利用反射可以做出一个对象具备的所有操作行为，这一切的操作核心是基于 Object类。

对于反射基本概念还有疑惑可以参考：

> Java 反射1-反射概念，Class类概念，反射创建对象实例化

# 1. 反射取得类信息

通过 Object类 的方法，取得类的信息。

## 1.1 取得类的包名称

```
// 取得当前Class对象包信息
public Package getPackage()
// 取得Class对象所在包名
public String getPackageName()
```

看下面的例子:

```
class Person {
}

public class Test {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        Class<?> cls = Person.class;
        System.out.println("getPackage(): "+cls.getPackage());
        System.out.println("getPackageName(): "+cls.getPackageName());
    }
}
```

运行结果:

## 1.2 取得父类信息

```
// 取得Class对象父类对象
public native Class<? super T> getSuperclass();
```

我们看下面的例子:

```
class Person {
}

class Student extends Person {
```

```
}

public class Test {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        Class<?> cls = Student.class;
        // 拿到Student父类对象并打印
        System.out.println(cls.getSuperclass().getName());
    }
}
```

运行结果：

## 1.3 取得所有父接口

```
// 获取一个Class对象所有的父接口信息
 public Class<?>[] getInterfaces()
```

```
interface A {
}

interface B {
}

class Demo implements A,B {
}


public class Test {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        Class<?> cls = Demo.class;
        // 因为接口可以多继承，所以在接收时应该时用一个对象数组来接收
        Class<?>[] classes = cls.getInterfaces();
        for(Class tmp : classes) {
            System.out.println(tmp);
        }
    }
}
```

运行结果：

Class<?> cls = Demo.class; Class<?>[] class = cls.getInterfaces();

# 2. 反射调用构造方法

Constructor :描述类中构造方法的类。

# 2.1 反射取得构造方法

```
// 1. 取得本类中所有非private构造方法
public Constructor<T> getConstructor(Class<?>... parameterTypes)
        throws NoSuchMethodException, SecurityException

// 2. 取得本类中指定的非private构造方法
public Constructor<?>[] getConstructors() throws SecurityException

// 3. 取得本类中指定的构造方法，与权限无关
public Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)
        throws NoSuchMethodException, SecurityException

// 4. 取得本类中所有的构造方法，和权限无关
public Constructor<?>[] getDeclaredConstructors() throws SecurityException
```

看下面的例子:

```
import java.lang.reflect.Constructor;

class Person {
    private Person() {
    }

    public Person(int a) {
        System.out.println("int a");
    }

    public Person(String b) {
        System.out.println("String b");
    }

    private Person(boolean bool) {
        System.out.println("boolean bool");
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;

        // 取得本类中指定的非private构造方法
        Constructor<?> constructor = cls.getConstructor(int.class);
        System.out.println("getConstructor():------------");
        System.out.println(constructor);
        for(Constructor tmp : constructors) {

        // 取得本类中所有非private构造方法
        Constructor<?>[] constructors = cls.getConstructors();
        System.out.println("getConstructors():------------");
            // 直接打印调用了 Constructor类 的 toString方法，取得了构造方法的完整信息(权限，参数列表等)
            // 如果再使用了 getName() 方法，只会返回构造方法的 包名.类名
            System.out.println(tmp);
        }

        // 取得本类中指定的构造方法，与权限无关
```

```
        Constructor<?> constructor2 = cls.getDeclaredConstructor(boolean.class);
        System.out.println("getDeclaredConstructor():------------");
        System.out.println(constructor2);

        // 取得本类中所有的构造方法，和权限无关
        Constructor<?>[] constructors2 = cls.getDeclaredConstructors();
        System.out.println("getDeclaredConstructors():------------");
        for(Constructor tmp : constructors2) {
            System.out.println(tmp);
        }
    }
}
```

int.class int基本类型的类信息。

## 2.2 反射调用构造方法

通过反射 newInstance()实例化类对象时，调用的是类的无参构造，如果该类没有无参构造，就会报错。

看下面的例子：

```
import java.lang.reflect.Constructor;

class Person {
    // 无参构造私有化
    private Person() {
    }

    public Person(int n) {
        System.out.println("int n");
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;
        Object object = cls.newInstance();
        System.out.println(object);
    }
}
```

运行结果：

会抛出 java.lang.IllegalAccessException 异常。

这时就只能通过getConstructor先取得该类其他构造，然后通过newInstance(typename)创建对象。

```
// newInstance()定义
public T newInstance()
        throws InstantiationException, IllegalAccessException
```

使用Construct类的newInstance()方法来进行实例化：

```
import java.lang.reflect.Constructor;

class Person {
    private Person() {
    }

    public Person(int n) {
        System.out.println("int n");
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;
        Constructor<?> constructor = cls.getConstructor(int.class);
        // 调用构造
        constructor.newInstance(10);
    }
}
```

运行结果：

由这个例子我们也可以得出一个结论：以后写简单Java类时要写上无参构造。

# 3. 反射调用普通方法

Method类：描述类中普通方法的类。

## 3.1 反射取得普通方法

和构造方法一样，反射取得普通方法分为如下四种：

```
// 1. 反射取得本类及父类中的所有非private普通方法
public Method[] getMethods() throws SecurityException

// 2. 反射取得本类及父类中指定普通方法(用方法名称和参数Class对象指定)
public Method getMethod(String name, Class<?>... parameterTypes)
        throws NoSuchMethodException, SecurityException

// 3. 反射取得本类中的所有普通方法，与权限无关
public Method[] getDeclaredMethods() throws SecurityException

// 4. 反射取得本类中指定普通方法,与权限无关
public Method getDeclaredMethod(String name, Class<?>... parameterTypes)
        throws NoSuchMethodException, SecurityException
```

看下面的例子：

```
import java.lang.reflect.Method;

class Person {
    public void printA(int A) {
    }

    public void printB(double B) {
    }

    private void printC(String C) {
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;
        // 反射取得本类及父类中的所有非private普通方法
        System.out.println("getMethods():--------------------");
        Method[] methods = cls.getMethods();
        for(Method tmp : methods) {
            System.out.println(tmp);
        }

        // 反射取得本类及父类中指定普通方法(用方法名称指定)
        System.out.println("getMethod():--------------------");
        Method method = cls.getMethod("printA", int.class);
        System.out.println(method);

        // 反射取得本类中的所有普通方法，与权限无关
        System.out.println("getDeclaredMethods():--------------------");
        Method[] methods2 = cls.getDeclaredMethods();
        for(Method tmp : methods2) {
            System.out.println(tmp);
        }

        // 反射取得本类中指定普通方法,与权限无关
        System.out.println("getDeclaredMethod():--------------------");
        Method method2 = cls.getDeclaredMethod("printC", String.class);

    }
}
```

运行结果：

# 3.1 反射调用普通方法

使用 Methodt 类的 invoke()方法 进行调用类中普通方法。

```
public Object invoke(Object obj, Object... args)
        throws IllegalAccessException, IllegalArgumentException,
           InvocationTargetException
```

看下面的例子：

```
import java.lang.reflect.Method;

class Person {
    public String name;
    public int 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: "+this.name+", age: "+this.age+"]";
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;
        // 取得set方法
        Method setNameMethod = cls.getDeclaredMethod("setName", String.class);
        Method setAgeMethod = cls.getDeclaredMethod("setAge", int.class);
        // invoke()要传入一个对象的实例化
        Object object = cls.newInstance();
        setNameMethod.invoke(object, "张三");
        setAgeMethod.invoke(object, 10);
        System.out.println(object);
    }
}
```

运行结果：

通过反射调用普通方法的好处是：

不再局限于某一具体类型的对象，而是可以通过Object类型进行所有类的方法调用。

# 4. 反射调用类中属性

Field类： 描述类中属性的类。

# 4.1 反射取得类中属性

同样的，它也有四种典型方法：

```
// 取得本类和父类中的所有非private属性，包括final，static
public Field[] getFields() throws SecurityException

// 取得本类和父类中指定的非private属性，包括final， static
public Field getField(String name)
        throws NoSuchFieldException, SecurityException

// 取得本类中所有属性，与权限无关
public Field[] getDeclaredFields() throws SecurityException

// 取得本类中指定的属性，与权限无关
public Field getDeclaredField(String name)
        throws NoSuchFieldException, SecurityException
```

看下面的例子：

```
import java.lang.reflect.Field;

class Person {
    public String name;
    public int age;
    private String gender;
}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;
        // 取得本类和父类中的所有非private属性，包括final，static
        Field[] fields = cls.getFields();
        System.out.println("getFields():----------------");
        for(Field tmp : fields) {
            System.out.println(tmp);
        }

        // 取得本类和父类中指定的非private属性，包括final， static
        Field field = cls.getField("name");
        System.out.println("getField():----------------");
        System.out.println(field);

        // 取得本类中所有属性，与权限无关
        Field[] fields2 = cls.getDeclaredFields();
        System.out.println("getDeclaredFields():----------------");
        for(Field tmp : fields2) {
            System.out.println(tmp);
        }

        // 取得本类中指定的属性，与权限无关
        Field field2 = cls.getDeclaredField("gender");
        System.out.println("getDeclaredField():----------------");
        System.out.println(field2);

    }
}
```

运行结果：

# 4.2 反射修改非private属性

和上一节的 invoke 方法类似，Field类提供了两个方法用来操作类中属性。

```
// 设置属性内容(非private)
// 与 invoke 用法类似，都要传入一个 Object 对象实例化
public void set(Object obj, Object value)
        throws IllegalArgumentException, IllegalAccessException

// 取得属性内容(非private)
public Object get(Object obj)
        throws IllegalArgumentException, IllegalAccessException
```

看一个例子：

```
import java.lang.reflect.Field;

class Person {
    public String name;
    public int age;
    private String gender;
    @Override
    public String toString() {
        return "[name: "+this.name+", age: "+this.age+", gender: "+this.gender+"]";
    }


}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;

        Field nameField = cls.getDeclaredField("name");
        Field ageField = cls.getDeclaredField("age");
        Object obj = cls.newInstance();
        nameField.set(obj, "张三");
        ageField.set(obj, 10);
        System.out.println("name: "+nameField.get(obj)+", age:"+ageField.get(obj));
        System.out.println(obj);
    }
}
```

运行结果：

# 4.3 反射破坏属性封装性

反射最强大的地方在于它可以修改对象类中private属性。

通过Field类的setAccessible()方法破坏掉private属性的封装性，进而修改属性。

```
// flag = true, 取消属性封装性
public void setAccessible(boolean flag)
```

看下面的例子：

```
import java.lang.reflect.Field;

class Person {
    public String name;
    public int age;
    private String gender;
    @Override
    public String toString() {
        return "[name: "+this.name+", age: "+this.age+", gender: "+this.gender+"]";
    }


}

public class Test {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Person.class;

        Object obj = cls.newInstance();
        Field genderField = cls.getDeclaredField("gender");
        genderField.setAccessible(true);
        genderField.set(obj, "男");
        System.out.println("gender: "+genderField.get(obj));
        System.out.println(obj);
    }
}
```

运行结果：

该方法实际上是 AccessibleObject类 提供的方法：

而Field类继承了它所以可以使用，同时Construct,Method类也继承了AccessibleObject类(继承关系如下图)，所以它们也有这个方法。

注意：破坏封装性只在本次JVM中进程中生效，即本质上没有修改属性的权限。

# 5. 总结