Java 反射机制

Java 反射机制在程序**运行时**，对于任意一个类，都能够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一个方法和属性。这种 **动态的获取信息** 以及 **动态调用对象的方法** 的功能称为 **java 的反射机制**。

反射机制很重要的一点就是“运行时”，其使得我们可以在程序运行时加载、探索以及使用编译期间完全未知的 `.class` 文件。换句话说，Java 程序可以加载一个运行时才得知名称的 `.class` 文件，然后获悉其完整构造，并生成其对象实体、或对其 fields（变量）设值、或调用其 methods（方法）。

使用反射获取类的信息

为使得测试结果更加明显，我首先定义了一个 `FatherClass` 类（默认继承自 `Object` 类），然后定义一个继承自 `FatherClass` 类的 `SonClass` 类，如下所示。可以看到测试类中变量以及方法的访问权限不是很规范，是为了更明显得查看测试结果而故意设置的，实际项目中不提倡这么写。

FatherClass.java

```
public class FatherClass {
    public String mFatherName;
    public int mFatherAge;

public void printFatherMsg(){}

}
```

SonClass.java

```
public class SonClass extends FatherClass{

private String mSonName;
protected int mSonAge;
public String mSonBirthday;

public void printSonMsg(){
    System.out.println("Son Msg - name : "

   + mSonName + "; age : " + mSonAge);
     }
```

```
private void setSonName(String name){
    mSonName = name;
}

private void setSonAge(int age){
    mSonAge = age;
}

private int getSonAge(){
    return mSonAge;
}

private String getSonName(){
    return mSonName;
}

}
```

1. 获取类的所有变量信息

```
/**
 * 通过反射获取类的所有变量
 */
private static void printFields(){
    //1.获取并输出类的名称
    Class mClass = SonClass.class;
    System.out.println("类的名称: " + mClass.getName());

    //2.1 获取所有 public 访问权限的变量
    // 包括本类声明的和从父类继承的
    Field[] fields = mClass.getFields();

    //2.2 获取所有本类声明的变量（不问访问权限）
    //Field[] fields = mClass.getDeclaredFields();

    //3. 遍历变量并输出变量信息
    for (Field field :
            fields) {
        //获取访问权限并输出
        int modifiers = field.getModifiers();
        System.out.print(Modifier.toString(modifiers) + " ");
        //输出变量的类型及变量名
        System.out.println(field.getType().getName()
                + " " + field.getName());
    }
}
```

输出:

```
类的名称: SonClass
public java.lang.String mSonBirthday
public java.lang.String mFatherName
public int mFatherAge
```

需要注意的是注释中 2.1 的 `getFields()` 与 2.2的 `getDeclaredFields()` 之间的区别，下面分别看一下两种情况下的输出。看之前强调一下： `SonClass` extends `FatherClass` extends `Object` ：

- 调用 `getFields()` 方法，输出 `SonClass` 类以及其所继承的父类(包括 `FatherClass` 和 `Object`) 的 `public` 方法。注： `Object` 类中没有成员变量，所以没有输出。

```
类的名称: obj.SonClass
public java.lang.String mSonBirthday
public java.lang.String mFatherName
public int mFatherAge
```

- 调用 `getDeclaredFields()` ， 输出 `SonClass` 类(不包含父类)的所有成员变量，不问访问权限。

```
类的名称: obj.SonClass
private java.lang.String mSonName
protected int mSonAge
public java.lang.String mSonBirthday
```

2. 获取类的所有方法信息

```
/**
 * 通过反射获取类的所有方法
 */
private static void printMethods(){
    //1.获取并输出类的名称
    Class mClass = SonClass.class;
    System.out.println("类的名称: " + mClass.getName());

    //2.1 获取所有 public 访问权限的方法
    //包括自己声明和从父类继承的
    Method[] mMethods = mClass.getMethods();

    //2.2 获取所有本类的的方法（不问访问权限）
    //Method[] mMethods = mClass.getDeclaredMethods();

    //3.遍历所有方法
    for (Method method :
            mMethods) {
        //获取并输出方法的访问权限（Modifiers: 修饰符）
        int modifiers = method.getModifiers();
        System.out.print(Modifier.toString(modifiers) + " ");
        //获取并输出方法的返回值类型
        Class returnType = method.getReturnType();
        System.out.print(returnType.getName() + " "
                + method.getName() + "( ");
        //获取并输出方法的所有参数
        Parameter[] parameters = method.getParameters();
        for (Parameter parameter:
             parameters) {
            System.out.print(parameter.getType().getName()
                    + " " + parameter.getName() + ",");
```

```
        }
        //获取并输出方法抛出的异常
        Class[] exceptionTypes = method.getExceptionTypes();
        if (exceptionTypes.length == 0){
            System.out.println(" )");
        }
        else {
            for (Class c : exceptionTypes) {
                System.out.println(" ) throws "
                        + c.getName());
            }
        }
    }
}
```

同获取变量信息一样，需要注意注释中 2.1 与 2.2 的区别，下面看一下打印输出：

- 调用 `getMethods()` 方法 获取 `SonClass` 类所有 `public` 访问权限的方法，包括从父类继承的。打印信息中，`printSonMsg()` 方法来自 `SonClass` 类， `printFatherMsg()` 来自 `FatherClass` 类，其余方法来自 Object 类。

```
类的名称: obj.SonClass
public void printSonMsg(  )
public void printFatherMsg(  )
public final void wait(  ) throws java.lang.InterruptedException
public final void wait( long arg0,int arg1, ) throws
java.lang.InterruptedException
public final native void wait( long arg0, ) throws
java.lang.InterruptedException
public boolean equals( java.lang.Object arg0, )
public java.lang.String toString(  )
public native int hashCode(  )
public final native java.lang.Class getClass(  )
public final native void notify(  )
public final native void notifyAll(  )
复制代码
```

- 调用 `getDeclaredMethods()` 方法

  打印信息中，输出的都是 `SonClass` 类(不包含父类)的方法，不问访问权限。

```
类的名称: obj.SonClass
private int getSonAge(  )
private void setSonAge( int arg0, )
public void printSonMsg(  )
private void setSonName( java.lang.String arg0, )
private java.lang.String getSonName(  )
```

访问或操作类的私有变量和方法

在上面，我们成功获取了类的变量和方法信息，验证了在运行时 **动态的获取信息** 的观点。那么，仅仅是获取信息吗？我们接着往后看。

都知道，对象是无法访问或操作类的私有变量和方法的，但是，通过反射，我们就可以做到。没错，反射可以做到！下面，让我们一起探讨如何利用反射访问 **类对象的私有方法** 以及修改 **私有变量或常量**。

老规矩，先上测试类。

注：

1. 请注意看测试类中变量和方法的修饰符（访问权限）；
2. 测试类仅供测试，不提倡实际开发时这么写：)

TestClass.java

```
public class TestClass {

    private String MSG = "Original";

    private void privateMethod(String head , int tail){
        System.out.print(head + tail);
    }

    public String getMsg(){
        return MSG;
    }
}
```

3.1 访问私有方法

以访问 `TestClass` 类中的私有方法 `privateMethod(...)` 为例，方法加参数是为了考虑最全的情况，很贴心有木有？先贴代码，看注释，最后我会重点解释部分代码。

```
/**
 * 访问对象的私有方法
 * 为简洁代码，在方法上抛出总的异常，实际开发别这样
 */
private static void getPrivateMethod() throws Exception{
    //1. 获取 Class 类实例
    TestClass testClass = new TestClass();
    Class mClass = testClass.getClass();

    //2. 获取私有方法
    //第一个参数为要获取的私有方法的名称
    //第二个为要获取方法的参数的类型，参数为 Class...，没有参数就是null
    //方法参数也可这么写 ：new Class[]{String.class , int.class}
    Method privateMethod =
            mClass.getDeclaredMethod("privateMethod", String.class, int.class);

    //3. 开始操作方法
    if (privateMethod != null) {
        //获取私有方法的访问权
        //只是获取访问权，并不是修改实际权限
        privateMethod.setAccessible(true);

        //使用 invoke 反射调用私有方法
        //privateMethod 是获取到的私有方法
        //testClass 要操作的对象
        //后面两个参数传实参
        privateMethod.invoke(testClass, "Java Reflect ", 666);
    }
}
```

需要注意的是，第3步中的 `setAccessible(true)` 方法，是获取私有方法的访问权限，如果不加会报异常 **IllegalAccessException**，因为当前方法访问权限是“private”的，如下：

```
java.lang.IllegalAccessException: Class MainClass can not access a member of
class obj.TestClass with modifiers "private"
复制代码
```

正常运行后，打印如下，**调用私有方法**成功：

```
Java Reflect 666
复制代码
```

3.2 修改私有变量

以修改 `TestClass` 类中的私有变量 `MSG` 为例，其初始值为 "Original" ，我们要修改为 "Modified"。老规矩，先上代码看注释。

```
/**
 * 修改对象私有变量的值
 * 为简洁代码，在方法上抛出总的异常
 */
private static void modifyPrivateFiled() throws Exception {
    //1. 获取 Class 类实例
    TestClass testClass = new TestClass();
    Class mClass = testClass.getClass();

    //2. 获取私有变量
    Field privateField = mClass.getDeclaredField("MSG");

    //3. 操作私有变量
    if (privateField != null) {
        //获取私有变量的访问权
        privateField.setAccessible(true);

        //修改私有变量，并输出以测试
        System.out.println("Before Modify: MSG = " + testClass.getMsg());

        //调用 set(object , value) 修改变量的值
        //privateField 是获取到的私有变量
        //testClass 要操作的对象
        //"Modified" 为要修改成的值
        privateField.set(testClass, "Modified");
        System.out.println("After Modify: MSG = " + testClass.getMsg());
    }
}
复制代码
```

此处代码和访问私有方法的逻辑差不多，就不再赘述，从输出信息看出 **修改私有变量** 成功：

```
Before Modify: MSG = Original
After Modify: MSG = Modified
```