主讲教师　张 智
计算机学院软件工程系
课程群: 421694618

13 反射机制

13.1 反射概念

■ 反射机制：

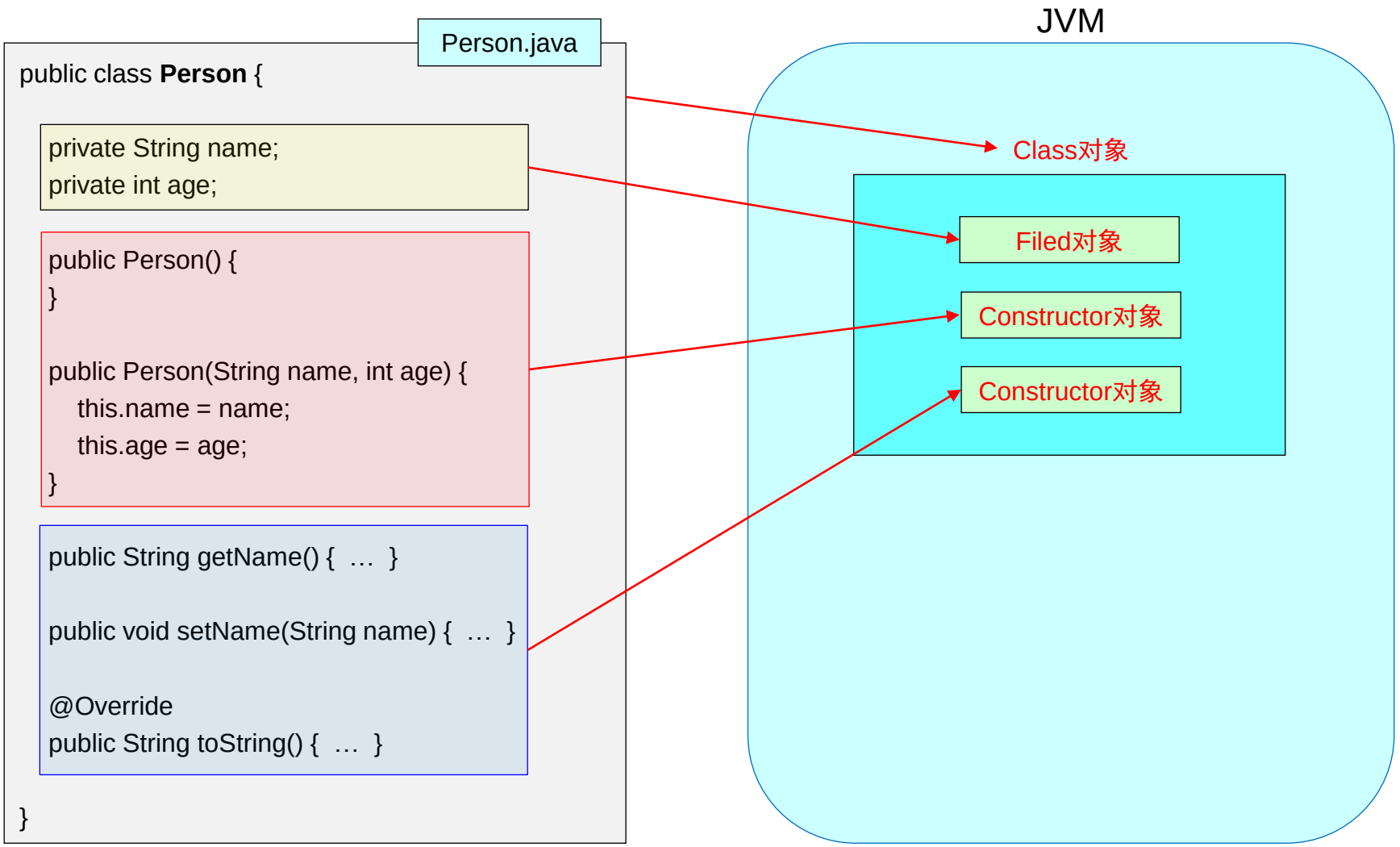
在运行状态中，对于任意一个类，都能知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为反射机制。

- Java反射机制的核心是在程序运行时动态加载类并获取类的详细信息，从而操作类或对象的属性和方法
- 反射机制本质是JVM得到Class对象之后，再通过Class对象进行反编译，从而获取对象的各种信息（下页图示）

反射机制主要功能

- 在运行时判断任意一个对象所属的类
- 在运行时构造任意一个类的对象
- 在运行时判断任意一个类所具有的成员变量和方法
- 在运行时调用任意一个对象的方法，甚至调用private方法
- 生成动态代理

上述功能都是在运行时环境中，而不是在编译时环境中


Person.java
public class Person {
private String name;
private int age;
public Person() {
}
public Person(String name, int age) {
this.name = name;
this.age = age;
}
public String getName() { … }
public void setName(String name) { … }
@Override
public String toString() { … }
}
JVM
Class对象
Filed对象
Constructor对象
Constructor对象

【返回】

13.2 反射示例

Person.java

```
package edu.wust.examples;
public class Person {
    private String name;
    private int age;

    public Person() {
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "name=" + name + ", age=" + age;
    }
}
```

```
class Test {
    public static void main(String[] args) throws Exception {
        // 1、获取类的 Class 对象实例
        Class claz = Class.forName("edu.wust.examples.Person");

        // 2、根据Class实例获取Constructor对象(构造函数)
        Constructor constructor = claz.getConstructor();

        // 3、使用Constructor对象创建对象实例
        Object obj = constructor.newInstance();     // Person obj = (Person) constructor.newInstance();

        // 4、根据Class实例获取方法的Method对象
        Method setNameMethod = claz.getMethod("setName", String.class);

        // 5、Method对象利用invoke方法调用方法
        setNameMethod.invoke(obj, "小明");

        // 6、根据Class实例获取方法的Method对象
        Method toString = claz.getMethod("toString");
        System.out.println( toString.invoke(obj)  + " ,好帅!");
    }
}
```

使用反射机制实现

```
//常规编程
Person p = new Person();
p.setName("小明");
System.out.println(p + " ,好帅!");
```

获得的是默认构造函数

setName()方法的参数类型

相当于obj.setName("小明")

运行结果

```
name=小明, age=0 ,好帅!
```

【返回】

13.3 反射机制API

- java.lang.Class：代表一个类
- java.lang.reflect.Constructor：类的构造函数
- java.lang.reflect.Method：类的方法
- java.lang.reflect.Field：类的成员变量(属性)

用法示例

- 获取Class对象
- 获取Constructor对象 (构造函数)
- 获取Field对象 (成员变量)
- 获取Method对象 (成员方法)

【返回】

1. 获取Class对象

- java.lang.Class 类是实现反射的关键所在
- Class实例对象是由JVM在类加载时自动创建的

方式1：通过类路径名获取（常用） 要添加ClassNotFoundException异常处理

- Class claz = Class.forName("edu.wust.examples.Person");

方法2：通过类型获取（每种类型都有class静态变量）

- Class claz = Person.class;
- Class claz = int.class;

方法3：通过对象获取（每对象都有getClass()方法）

- String str = "Hello";
- Class claz = str.getClass();

- getClass方法，有多态能力，运行时可以返回子类的类型信息
- .class是没有多态的，是静态解析的，编译时可以确定类型信息

Class对象常用方法

类型	访问方法	返回值类型	说明
包路径	getPackage()	Package 对象	获取该类的存放路径
类名称	getName()	String 对象	获取该类的名称
继承类	getSuperclass()	Class 对象	获取该类继承的类
实现接口	getInterfaces()	Class 型数组	获取该类实现的所有接口
构造方法	getConstructors()	Constructor 型数组	获取所有权限为 public 的构造方法
	getDeclaredContruectors()	Constructor 对象	获取当前对象的所有构造方法
方法	getMethods()	Methods 型数组	获取所有权限为 public 的方法
	getDeclaredMethods()	Methods 对象	获取当前对象的所有方法
成员变量	getFields()	Field 型数组	获取所有权限为 public 的成员变量
	getDeclareFileds()	Field 对象	获取当前对象的所有成员变量
内部类	getClasses()	Class 型数组	获取所有权限为 public 的内部类
	getDeclaredClasses()	Class 型数组	获取所有内部类
内部类的声明类	getDeclaringClass()	Class 对象	如果该类为内部类，则返回它的成员类，否则返回 null

示例：获取Class对象的相关信息

```
Class claz = Integer.class;
System.out.println("clz类名称(包含包名)：" + claz.getName() );
System.out.println("clz类的包信息：" + claz.getPackage() );
System.out.println("clz类名：" + claz.getSimpleName());
System.out.println("clz父类名称：" + claz.getSuperclass().getName() );
System.out.println("clz是否为基本类型：" + claz.isPrimitive() );
System.out.println("clz是否为接口：" + claz.isInterface() );
System.out.println("clz是否为数组对象：" + claz.isArray() );
```

一些判断方法

运行结果

```
clz类名称(包含包名)：java.lang.Integer
clz类的包信息：package java.lang
clz类名：Integer
clz父类名称：java.lang.Number
clz是否为基本类型：false
clz是否为接口：false
clz是否为数组对象：false
```

【返回】

2. 获取Constructor对象

要添加NoSuchMethodException异常处理

```
// 返回指定参数类型的public构造器（常用）
Constructor<T> getConstructor(类<?>... parameterTypes)

// 返回所有public类型的构造器
Constructor<?>[] getConstructors()

// 返回指定参数类型的构造器（包括：私有、受保护、默认、公有）
Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)

// 返回所有的构造器（包括：私有、受保护、默认、公有）
Constructor<?>[] getDeclaredConstructors()
```

示例

```
class Test {
    public static void main(String[] args) throws
            ClassNotFoundException, NoSuchMethodException {

        Class claz = Class.forName("edu.wust.examples.Person");
        System.out.println("返回所有public构造方法：");
        Constructor[] constructors = claz.getConstructors();
        for (Constructor constructor : constructors) {
            System.out.println(constructor);
        }

        System.out.println("返回所有构造方法(包括：私有、受保护、    }
默认、公有)：");                                                    }
        Constructor[] constructors2 = claz.getDeclaredConstructors();
        for (Constructor constructor : constructors2) {
            System.out.println(constructor);
        }

        Constructor constructor;
        System.out.println("返回默认的public构造器：");
        constructor = claz.getConstructor();
        System.out.println(constructor);

        System.out.println("返回指定参数类型的public构造器：");
        constructor = claz.getConstructor(String.class, int.class);
        System.out.println(constructor);
    }
```

运行结果

```
返回所有public构造方法：
public edu.wust.examples.Person()
public edu.wust.examples.Person(java.lang.String,int)
所有的构造方法(包括：私有、受保护、默认、公有)：
public edu.wust.examples.Person()
public edu.wust.examples.Person(java.lang.String,int)
返回默认的public构造器：
public edu.wust.examples.Person()
返回指定参数类型的public构造器：
public edu.wust.examples.Person(java.lang.String,int)
```

Constructor对象常用方法

方法	说明
isVarArgs()	判断该构造方法是否允许带可变数量的参数
getParameterTypes()	按照声明顺序以Class数组的形式获取该构造方法各个参数的类型
getExceptionTypes()	以Class数组的形式获取该构造方法可能抛出的异常类型
newInstance(Object … initargs)	通过该构造方法利用指定参数创建一个该类型的对象，如果未设置参数则表示采用默认无参的构造方法（示例见下页）
setAccessiable(boolean flag)	如果该构造方法的权限为 private，默认为不允许通过反射利用 netInstance()方法创建对象。如果先执行该方法，并将入口参数设置为 true，则允许创建对象
getModifiers()	获得可以解析出该构造方法所采用修饰符的整数

示例：通过Constructor创建类对象

另一种方法：不推荐
Class clz = Person.class;
Person person = (Person)clz.newInstance();

```
Class claz = Class.forName("edu.wust.examples.Person");
Constructor constructor = claz.getConstructor();
Person obj = (Person) constructor.newInstance();

Class claz2 = Class.forName("edu.wust.examples.Person");
Constructor constructor2 = claz2.getConstructor(String.class, int.class);
Person obj2 = (Person) constructor2.newInstance("小明",19);
```

获取默认public构造器

相当于Person obj=new Person();

获取指定的public构造器

相当于Person obj=new Person("小明",19);

不强转用：Object obj = constructor.newInstance();

【返回】

3. 获取Filed对象

要添加NoSuchFieldException异常处理

```
// 根据变量名获得对应的变量，访问权限不限；
Field getDeclaredField(String name)
// 获得类中所有属性变量
Field[] getDeclaredFields()
// 根据变量名获取public类型的属性变量
Field getField(String name)
// 获取类中所有public类型的属性变量
Field[] getFields()
```

示例

```
class Test {
    public static void main(String[] args) throws
            ClassNotFoundException, NoSuchFieldException {

        Class claz = Class.forName("edu.wust.examples.Person");
        System.out.println("返回所有属性：");
        Field[] fields = claz.getDeclaredFields();
        for (Field field : fields) {
            System.out.println(field);
        }

        System.out.println("返回所有public属性：");
        Field[] fields2 = claz.getFields();
        for (Field field : fields2) {
            System.out.println(field);
        }

        System.out.println("返回age属性：");
        Field field2 = claz.getDeclaredField("age");
        System.out.println(field2);

        System.out.println("返回public的name属性：");
        Field field1 = claz.getField("name");
        System.out.println(field1);
    }
}
```

运行报错，name不是public

运行结果

```
返回所有属性：
private java.lang.String edu.wust.examples.Person.name
private int edu.wust.examples.Person.age
返回所有public属性：
返回age属性：
private int edu.wust.examples.Person.age
返回public的name属性：
Exception in thread "main" java.lang.NoSuchFieldException Create breakpoint : name
```

Filed对象常用方法

方法	说明
getName()	获得该成员变量的名称
getType()	获取表示该成员变量的 Class 对象
get(Object obj)	获得指定对象 obj 中成员变量的值，返回值为 Object 类型
set(Object obj, Object value)	将指定对象 obj 中成员变量的值设置为 value
getInt(0bject obj)	获得指定对象 obj 中成员类型为 int 的成员变量的值
setInt(0bject obj, int i)	将指定对象 obj 中成员变量的值设置为 i
setFloat(Object obj, float f)	将指定对象 obj 中成员变量的值设置为 f
getBoolean(Object obj)	获得指定对象 obj 中成员类型为 boolean 的成员变量的值
setBoolean(Object obj, boolean b)	将指定对象 obj 中成员变量的值设置为 b
getFloat(Object obj)	获得指定对象 obj 中成员类型为 float 的成员变量的值
setAccessible(boolean flag)	此方法可以设置是否忽略权限直接访问private成员变量
getModifiers()	获得可以解析出该方法所采用修饰符的整数

示例：修改私有成员变量的值

```
class Test {
    public static void main(String[] args) throws ClassNotFoundException, NoSuchFieldException, NoSuchMethodException,
InvocationTargetException, InstantiationException, IllegalAccessException {


        Class claz = Class.forName("edu.wust.examples.Person");
        Constructor constructor = claz.getConstructor(String.class, int.class);
        Person obj = (Person) constructor.newInstance("小明",19);
        //获取age字段
        Field field = claz.getDeclaredField("age");
        System.out.println("成员变量名称为：" + field.getName() );
        System.out.println("成员变量类型为：" + field.getType() );
        System.out.println("成员变量访问权限是否是private：" + Modifier.isPrivate( field.getModifiers() ));
        field.setAccessible(true);      //忽略权限,直接访问private成员变量
        System.out.println("修改前成员的值为：" + field.get(obj) );
        field.set(obj,20);     //修改age字段为20
        System.out.println("修改后成员的值为：" + field.get(obj) );
    }
}
```

Modifier用法

【返回】

4. 获取Method对象

要添加NoSuchMethodException异常处理

```
// 获取"名称是name，参数是parameterTypes"的public方法(包括从基类继承的、从接口实现的) -- 常用
public Method  getMethod(String name, Class[] parameterTypes)
// 获取全部的public方法(包括从基类继承的、从接口实现的)
public Method[]  getMethods()
// 获取"名称是name，参数是parameterTypes"，并且是类自身声明的方法(包含public、protected、private和default方法)
public Method  getDeclaredMethod(String name, Class[] parameterTypes)
// 获取全部的类自身声明的方法(包含public、protected、private和default方法)
public Method[]  getDeclaredMethods()
```

示例

```
class Test {
    public static void main(String[] args) throws
            ClassNotFoundException, NoSuchFieldException {

        Class claz = Class.forName("edu.wust.examples.Person");
        System.out.println("返回所有的public方法：");
        Method[] methods = claz.getMethods();
        for (Method method : methods) {
            System.out.println(method);
        }

        System.out.println("返回自身声明的所有方法：");
        Method[] methods2 = claz.getDeclaredMethods();
        for (Method method : methods2) {
            System.out.println(method);
        }

        System.out.println("返回public方法：");
        Method method = claz.getMethod("setName",String.class);
        System.out.println(method);

        System.out.println("返回自身声明的方法：");
        method = claz.getDeclaredMethod("getAge");
        System.out.println(method);
    }
}
```

运行结果见下页

运行结果

```
返回所有的public方法:
public java.lang.String edu.wust.examples.Person.getName()
public java.lang.String edu.wust.examples.Person.toString()
public void edu.wust.examples.Person.setName(java.lang.String)
public int edu.wust.examples.Person.getAge()
public void edu.wust.examples.Person.setAge(int)
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
public boolean java.lang.Object.equals(java.lang.Object)
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
返回自身声明的所有方法:
public java.lang.String edu.wust.examples.Person.getName()
public java.lang.String edu.wust.examples.Person.toString()
public void edu.wust.examples.Person.setName(java.lang.String)
public int edu.wust.examples.Person.getAge()
public void edu.wust.examples.Person.setAge(int)
返回public方法:
public void edu.wust.examples.Person.setName(java.lang.String)
返回自身声明的方法:
public int edu.wust.examples.Person.getAge()
```

Method对象常用方法

方法	说明
getName()	获取该方法的名称
getParameterType()	按照声明顺序以 Class 数组的形式返回该方法各个参数的类型
getReturnType()	以 Class 对象的形式获得该方法的返回值类型
getExceptionTypes()	以 Class 数组的形式获得该方法可能抛出的异常类型
invoke(Object obj,Object...args)	利用args参数执行指定对象obj中的该方法，返回值为Object类型
isVarArgs()	查看该方法是否允许带有可变数量的参数，如果允许返回 true，否则返回 false
getModifiers()	获得可以解析出该方法所采用修饰符的整数

示例：执行对象的方法

```
class Test {
    public static void main(String[] args) throws ClassNotFoundException, NoSuchMethodException, InvocationTargetException,
InstantiationException, IllegalAccessException {
        Class claz = Class.forName("edu.wust.examples.Person");
        Constructor constructor = claz.getConstructor(String.class, int.class);
        Person obj = (Person) constructor.newInstance("小明", 19);

        //调用getAge方法获取age值
        Method method = claz.getMethod("getAge");
        System.out.println("修改前age值为：" + method.invoke(obj) );

        //调用setAge方法修改age值
        method = claz.getMethod("setAge", int.class);
        method.invoke(obj, 20);

        //调用toString方法
        method = claz.getMethod("toString");
        System.out.println("修改后的对象为：" + method.invoke(obj) );
    }
}
```

运行结果

```
修改前age值为：19
修改后的对象为：name=小明, age=20
```

【完】