (Table) clazz.getAnnotation(Table.class);
        String tableName=table.value();
        StringBuilder sb=new StringBuilder();
        String sql="select * from "+tableName+" where 1=1";
        sb.append(sql);
        // 获得所有字段
        Field[] fields = clazz.getDeclaredFields();
        for (Field field : fields) {
            field.setAccessible(true);
            //检查字段是否有colunm注解
            if(!field.isAnnotationPresent(Column.class)){
                continue;
            }
            Column colunm = field.getAnnotation(Column.class);
            String colunmName=colunm.value();
            Object param=field.get(obj);
            if(param==null){
                continue;
            }
            if(param instanceof Integer && (Integer)param==0){
                continue;
            }
            if(param instanceof String){
                if(((String) param).contains(",")){
                    String[] values = ((String) param).split(",");
                    sb.append(" and ").append(colunmName).append(" in (");
                    for (String value : values) {
                        sb.append("'").append(value).append("'").append(",");
                    }
                    sb.deleteCharAt(sb.length()-1);
                    sb.append(")");
                }else{
                    sb.append(" and ").append(colunmName).append("=").append("'
                }
            } else {
                sb.append(" and ").append(colunmName).append("=").append(param)
            }

        }
        return sb.toString();
    }
}

* 结果
select * from user where 1=1 and id=1001
select * from user where 1=1 and username='xiaohei' and password='123'
select * from user where 1=1 and mobile in ('15918747136','15815834856')
```