一、Java反射基础

其实，在Java命令执行和Java数据库操作章节，我们就已经简单接触到Java反射了。

一是 `Java命令执行` 章节中，使用 `java.lang.ProcessImpl` 的话需要配合反射机制来执行命令。

```
Class clazz = Class.forName("java.lang.ProcessImpl");
Method method = clazz.getDeclaredMethod("start", String[].class,
Map.class, String.class, ProcessBuilder.Redirect[].class,
boolean.class);
method.setAccessible(true);
Process process = (Process) method.invoke(null, cmds, null, ".", null,
true);
```

二是 `Java数据库操作` 章节中，使用反射机制来注册Mysql驱动。

```
// 注册驱动
Class.forName("com.mysql.cj.jdbc.Driver");
```

本节我们就学习下Java反射机制的基础知识吧。

1、什么是反射?

Java的反射（reflection）机制是指在程序运行中，可以构造任意一个类的对象，可以了解任意一个对象所属的类，可以了解任意一个类的成员变量和方法，可以调用任意一个对象的属性和方法。 这种动态获取程序信息以及动态调用对象的功能称为Java语言的反射机制 。

反射机制允许我们在Java程序运行时检查，操作或者说获取任何类、接口、构造函数、方法和字段。还可以动态创建Java类实例、调用任意的类方法、修改任意的类成员变量值等操作。

在Java代码审计中学习反射机制，我们目的是可以利用反射机制操作目标方法执行系统命令。比如我们想要反射调用 `java.lang.runtime` 去执行系统命令。这个下面会讲到。

下面，通过代码案例来学习反射API，进一步理解反射机制。

2、创建练习项目工程

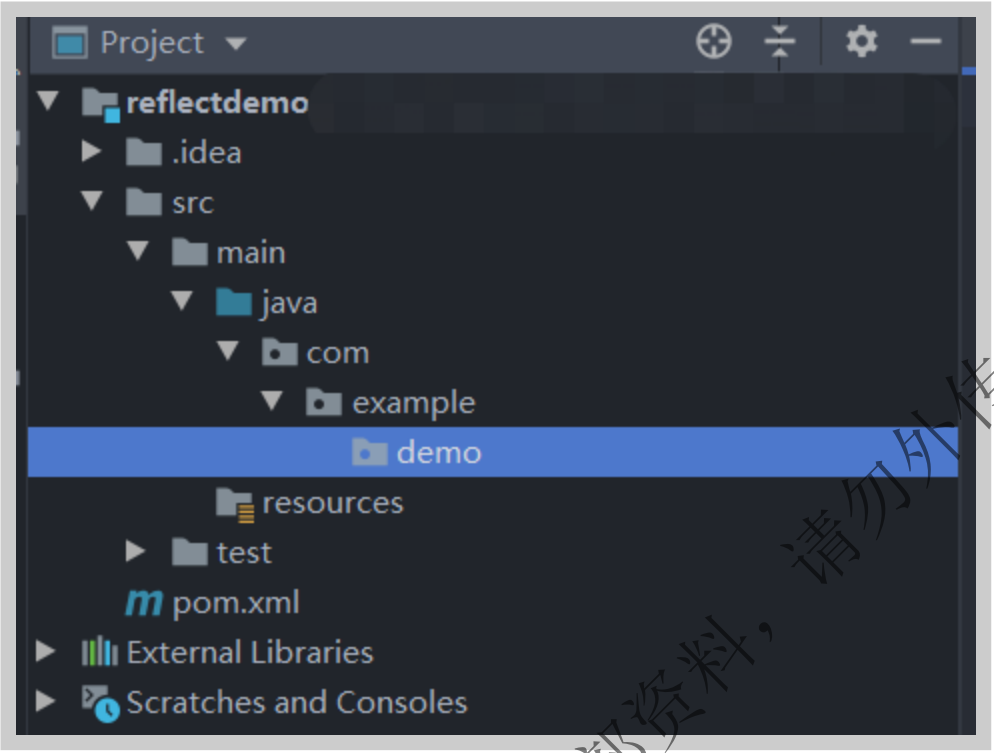
老规矩，先创建一个名为 `reflectdemo` 的项目工程，用于下面示例代码的练习。

①、打开IDEA，点击 `Create New Project` ，创建新的工程。

②、左侧选择Maven，配置默认即可，不选择任何模板，点击Next。

③、起个项目名称为 `reflectdemo` ，其他默认即可，点击Finish。

④、在Java目录下创建名为 `com.exampl.demo` 的包，并在demo包下再创建一个名为 `entity` 的包，最终目录结构如下图所示：



3、获取Class对象

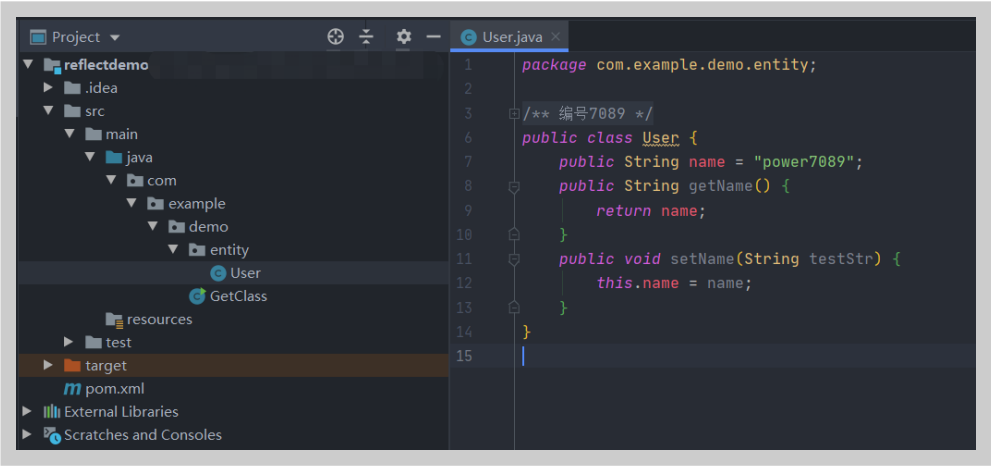
获取Class对象的方式有下面几种：

- 根据类名：类名.class
- 根据对象：对象.getClass()
- 根据全限定类名：Class.forName(全路径类名)
- 通过类加载器获得class对象：ClassLoader.getSystemClassLoader().loadClass("com.example.xxx");

举个简单的例子。

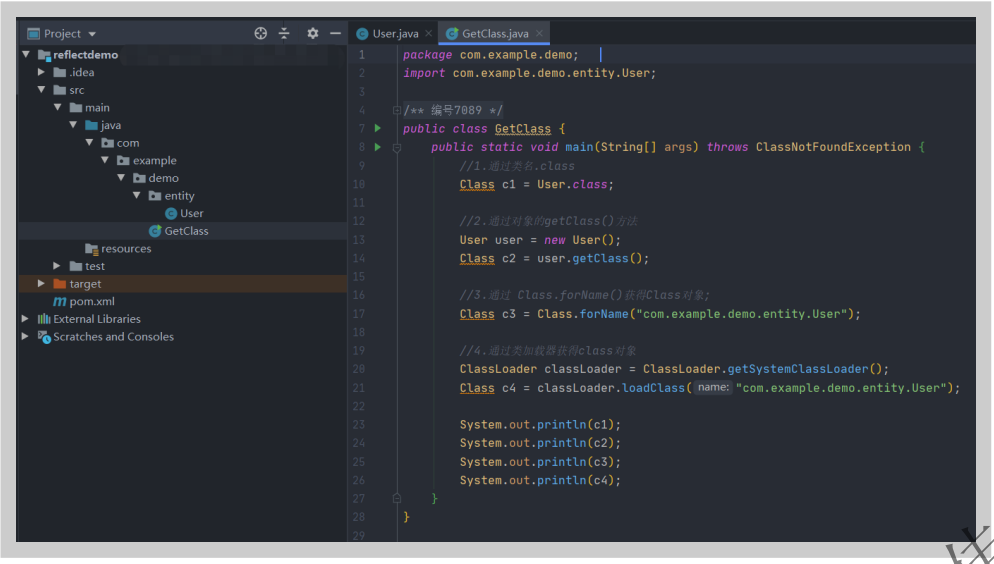
①、我们在 `com.example.demo.entity` 下创建个 `User` 类，代码如下：

```
public class User {
    public String name = "power7089";
    public String getName() {
        return name;
    }
    public void setName(String testStr) {
        this.name = name;
    }
}
```



②、我们再 `com.example.demo` 先创建一个名为 `GetClass` 的类，用于演示获取User Class对象几种方式，代码如下：

```
package com.example.demo;
import com.example.demo.entity.User;

public class GetClass {
    public static void main(String[] args) throws
ClassNotFoundException {
        //1.通过类名.class
        Class c1 = User.class;

        //2.通过对象的getClass()方法
        User user = new User();
        Class c2 = user.getClass();

        //3.通过 Class.forName()获得Class对象;
        Class c3 = Class.forName("com.example.demo.entity.User");

        //4.通过类加载器获得class对象
        ClassLoader classLoader = ClassLoader.getSystemClassLoader();
        Class c4 =
classLoader.loadClass("com.example.demo.entity.User");

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
        System.out.println(c4);
    }
}
```



动手操作调试，观察运行结果，并加以思考。

那他们几个有什么需要注意的呢？

- 类名.class：需要导入类的包。
- 对象.getClass()：初始化对象后，其实不需要再使用反射了。
- Class.forName(全路径类名)：需要知道类的完整全路径，这是我们常使用的方法。
- 通过类加载器获得class对象：
  ClassLoader.getSystemClassLoader().loadClass("com.example.xxx");

Class.forName()获取class对象方法是常用的一种方式，下面所有示例代码我们都使用 `Class.forName()` 这个方法来获取Class对象。

在获取到目标Class对象后，我们可以做的事就多了，下面我们通过示例代码进一步演示。

4、Java反射API

Java提供了一套反射API，该API由 `Class` 类与 `java.lang.reflect` 类库组成。

该类库包含了 `Field` 、 `Method` 、 `Constructor` 等类。

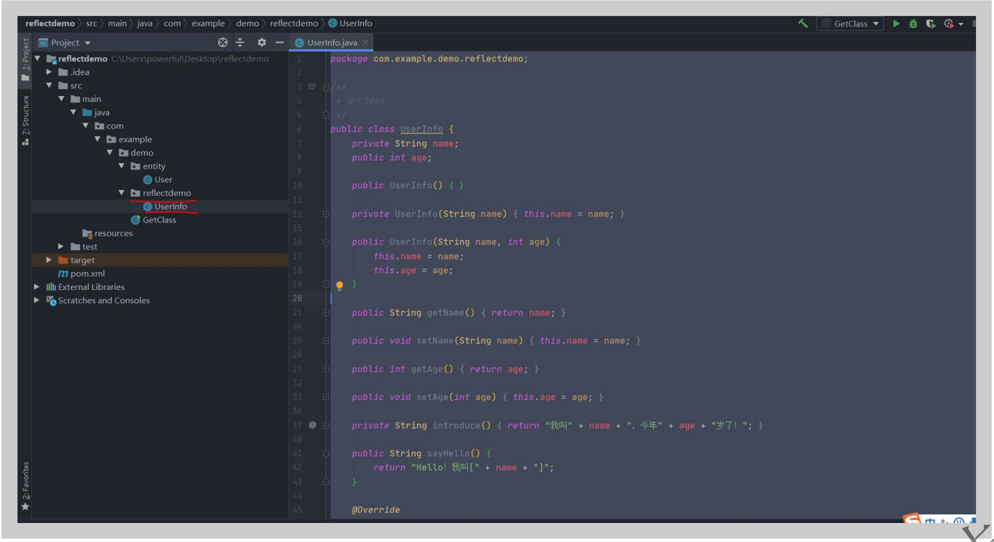
java.lang.reflect官方文档：
`https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html`

（这部分的官方文档我都是给的java8的，大家可以将路径中的数字8改为7，9，10等，这几个版本会有不同的地方大家可自行比对学习）

⚠在进行下面练习前，首先我们需要在 `com.example.demo` 下新建一个名为 `reflectdemo` 的包，并新建一个名为 `UserInfo` 的Java Class，并键入一下代码，最终如下图所示：

```
package com.example.demo.reflectdemo;
```

```java
public class UserInfo {
    private String name;
    public int age;

    public UserInfo() { }

    private UserInfo(String name) {
        this.name = name;
    }

    public UserInfo(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;
    }

    private String introduce(){
        return "我叫" + name + ", 今年" + age + "岁了! ";
    }

    public String sayHello() {
        return "Hello! 我叫[" + name + "]";
    }

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



4.1、java.lang.Class

用来描述类的内部信息， `Class` 的实例可以获取类的包、注解、修饰符、名称、超类、接口等。

官方文档： `https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html` 。

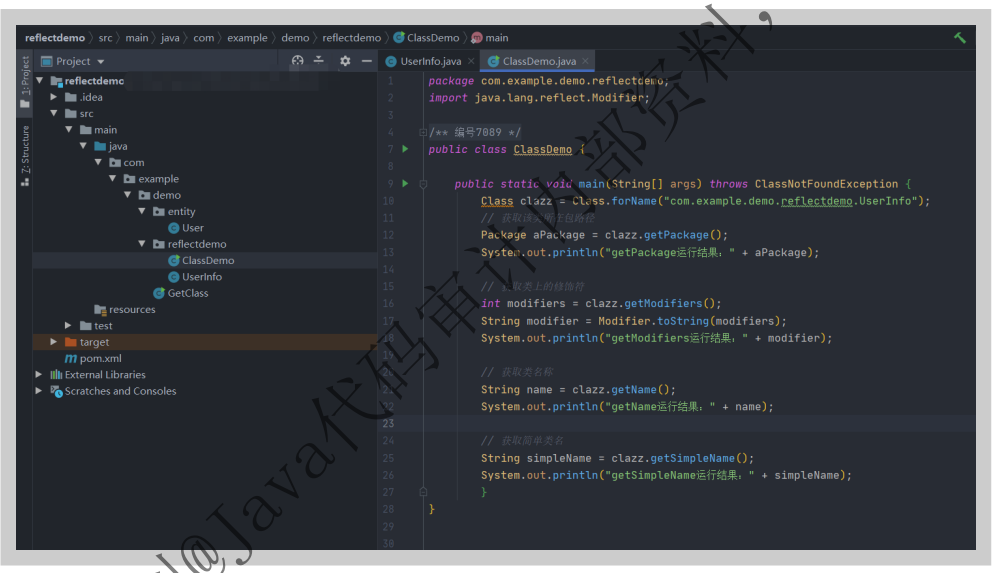
方法名	释义
getPackage()	获取该类的包
getDeclaredAnnotations()	获取该类上所有注解
getModifiers()	获取该类上的修饰符
getName()	获取类名称
getSimpleName()	获取简单类名称
getGenericSuperclass()	获取直属超类
getGenericInterfaces()	获取直属实现的接口
newInstance()	根据构造函数创建一个实例
更多方法可查看官方文档......	

4.1.1、示例代码

①、在创建 `com.example.demo.reflectdemo` 下创建一个名为 `ClassDemo` 的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.demo.reflectdemo;
import java.lang.reflect.Modifier;

/**
 * 编号7089
 */
public class ClassDemo {
```

```
    public static void main(String[] args) throws
ClassNotFoundException {
        Class clazz =
Class.forName("com.example.demo.reflectdemo.UserInfo");
        // 获取该类所在包路径
        Package aPackage = clazz.getPackage();
        System.out.println("getPackage运行结果：" + aPackage);

        // 获取类上的修饰符
        int modifiers = clazz.getModifiers();
        String modifier = Modifier.toString(modifiers);
        System.out.println("getModifiers运行结果：" + modifier);

        // 获取类名称
        String name = clazz.getName();
        System.out.println("getName运行结果：" + name);
        // 获取简单类名
        String simpleName = clazz.getSimpleName();
        System.out.println("getSimpleName运行结果：" + simpleName);
        }
}
```



运行结果如下图所示：

```
getPackage运行结果：package com.example.demo.reflectdemo
getModifiers运行结果：public
getName运行结果：com.example.demo.reflectdemo.UserInfo
getSimpleName运行结果：UserInfo
```

4.2、java.lang.reflect.Field

提供了类的属性信息。可以获取属性上的注解、修饰符、属性类型、属性名等。

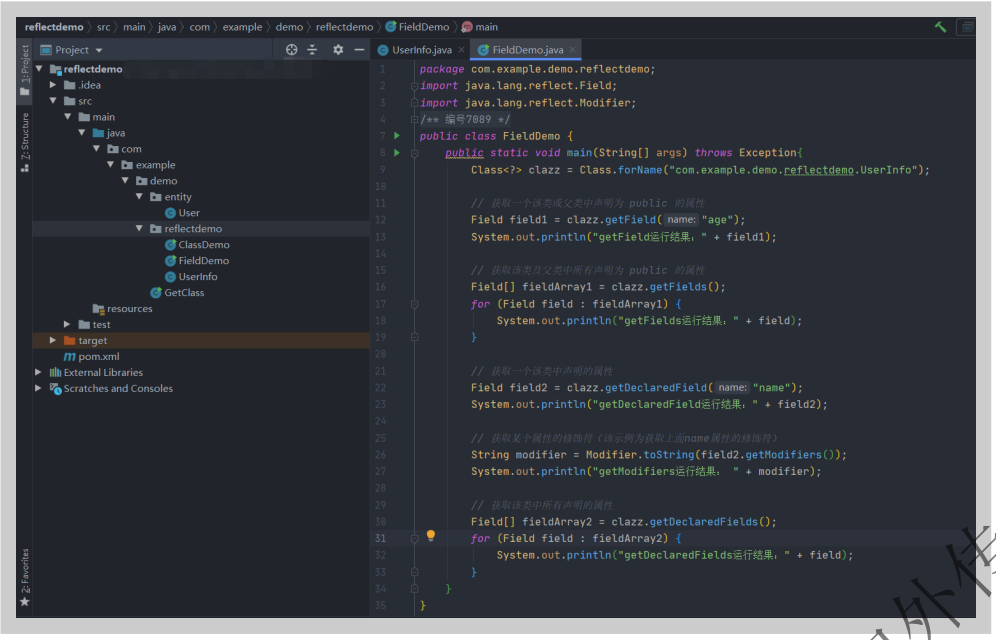
官方文档：`https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Field.html` 。

方法名	释义
getField("xxx")	获取目标类中声明为 public 的属性
getFields()	获取目标类中所有声明为 public 的属性
getDeclaredField("xxx")	获取目标类中声明的属性
getDeclaredFields()	获取目标类中所有声明的属性
更多方法可查看官方文档.....	

4.2.1、示例代码

①、在创建 `com.example.demo.reflectdemo` 下创建一个名为 `FieldDemo` 的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.demo.reflectdemo;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;

public class FieldDemo {
    public static void main(String[] args) throws Exception{
        Class<?> clazz =
Class.forName("com.example.demo.reflectdemo.UserInfo");

        // 获取一个该类或父类中声明为 public 的属性
        Field field1 = clazz.getField("age");
        System.out.println("getField运行结果：" + field1);

        // 获取该类及父类中所有声明为 public 的属性
        Field[] fieldArray1 = clazz.getFields();
        for (Field field : fieldArray1) {
            System.out.println("getFields运行结果：" + field);
        }

        // 获取一个该类中声明的属性
        Field field2 = clazz.getDeclaredField("name");
        System.out.println("getDeclaredField运行结果：" + field2);

        // 获取某个属性的修饰符（该示例为获取上面name属性的修饰符）
        String modifier = Modifier.toString(field2.getModifiers());
        System.out.println("getModifiers运行结果：  " + modifier);

        // 获取该类中所有声明的属性
        Field[] fieldArray2 = clazz.getDeclaredFields();
        for (Field field : fieldArray2) {
            System.out.println("getDeclaredFields运行结果：" + field);
        }
    }
}
```



运行结果如下图所示：

```
getField运行结果：public int com.example.demo.reflectdemo.UserInfo.age
getFields运行结果：public int com.example.demo.reflectdemo.UserInfo.age
getDeclaredField运行结果：private java.lang.String com.example.demo.reflectdemo.UserInfo.name
getModifiers运行结果： private
getDeclaredFields运行结果：private java.lang.String com.example.demo.reflectdemo.UserInfo.name
getDeclaredFields运行结果：public int com.example.demo.reflectdemo.UserInfo.age
```

4.3、java.lang.reflect.Method

提供了类的方法信息。可以获取方法上的注解、修饰符、返回值类型、方法名称、所有参数。

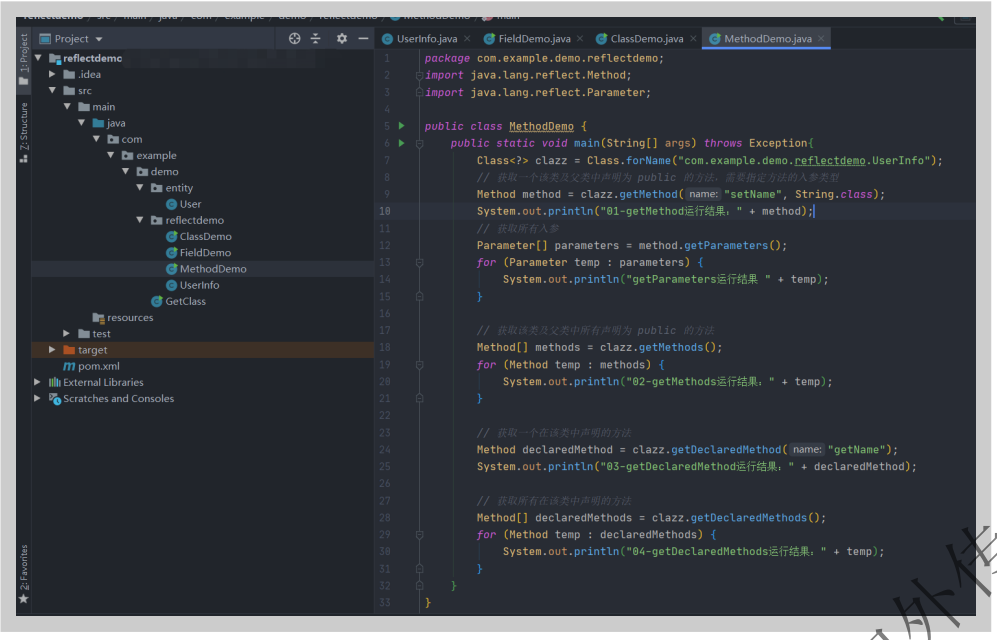
官方文档：`https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Method.html`。

方法名	释义
getMethod("setAge", String.class)	获取目标类及父类中声明为 public 的方法，需要指定方法的入参类型
getMethods()	获取该类及父类中所有声明为 public 的方法
getDeclaredMethod()	获取一个在该类中声明的方法
getDeclaredMethods()	获取所有在该类中声明的方法
getParameters()	获取所有传参
更多方法可查看官方文档.....	

4.3.1、示例代码

①、在创建 `com.example.demo.reflectdemo` 下创建一个名为 `MethodDemo` 的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.demo.reflectdemo;
import java.lang.reflect.Method;
import java.lang.reflect.Parameter;

public class MethodDemo {
    public static void main(String[] args) throws Exception{
        Class<?> clazz =
Class.forName("com.example.demo.reflectdemo.UserInfo");

        // 获取一个该类及父类中声明为 public 的方法，需要指定方法的入参类型
        Method method = clazz.getMethod("setName", String.class);
        System.out.println("01-getMethod运行结果：" + method);

        // 获取所有入参
        Parameter[] parameters = method.getParameters();
        for (Parameter temp : parameters) {
            System.out.println("getParameters运行结果 " + temp);
        }

        // 获取该类及父类中所有声明为 public 的方法
        Method[] methods = clazz.getMethods();
        for (Method temp : methods) {
            System.out.println("02-getMethods运行结果：" + temp);
        }

        // 获取一个在该类中声明的方法
        Method declaredMethod = clazz.getDeclaredMethod("getName");
        System.out.println("03-getDeclaredMethod运行结果：" +
declaredMethod);

        // 获取所有在该类中声明的方法
        Method[] declaredMethods = clazz.getDeclaredMethods();
        for (Method temp : declaredMethods) {
            System.out.println("04-getDeclaredMethods运行结果：" + temp);
        }
    }
}
```



运行结果如下图所示：

```
01-getMethod运行结果：public void com.example.demo.reflectdemo.UserInfo.setName(java.lang.String)
getParameters运行结果 java.lang.String arg0
02-getMethods运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.toString()
02-getMethods运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.getName()
02-getMethods运行结果：public void com.example.demo.reflectdemo.UserInfo.setName(java.lang.String)
02-getMethods运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.sayHello()
02-getMethods运行结果：public int com.example.demo.reflectdemo.UserInfo.getAge()
02-getMethods运行结果：public void com.example.demo.reflectdemo.UserInfo.setAge(int)
02-getMethods运行结果：public final void java.lang.Object.wait() throws java.lang.InterruptedException
02-getMethods运行结果：public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
02-getMethods运行结果：public final native void java.lang.Object.wait(long) throws java.lang.InterruptedExceptio
02-getMethods运行结果：public boolean java.lang.Object.equals(java.lang.Object)
02-getMethods运行结果：public native int java.lang.Object.hashCode()
02-getMethods运行结果：public final native java.lang.Class java.lang.Object.getClass()
02-getMethods运行结果：public final native void java.lang.Object.notify()
02-getMethods运行结果：public final native void java.lang.Object.notifyAll()
03-getDeclaredMethod运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.getName()
04-getDeclaredMethods运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.toString()
04-getDeclaredMethods运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.getName()
04-getDeclaredMethods运行结果：public void com.example.demo.reflectdemo.UserInfo.setName(java.lang.String)
04-getDeclaredMethods运行结果：private java.lang.String com.example.demo.reflectdemo.UserInfo.introduce()
04-getDeclaredMethods运行结果：public java.lang.String com.example.demo.reflectdemo.UserInfo.sayHello()
04-getDeclaredMethods运行结果：public int com.example.demo.reflectdemo.UserInfo.getAge()
04-getDeclaredMethods运行结果：public void com.example.demo.reflectdemo.UserInfo.setAge(int)
```

4.4、java.lang.reflect.Modifier

提供了访问修饰符信息。通过 `Class` 、 `Field` 、 `Method` 、 `Constructor` 等对象都可以获取修饰符，这个访问修饰符是一个整数，可以通过 `Modifier.toString` 方法来查看修饰符描述。并且该类提供了一些静态方法和常量来解码访问修饰符。

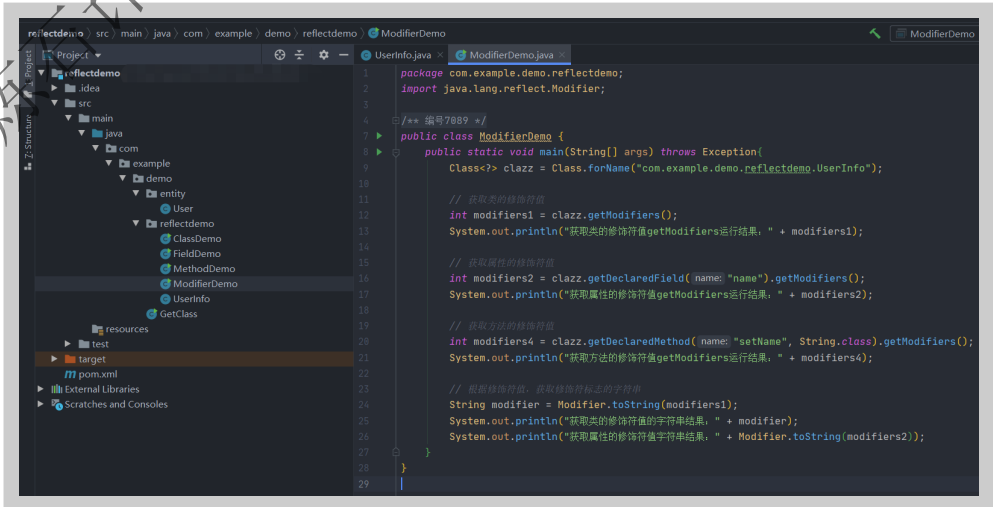
官方文档：
`https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Modifier.html`

方法名	释义
getModifiers()	获取类的修饰符值
getDeclaredField("username").getModifiers()	获取属性的修饰符值
更多方法可查看官方文档.....	

4.4.1、示例代码

①、在创建 `com.example.demo.reflectdemo` 下创建一个名为 `ModifierDemo` 的 Java Class，并键入以下代码，最终如下图所示：

```
package com.example.demo.reflectdemo;
import java.lang.reflect.Modifier;

/**
 * 编号7089
 */
public class ModifierDemo {
    public static void main(String[] args) throws Exception{
        Class<?> clazz =
Class.forName("com.example.demo.reflectdemo.UserInfo");

        // 获取类的修饰符值
        int modifiers1 = clazz.getModifiers();
        System.out.println("获取类的修饰符值getModifiers运行结果：" +
modifiers1);

        // 获取属性的修饰符值
        int modifiers2 = clazz.getDeclaredField("name").getModifiers();
        System.out.println("获取属性的修饰符值getModifiers运行结果：" +
modifiers2);

        // 获取方法的修饰符值
        int modifiers4 = clazz.getDeclaredMethod("setName",
String.class).getModifiers();
        System.out.println("获取方法的修饰符值getModifiers运行结果：" +
modifiers4);

        // 根据修饰符值，获取修饰符标志的字符串
        String modifier = Modifier.toString(modifiers1);
        System.out.println("获取类的修饰符值的字符串结果：" + modifier);
        System.out.println("获取属性的修饰符值字符串结果：" +
Modifier.toString(modifiers2));
    }
}
```



运行结果如下图所示：

```
获取类的修饰符值getModifiers运行结果：1
获取属性的修饰符值getModifiers运行结果：2
获取方法的修饰符值getModifiers运行结果：1
获取类的修饰符值的字符串结果：public
获取属性的修饰符值字符串结果：private
```

4.5、java.lang.reflect.Constructor

提供了类的构造函数信息。可以获取构造函数上的注解信息、参数类型等。

官方文档：
`https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Constructor.html` 。

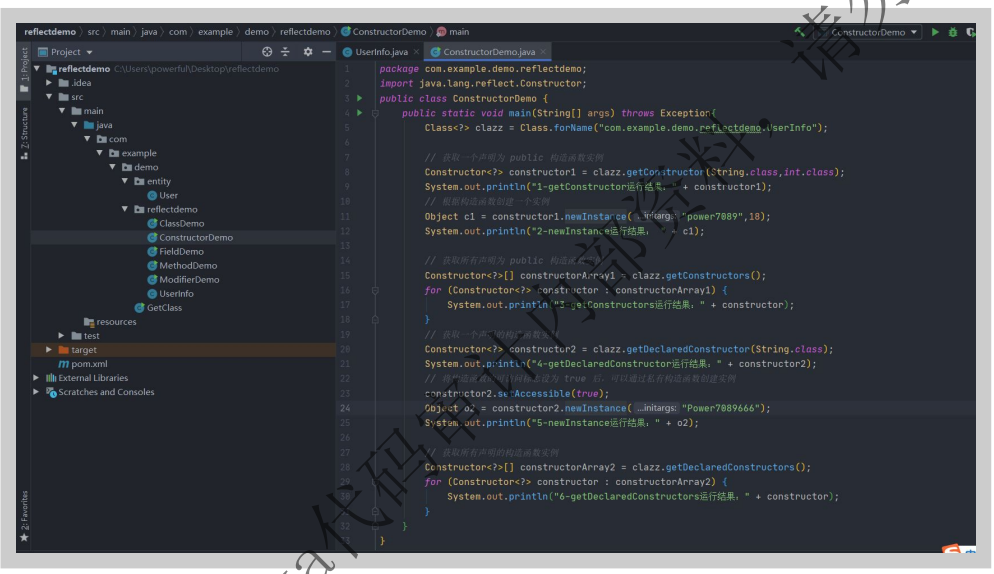
方法名	释义
getConstructor()	获取一个声明为 public 构造函数实例
getConstructors()	获取所有声明为 public 构造函数实例
getDeclaredConstructor()	获取一个声明的构造函数实例
getDeclaredConstructors()	获取所有声明的构造函数实例
更多方法可查看官方文档.....	

4.5.1、示例代码

①、在创建 `com.example.demo.reflectdemo` 下创建一个名为 `ConstructorDemo` 的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.demo.reflectdemo;
import java.lang.reflect.Constructor;
public class ConstructorDemo {
    public static void main(String[] args) throws Exception{
        Class<?> clazz =
Class.forName("com.example.demo.reflectdemo.UserInfo");

        // 获取一个声明为 public 构造函数实例
        Constructor<?> constructor1 =
clazz.getConstructor(String.class,int.class);
        System.out.println("1-getConstructor运行结果：" + constructor1);
        // 根据构造函数创建一个实例
        Object c1 = constructor1.newInstance("power7089",18);
        System.out.println("2-newInstance运行结果：  " + c1);

        // 获取所有声明为 public 构造函数实例
        Constructor<?>[] constructorArray1 = clazz.getConstructors();
        for (Constructor<?> constructor : constructorArray1) {
            System.out.println("3-getConstructors运行结果：" +
constructor);
        }
        // 获取一个声明的构造函数实例
```

```
        Constructor<?> constructor2 =
clazz.getDeclaredConstructor(String.class);
        System.out.println("4-getDeclaredConstructor运行结果：" +
constructor2);
        // 将构造函数的可访问标志设为 true 后，可以通过私有构造函数创建实例
        constructor2.setAccessible(true);
        Object o2 = constructor2.newInstance("Power7089666");
        System.out.println("5-newInstance运行结果：" + o2);

        // 获取所有声明的构造函数实例
        Constructor<?>[] constructorArray2 =
clazz.getDeclaredConstructors();
        for (Constructor<?> constructor : constructorArray2) {
            System.out.println("6-getDeclaredConstructors运行结果：" +
constructor);
        }
    }
}
```



运行结果如下图所示：

```
1-getConstructor运行结果：public com.example.demo.reflectdemo.UserInfo(java.lang.String,int)
2-newInstance运行结果：  Person{name='power7089', age=18}
3-getConstructors运行结果：public com.example.demo.reflectdemo.UserInfo(java.lang.String,int)
3-getConstructors运行结果：public com.example.demo.reflectdemo.UserInfo()
4-getDeclaredConstructor运行结果：private com.example.demo.reflectdemo.UserInfo(java.lang.String)
5-newInstance运行结果：Person{name='Power7089666', age=0}
6-getDeclaredConstructors运行结果：public com.example.demo.reflectdemo.UserInfo(java.lang.String,int)
6-getDeclaredConstructors运行结果：private com.example.demo.reflectdemo.UserInfo(java.lang.String)
6-getDeclaredConstructors运行结果：public com.example.demo.reflectdemo.UserInfo()

Process finished with exit code 0
```

4.6、java.lang.reflect.Parameter

提供了方法的参数信息。可以获取方法上的注解、参数名称、参数类型等。

官方文档：`https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Parameter.html`。

方法名	释义
getParameters()	获取构造函数/方法的参数
更多方法可查看官方文档.....	

4.7、java.lang.reflect.AccessibleObject

是 `Field` 、 `Method` 和 `Constructor` 类的超类。该类提供了对类、方法、构造函数的访问控制检查的能力（如：私有方法只允许当前类访问）。

该**访问检查**在设置/获取属性、调用方法、创建/初始化类的实例时执行。

方法名	释义
setAccessible()	将可访问标志设为 `true` （默认为 `false` ），会关闭访问检查。这样即使是私有的属性、方法或构造函数，也可以访问。

4.7.1、示例代码

可以看 `ConstructorDemo` 类代码，涉及到了 `setAccessible()` ，如下：

```
// 获取一个声明的构造函数实例
Constructor<?> constructor2 =
clazz.getDeclaredConstructor(String.class);
System.out.println("4-getDeclaredConstructor运行结果：" + constructor2);
// 将构造函数的可访问标志设为 true 后，可以通过私有构造函数创建实例
constructor2.setAccessible(true);
Object o2 = constructor2.newInstance("Power7089666");
System.out.println("5-newInstance运行结果：" + o2);
```

5、常用方法整理

1、getMethod()

getMethod()方法获取的是当前类中所有公共(public)方法。包括从父类里继承来的方法。

2、getDeclaredMethod()

getDeclaredMethod()系列方法获取的是当前类中“声明”的方法，包括private，protected和public，不包含从父类继承来的方法。

3、getConstructor()

getConstructor()方法获取的是当前类声明为公共(public)构造函数实例。

4、getDeclaredConstructor()

getDeclaredConstructor() 方法获取的是当前类声明的构造函数实例，包括private，protected和public。

5、setAccessible()

在获取到私有方法或构造方法后，使用 `setAccessible(true);` ，改变其作用域，这样及时私有的属性，方法，构造函数也都可以访问调用了。

6、newInstance()

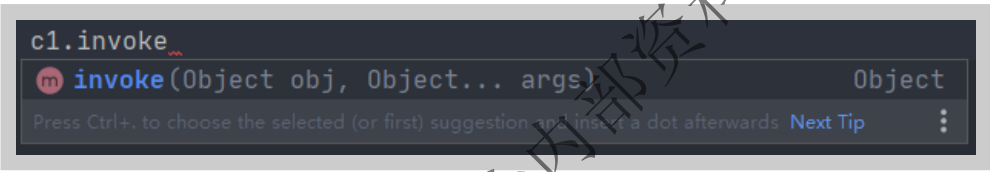
将获取到的对象实例化。调用的是这个类的无参构造函数。

使用 newInstance 不成功的话可能是因为：①、你使用的类没有无参构造函数，②、你使用的类构造函数是私有的。

7、invoke()

调用包装在当前Method对象中的方法。

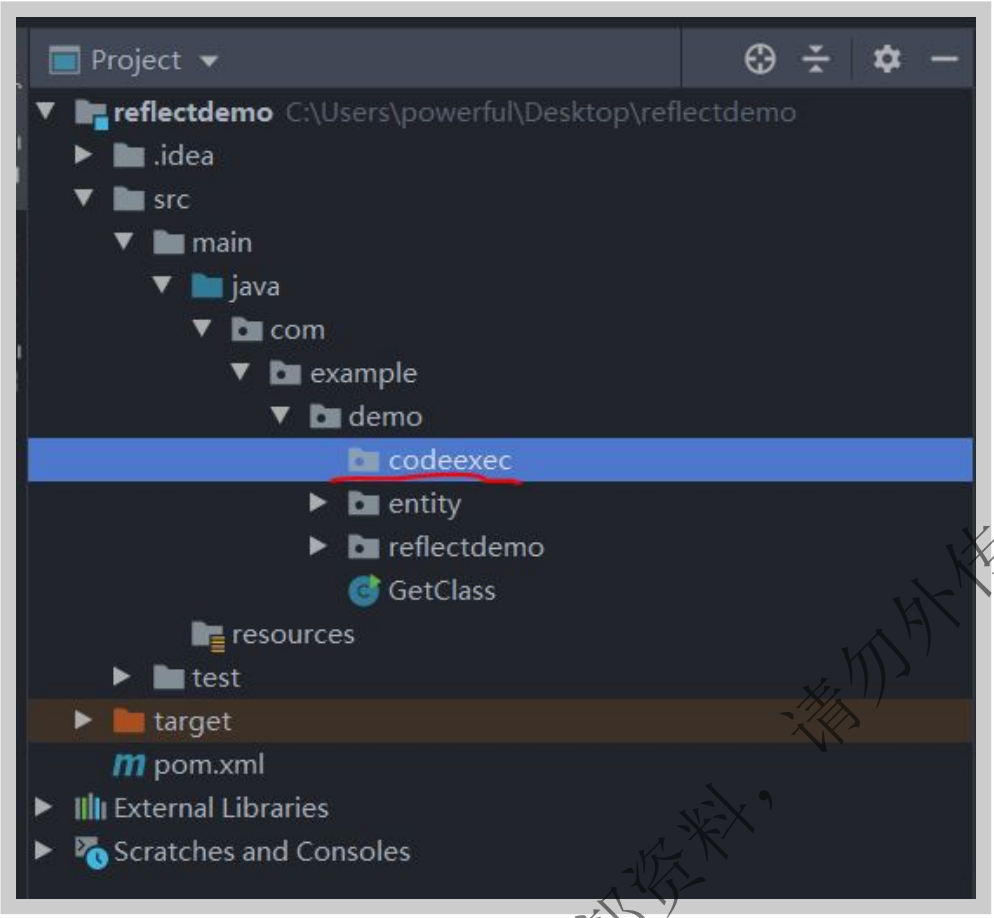
invoke传参如下图所示：



二、Java反射到命令执行

学习ava反射机制，其实我们更关心如何利用Java反射实现命令执行。下面举例讲解下Java反射命令执行的几种情况。

⚠ 首先在 `com.example.demo` 下新建一个名为 `codeexec` 的包。用于命令执行示例代码的编写调试。最终目录如下图所示：



1、反射之Java.lang.Runtime

下面是两种通过反射java.lang.Runtime来达到命令执行的方式。

1.1、方式一：通过getMethod

由于java.lang.Runtime类的构造函数是私有的，因此不能直接使用 `newInstance()` 创建一个实例。

那为什么这个类的构造函数会是私有的呢?

这涉及到一个“单例模式”的概念。举个例子：我们在链接数据库时只有最开始链接一次，而不是用一次链接一次，如果这样的话，资源消耗太大了。

因此可以将类的构造函数设为私有，再通过静态方法来获取。

由于java.lang.Runtime使用了单例模式，我们可以通过Runtime.getRuntime()来获取Runtime对象。

先看一段代码：

```
Class<?> clazz = Class.forName("java.lang.Runtime");
Method execMethod = clazz.getMethod("exec", String.class);
Method getRuntimeMethod = clazz.getMethod("getRuntime");
Object runtime = getRuntimeMethod.invoke(clazz);
execMethod.invoke(runtime, "calc.exe");
```

简单解读：

首先通过Class.forName获取java.lang.Runtime。

接下来通过getMethod()方法获exec方法，在java命令执行章节中我们了解到，`exec()`方法有六种调用方式(重载)，我们选择最简单的String方式，则getMethod方法我们设定的入参方式为`String.class`。

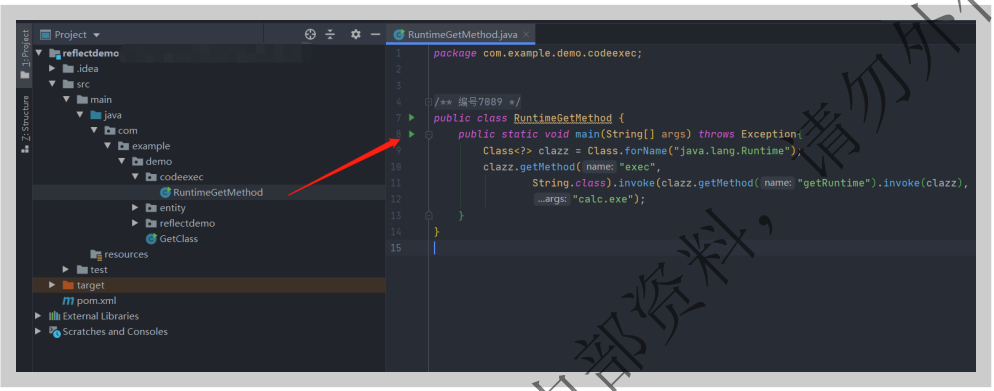
然后获取getRuntime方法后，使用invoke执行方法。

最后在通过invoke方法调用runtime对象执行命令。

将上述代码可以简化如下，简化前后有什么区别？大家可自行调试一下，观察不同。

```
Class<?> clazz = Class.forName("java.lang.Runtime");
clazz.getMethod("exec",String.class).invoke(clazz.getMethod("getRuntime
").invoke(clazz),"calc.exe");
```

①、在`com.example.demo.codeexec`下新建以及各名为`RuntimeGetMethod`的Java Class，并键入以上代码，最终如下图所示：



上述两种方式代码，自行调试运行观察结果。

1.2、方式二：通过getDeclaredConstructor

如果方法或构造函数是私有的，我们可以使用`getDeclaredMethod`或`getDeclaredConstructor`来获取执行。

在这里，java.lang.Runtime的构造函数为私有的，因此我们可以使用getDeclaredConstructor方法获取java.lang.Runtime并执行。

先看一段代码：

```
Class<?> clazz = Class.forName("java.lang.Runtime");
Constructor m = clazz.getDeclaredConstructor();
m.setAccessible(true);
Method c1 = clazz.getMethod("exec", String.class);
c1.invoke(m.newInstance(), "calc.exe");
```

简单解读：

首先通过Class.forName获取java.lang.Runtime。

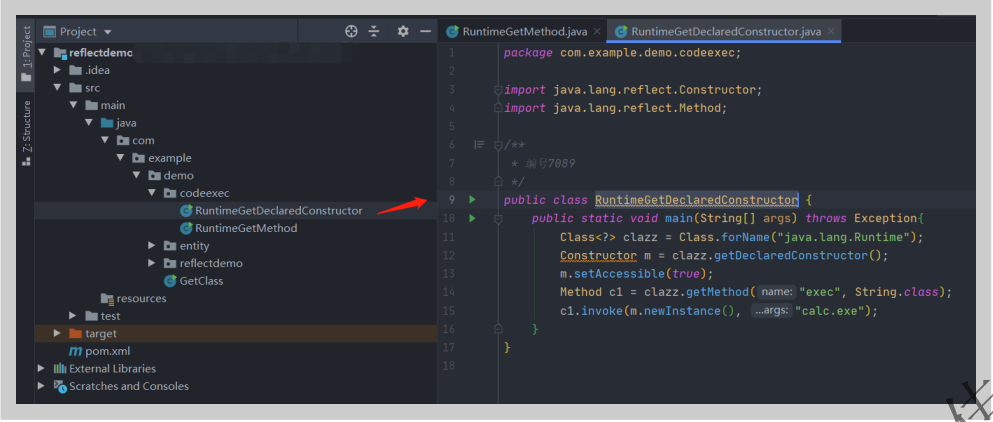
接下来通过getDeclaredConstructor获取构造函数。

通过`setAccessible(true)`设置改变作用域，让我们可以调用他的私有构造函数。

调用exec方法，入参设置为`String.class`。

最后使用Invoke执行方法。

①、在 `com.example.demo.codeexec` 下新建以及各名为 `RuntimeGetDeclaredConstructor` 的Java Class，并键入以上代码，最终如下图所示：



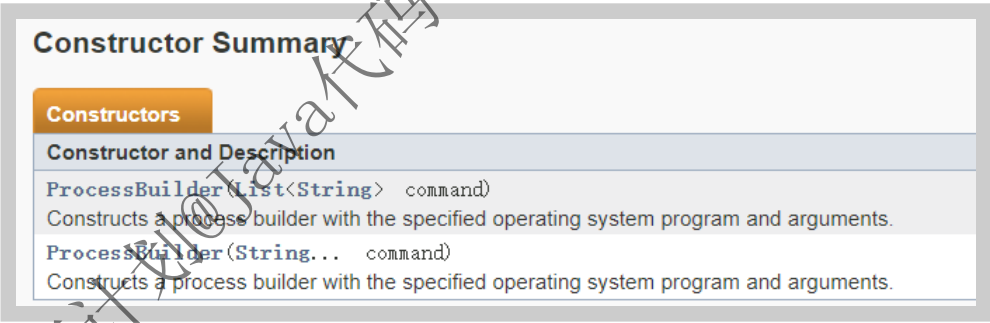
自行调试运行观察结果。

2、反射之java.lang.ProcessBuilder

如果一个类没有无参构造方法，也没有类似单例模式里的静态方法，我们可以通过 `getConstructor()` 方法实例化该类。当然也可以使用 `getDeclaredConstructor()` 方法。

java.lang.ProcessBuilder有两个构造函数，构造函数也是支持重载的。如下图所示：

```
ProcessBuilder(List<String> command)
ProcessBuilder(String... command)
```

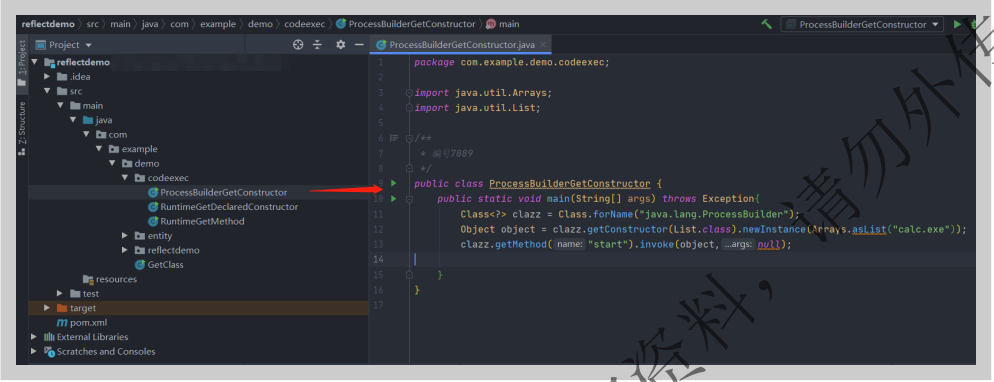


在Java命令执行章节，我们了解到java.lang.ProcessBuilder使用start()方法执行命令。

我们以 `ProcessBuilder(List<String> command)` 为例。进行讲解。

①、在 `com.example.demo.codeexec` 下新建以及各名为 `ProcessBuilderGetConstructor` 的Java Class，并键入以下代码，最终如下图所示：

```
package com.example.demo.codeexec;
import java.util.Arrays;
import java.util.List;

public class ProcessBuilderGetConstructor {
    public static void main(String[] args) throws Exception{
        Class<?> clazz = Class.forName("java.lang.ProcessBuilder");
        Object object =
clazz.getConstructor(List.class).newInstance(Arrays.asList("calc.exe"))
;
        clazz.getMethod("start").invoke(object,null);
    }
}
```



写法：

```
Class clazz = Class.forName("java.lang.ProcessBuilder");
((ProcessBuilder)
clazz.getConstructor(List.class).newInstance(Arrays.asList("calc.exe"))
).star
t();
```

```
Class clazz = Class.forName("java.lang.ProcessBuilder");
clazz.getMethod("start").invoke(clazz.getConstructor(List.class).newIns
tance(
Arrays.asList("calc.exe")));
```

大家自行运行观察分析结果。

⚠注意：

在开头我们介绍引入的是反射调用java.lang.ProcessImpl，代码如下。留个作业，大家自行调试，将分析结果形成文档提交到本章节对应的作业处“【第一阶段】Java代码审计之基础篇作业”。

反射这节基础很重要，希望大家能够积极练习，并记录笔记提交到对应的作业处。

```
Class clazz = Class.forName("java.lang.ProcessImpl");
Method method = clazz.getDeclaredMethod("start", String[].class,
Map.class, String.class, ProcessBuilder.Redirect[].class,
boolean.class);
method.setAccessible(true);
Process process = (Process) method.invoke(null, cmds, null, ".", null,
true);
```

至此，Java反射机制知识点到这就结束了。

本章节提到了一些特殊场景以及解决方法。

当然在实际中，我们会遇见各种情况，加以分析，再配合掌握的这些函数方法后，可以更好的解决应对。