学习目标

- [ ] 理解为什么要使用反射
- [ ] 理解什么是字节码对象
- [ ] 熟练掌握获取字节码对象的三种方法
- [ ] 熟练掌握反射操作构造器/方法

1_为什么要使用反射(理解)

> **需求**: 书写一个工具方法,获取打印传入的 JavaBean 中的数据.

```
// 测试代码
@Test
public void testUtil(){
    User user = new User("小普",18);
    // 调用工具方法打印对象
    JavaBeanUtil.printJavaBean(user);
    // JavaBeanUtil.printJavaBean(person);
}

// 工具类
class JavaBeanUtil{
    public static void printJavaBean(Object obj){
        // 如果知道调用者传入的真实类型是User,代码如下
        User user = (User)obj;
        System.out.println(user.getName());
        System.out.println(user.getAge());
    }
}
```


上面代码存在的问题:

1. printJavaBean 方法中要获取传入的 User 对象的数据,就必须强转.
2. 咱们写的是牛逼的工具类,**在不知调用者带了什么对象时,无法强转**,强转则有风险.
3. 在 printJavaBean 方法中已经拿到对象了,却**得不到对象的真实类型**.

解决方案: 使用反射 (可以通过一个对象,获取到它的真实类型以及调用该对象中的方法).

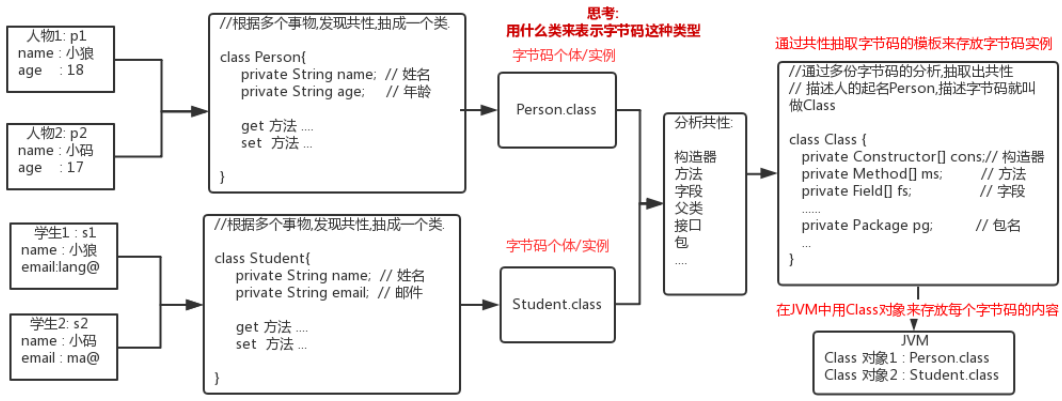
2_反射

在程序运行过程中，通过**字节码对象**，去获取到类中的成员信息（构造器，方法，字段）,这就称为反射.

目的是通过程序自动获取构造器来创建对象.获取到方法并调用等等.

2.1_字节码对象(理解)

在 Java 中,万物皆对象.我们可以通过多个事物,发现他们的共性,来抽象成一个类,类就是对象的模板,而一个个的个体,就是对象. 比如人类和学生.



字节码也是真实存在的文件,每个字节码都是一个个例,而 JVM 要来存放这些字节码就需要抽象成模板,再通过模板来创建对象,存放每份字节码的信息.当要使用某份字节码时(例如要创建Person对象),就从 JVM 中调出存了Person.class 内容的Class 对象,然后拿去创建 Person 对象.

JDK 中定义好的 Class 类: java.lang.Class

Field	**getField**(String name) 返回一个 Field 对象，它反映此 Class 对象所表示的类或接口的指定公
Field[]	**getFields**() 返回一个包含某些 Field 对象的数组，这些对象反映此 Class 对象所表
Type[]	**getGenericInterfaces**() 返回表示某些接口的 Type，这些接口由此对象所表示的类或接口直接实
Type	**getGenericSuperclass**() 返回表示此 Class 所表示的实体（类、接口、基本类型或 void）的直接
ss<?>[]	**getInterfaces**() 确定此对象所表示的类或接口实现的接口。
Method	**getMethod**(String name, Class<?>... parameterTypes) 返回一个 Method 对象，它反映此 Class 对象所表示的类或接口的指定公
Method[]	**getMethods**() 返回一个包含某些 Method 对象的数组，这些对象反映此 Class 对象所表示的类或接口（包括从超类和超接口继承的那些的类或接口）的公共 *member* 方法。
int	**getModifiers**() 返回此类或接口以整数编码的 Java 语言修饰符。
String	**getName**() 以 String 的形式返回此 Class 对象所表示的实体（类、接口、数组类、
Package	**getPackage**() 获取此类的包。

该类中有大量的 get 开头的方法.表示可以使用字节码对象来获取信息.**所以当我们拿到了字节码对象，就可以直接操作当前字节码中的构造器,方法,字段.**

2.2_获取字节码对象(掌握)

通过 API ,我们得知 Class ,没有公共的构造器,其原因是Class 对象是在加载类时由 Java 虚拟机自动构造的。

获取字节码的方式:

1. 通过 Class 类的 forName() 方法来获取字节码对象
    - Class.forName(String classsName) : 通过**类的全限定名**获取字节码对象
    - 全限定名: 包名.类型
    - Class.forName("java.lang.String"); // JVM 中存在则返回,不存在则加载
2. 通过对象的 getClass() 方法来获取字节码对象
    - new User().getClass(); 使用的是父类 Object 中的 getClass() 方法
3. 通过类型(基本类型)的 class 字段来获取字节码对象
    - int.class

上面三种方式应用都比较多,尤其是第一种方式,通过 Class.forName,大量的在框架中使用.

实例:

```
@Test
public void testGetClass() throws Exception {
    // 1 使用Class.forName
    Class clz1 = Class.forName("cn.wolfcode._04_reflect.Person");
    // 2 使用对象的getClass()
    Class clz2 = new Person().getClass();
    // 3 使用 class 字段
    Class clz3 = Person.class;
}
```


思考:

1. 三种方式获取到的字节码是同一个吗?
2. int 类型和 int[] 它们的字节码是同一个吗?

```
@Test
public  void testGetClass() throws Exception {
    // 1 通过类的全限定名 Class.forName();
    Class clz1 = Class.forName("cn.wolfcode._01_reflect.Person");
    // 2 通过对象的getClass() 方法
    Person p = new Person();
    Class clz2 = p.getClass();
    // 3 通过class 字段去获取
    Class clz3 = Person.class;
    // 字节码只会加载一次,所有不管用的哪种方式去获取字节码,都是同一个
    System.out.println(clz1 == clz2);   //true
    System.out.println(clz2 == clz3);   //true
    System.out.println(clz1 == clz3);   //true
```

```
    // int 类型和int数据类型不是同一个
    System.out.println(int.class);
    System.out.println(int[].class);
}
```

2.3_获取构造器

2.3.1_ 正常调用方法的流程

1. 非 static 修饰的方法: 创建对象,调用方法
2. static 修饰的方法 : 类名调用方法

> 使用反射其目的无外乎就是使用程序动态操作类的成员,比如方法,而要操作方法首先得有对象,而对象是通过构造器来创建的,所以需要先获取构造器.

2.3.2_反射操作构造器

使用反射获取构造器 :

- 获取所有的构造器 public Constructor<?>[] getConstructors(): 获取所有的 public 修饰的构造器 public Constructor<?>[] getDeclaredConstructors(): 获取所有的构造器(包括非public)
- 获取指定的构造器 public Constructor getConstructor(Class... parameterTypes) public Constructor getDeclaredConstructor(Class... parameterTypes): parameterTypes : 参数的类型(构造方法的参数列表的类型).

  注意: 找构造器/方法,传递的是参数的类型.

  结论 : 带着 s 表示获取多个.带着 Declared 表示忽略权限,包括私有的也可以获取到.

实战:

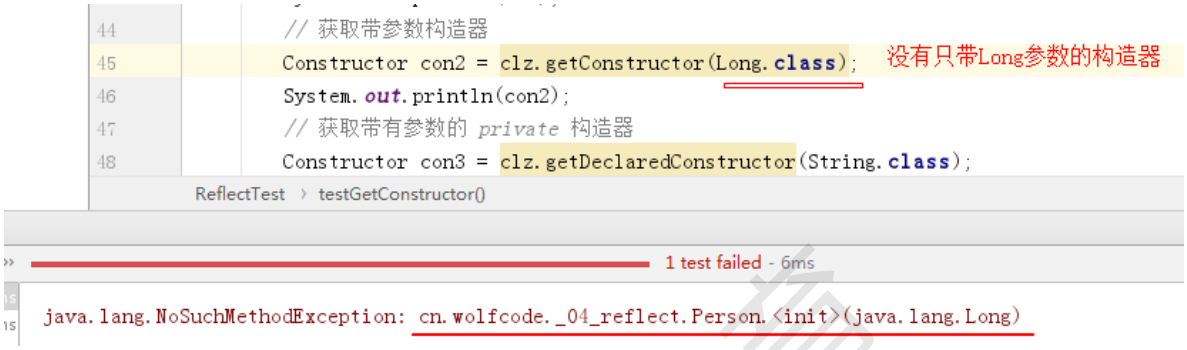
> 1 获取所有 public 构造器 2 获取所有构造器,包括 private 3 获取无参构造器 4 获取带参构造器 5 获取指定 private 构造器

代码:

```
@Test
public void testGetConstructor() throws NoSuchMethodException {
    // 获取字节码对象
    Class clz = Person.class;
    // 1 获取所有 public 构造器
    Constructor[] cons1 = clz.getConstructors();
    for(Constructor con : cons1){
        System.out.println(con);
    }
    System.out.println("--------");
    // 2 获取所有构造器,包括 private
    Constructor[] cons2 = clz.getDeclaredConstructors();
    for(Constructor con : cons2){
        System.out.println(con);
    }
```

```
    // 3 获取无参构造器
    Constructor con1 = clz.getConstructor();
    System.out.println(con1);
    // 4 获取带参构造器
    Constructor con2 = clz.getConstructor(Long.class, String.class);
    System.out.println(con2);
    // 5 获取指定 private 构造器
    Constructor con3 = clz.getDeclaredConstructor(String.class);
    System.out.println(con3);
}
```

常见错误: 参数不匹配,报错.找不到指定的构造器



调用构造器创建对象

```
public Object newInstance(Object... initargs)
// initargs: 调用该构造器传递的实际参数.参数列表一定要匹配(类型,个数,顺序).
```

实战:

```
@Test
public void testCreateObject() throws Exception {
    // 获取字节码对象
    Class clz = Class.forName("cn.wolfcode._04_reflect.Person");
    // 获取带参数构造器,参数为参数类型
    Constructor con1 = clz.getConstructor(Long.class, String.class);
    //调用构造器
    Object obj = con1.newInstance(1L, "小狼");
    System.out.println(obj);
    // 获取带有参数的 private 构造器
    Constructor con2 = clz.getDeclaredConstructor(String.class);
    // 调用私有构造器,必须先设置为可访问
    con2.setAccessible(true);
    Object obj2 = con2.newInstance("小码");
    System.out.println(obj2);
}
```

注意: 不能直接访问没有权限(非public)的成员,如果想要使用反射去操作非public的成员.必须设置一个可以访问的标记.

> public void setAccessible(boolean flag): 传递一个true,表示可以访问,表示不管权限.

从 API 中我们可以发现,Constructor,Field,Method是 AccessibleObject 的子类,因为这三种成员都是可以被访问private 修饰符修饰的.

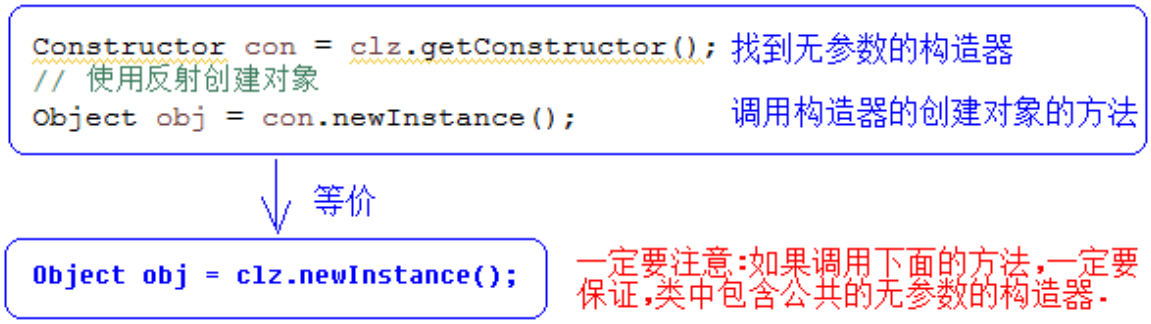
代码:

```
@Test
public void testCreateObject() throws Exception {
    // 获取带有参数的 private 构造器
    Constructor con2 = clz.getDeclaredConstructor(String.class);
    // 调用私有构造器,必须先设置为可访问
    con2.setAccessible(true);
    Object obj2 = con2.newInstance("小码");
    System.out.println(obj2);
}
```


创建对象的快捷方式: 调用公共无参构造器

一般来说,大多数类都有公共的无参数的构造器.使用反射创建对象,需要 1.找到公共的无参数的构造器, 2.调用构造器的newInstance.

SUN公司为了方便我们调用**公共的无参数**的构造器来创建对象,提供了一个方便的方法. **Class对象.newInstance()**;



经验 : 只要看到传入全限定名,基本上都是要使用反射,通过全限定名来获取字节码对象. 只要看到无指定构造器但是能创建对象,基本上都是要通过字节码对象的 newInstance 去创建对象.

2.4_获取操作方法

在反射中,使用 Method 类来描述方法这一类事物.

2.4.1_反射获取方法

获取所有方法:

- public Method[] getMethods(): 可以获取到所有的公共的方法,包括继承的.+
- public Method[] getDeclaredMethods():获取到本类中所有的方法,包括非public的,不包括继承的.

获取指定的方法:

- public Method getMethod(String name, Class<?>... parameterTypes):
- public Method getDeclaredMethod(String name, Class<?>... parameterTypes): name: 方法名 parameterTypes: 当前方法的参数列表的类型.
- 注意,要找到某一个指定的方法,必须要使用方法签名才能定位到.而方法签名=方法名+参数列表,还记得我们获取构造器的经验吗?带着s表示获取多个,带着declared表示忽略访问权限.

实战

> 1 获取所有 public 方法,包括父类的 2 获取所有方法,包括 private 不包括父类的 3 获取指定参数的 public 的方法,包括父类的 4 获取指定参数的private 方法,不包括父类的

代码

```
@Test
public  void testGetMethod() throws Exception {
    // 1 获取字节码对象
    Class clz = Class.forName("cn.wolfcode._01_reflect.Person");
    // 2 获取构造器来创建对象
    // 3 获取方法
    //1 获取所有 public 方法,包括父类的
    Method[] methods = clz.getMethods();
    for(Method m : methods){
        System.out.println(m);
    }
    System.out.println("---------");
    //2 获取所有方法,包括 private 不包括父类的
    Method[] methods2 = clz.getDeclaredMethods();
    for(Method m : methods2){
        System.out.println(m);
    }
    System.out.println("---------");
    //3 获取指定参数的 public 的方法,包括父类的
    Method sayHelloMethod = clz.getMethod("sayHello", String.class);
    System.out.println(sayHelloMethod);
    //4 获取指定参数的private 方法,不包括父类的
    Method doWorkMethod = clz.getDeclaredMethod("doWork", String.class);
    System.out.println(doWorkMethod);
}
```

2.4.2_调用方法

API :

public Object invoke(Object obj, Object... args): obj: 表示调用该方法要作用到那个对象上. args:调用方法的实际参数 方法的返回值表示,调用该方法是否有返回值,如果有就返回,如果没有,返回null.

传统创建对象调用方法: Person p = new Person(1L, "小狼"); p.sleep(5);// 小狼,睡,5个时辰.

使用反射创建对象调用方法: Method m = clz.getMethod("sleep", int.class);// 找到sleep方法. m.invoke(obj, 5);// 睡,小狼,5个时辰.

实战:

```
@Test
public  void testGetMethod() throws Exception {
    // 1 获取字节码对象
    Class clz = Class.forName("cn.wolfcode._01_reflect.Person");
    // 2 获取构造器来创建对象
    Object obj = clz.newInstance(); // 使用公共的无参数的构造器
    // 3 获取方法
    //1 获取所有 public 方法,包括父类的
    Method[] methods = clz.getMethods();
    for(Method m : methods){
        System.out.println(m);
    }
    System.out.println("---------");
    //2 获取所有方法,包括 private 不包括父类的
    Method[] methods2 = clz.getDeclaredMethods();
    for(Method m : methods2){
        System.out.println(m);
    }
    System.out.println("---------");
    //3 获取指定参数的 public 的方法,包括父类的
    Method sayHelloMethod = clz.getMethod("sayHello", String.class);
    System.out.println(sayHelloMethod);
    // 调用方法
    Object val1 = sayHelloMethod.invoke(obj, "小狼");

    System.out.println("值1:" + val1);
    //4 获取指定参数的private 方法,不包括父类的
    Method doWorkMethod = clz.getDeclaredMethod("doWork", String.class);
    System.out.println(doWorkMethod);
    // 设置可访问
    doWorkMethod.setAccessible(true);
    // 调用私有的方法
    doWorkMethod.invoke(obj,"小码");
}
```

注意:

1. 方法也是可以被访问私有修饰符修饰的,所以,如果要访问非 public 修饰的方法,需要在访问之前设置可访问 method.setAccessible(true);

2. 如果调用的是静态方法,是不需要对象的,所以此时在invoke方法的第一个参数,对象直接传递一个 null 即可.

```
 // 调用静态方法
Method staticSayHelloMethod = clz.getDeclaredMethod("sayHello", String.class, Long.class);
// 不需要对象去调用,但是参数必须加上null,不然会把后面的参数作为调用方法的对象了.
staticSayHelloMethod.invoke(null,"小明",1L);
```

Person.java

```
package cn.wolfcode._01_reflect;

public class Person {
    private Long id;
    private String name;
    public Person() {
    }
    public Person(Long id, String name) {
        this.id = id;
        this.name = name;
    }
    private Person(String name) {
        this.name = name;
    }
    public void sayHello(){
        System.out.println("hello");
    }
    public String sayHello(String name){
        System.out.println(name + ": hello");
        return "您好";
    }
    public static void sayHello(String name,Long id){
        System.out.println("调用静态方法");
    }
    private void doWork(){
        System.out.println("doWork");
    }
    private void doWork(String name){
        System.out.println(name + ": doWork");
    }
    // getter方法 setter 方法
    public String toString() {
        return "Person{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```


2.5_操作字段(了解)

在反射中,使用 Field 去描述字段这一类事物.

2.5.1_获取字段

获取单个字段:

public Field getField(String name) : name 要获取的字段的名称 public Field getDeclaredField(String name) :

获取所有字段

public Field[] getFields() public Field[] getDeclaredFields()

代码:

```
@Test
public void testField() throws Exception {
    // 1 获取字节码对象
    Class clz = Person.class;
    Object obj = clz.newInstance();
    // 2 获取字段
    Field[] fs = clz.getFields();
    for(Field f: fs){
        System.out.println(f);
    }
    Field[] fs2 = clz.getDeclaredFields();
    for(Field f: fs2){
        System.out.println(f);
    }
    // 获取单个字段
    Field nameField = clz.getDeclaredField("name");
    System.out.println(nameField);
}
```

2.5.2_操作字段

get(Object obj);

set(Object obj,Object value)

```
// 设置私有字段可访问
nameField.setAccessible(true);
// 操作name字段
// 设置那么字段的数据
nameField.set(obj,"小狼");
// 获取name字段的数据
Object nameValue = nameField.get(obj);
System.out.println(nameValue);
```

小结

1. 理解为什么需要使用反射

2. 掌握获取字节码对象的方式(有代码可以体现获取的方式)
3. 掌握通过字节码对象去获取构造器以及调用构造器
4. 掌握方法的获取和调用

拓展: 使用反射去完成 工具方法来答应 JavaBean 中的数据(不做要求)