ReflectDemo1

```
package reflect;

import java.lang.reflect.Method;
import java.util.Scanner;

/**
 * java反射机制：
 * 反射机制是java的动态机制，可以在程序“运行期间”再确定实例化对象、方法调用、属性操作等。
 * 反射机制可以提高代码的灵活度，但是会带来较多的系统开销和较低的运行效率，因此不能过度依赖。
 */
public class ReflectDemo1 {
    public static void main(String[] args) throws ClassNotFoundException {
        /*
        Class类：Class类称为类的类对象。
        JVM加载一个类的class文件时，就会创建一个Class实例与该类绑定。
        因此每个被加载的类都有且只有一个Class实例，这个实例就是加载的该类的类对象。
        通过一个类的类对象我们可以获取这个类的一切信息（类名、属性、方法、构造器等），从而在程序运行期间进行相关
的操作。

        因此反射第一步就是要获取操作的类的类对象。获取方式有三种：
        1、类名.class：
        Class cls = String.class;
```

```
2、Class.forName(String className):
Class cls = Class.forName("java.lang.String"); //参数需要是完全限定名：包名.类名
注意：基本类型不支持此种方式获取类对象！

3、ClassLoader类加载器形式获取：......
 */

//1、类名.class:
//Class cls = String.class;

//2、Class.forName(String className):
//Class cls = Class.forName("java.lang.String");

/*
可用例子：
java.util.ArrayList
java.util.HashMap
java.io.FileInputStream
java.lang.String
reflect.Person
 */
Scanner scanner = new Scanner(System.in);
System.out.println("请输入要加载的类的名字：");
String className = scanner.nextLine();
Class cls = Class.forName(className);

String name = cls.getName(); //获取完全限定名：包名.类名
System.out.println(name);
```

```
        name = cls.getSimpleName(); //仅获取类名
        System.out.println(name);

        String packageName = cls.getPackage().getName(); //获取包名
        System.out.println(packageName);

        //获取String类的所有public方法（包含从超类（Object）继承的方法）
        //Method[] methods = cls.getMethods();

        //获取本类自己定义的方法（包含private方法）
        Method[] methods = cls.getDeclaredMethods();
        for (Method method : methods) {
            System.out.println(method.getName());
        }
    }
}
```

ReflectDemo2

```
package reflect;

import java.util.ArrayList;
import java.util.Scanner;

//使用反射机制实例化对象
public class ReflectDemo2 {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException {
        ArrayList<String> list = new ArrayList<>();
        System.out.println(list);

        //Class cls = Class.forName("java.util.ArrayList");

        //1、获取要实例化对象的类的类对象
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入类名：");
        /*
        java.util.HashMap
        java.util.Date
        reflect.Person
         */
        String className = scanner.nextLine();
        Class cls = Class.forName(className);
```

```
        //2、调用Class的newInstance()方法来调用无参构造器
        Object obj = cls.newInstance();
        System.out.println(obj);
    }
}
```

Person

```java
package reflect;

import reflect.annotations.AutoRunClass;
import reflect.annotations.AutoRunMethod;

@AutoRunClass
public class Person {
    private String name = "张三";
    private int age = 22;

    public Person() {

    }

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

    @AutoRunMethod(5)
    public void sayHello() {
        System.out.println(name + ": hello! ");
    }
```

```
    public void sayHi() {
        System.out.println(name + ": hi! ");
    }

    public void dance() {
        System.out.println(name + ": 跳舞! ");
    }

    public void sing() {
        System.out.println(name + ": 唱歌! ");
    }

    public void watchTV() {
        System.out.println(name + ": 看电视! ");
    }

    public void playGame() {
        System.out.println(name + ": 玩游戏! ");
    }

    public void say(String info) {
        System.out.println(name + ": 说: " + info);
    }

    public void say(String info, int count) {
        for (int i = 0; i < count; i++) {
            System.out.println(name + ": 说: " + info);
        }
```

```
    }

    private void hehe() {
        System.out.println("我是Person类的private方法! ");
    }

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

ReflectDemo3

```
package reflect;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

//使用指定的构造器实例化对象
public class ReflectDemo3 {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, InvocationTargetException, InstantiationException,
IllegalAccessException {
        Person p = new Person("李四", 33);
        System.out.println(p);

        //1、加载类对象
        Class cls = Class.forName("reflect.Person");

        //2、通过类对象获取指定的有参构造器：Person(String,int)
        //Constructor c = cls.getConstructor(); //不传参获取的是无参构造器
        Constructor c = cls.getConstructor(String.class, int.class); //指定是哪个构造器
        Object obj = c.newInstance("王五", 66); //new Person("王五",66);
        System.out.println(obj);
    }
}
```

ReflectDemo4

```
package reflect;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Scanner;

//利用反射机制调用方法
public class ReflectDemo4 {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, NoSuchMethodException,
InvocationTargetException {
        Person p = new Person();
        p.watchTV();

        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入类名：");  //reflect.Person
        String className = scanner.nextLine();
        System.out.println("请输入方法名：");  //watchTV
        String methodName = scanner.nextLine();

        //1、实例化
        //1.1、加载类对象
        Class cls = Class.forName(className);
        //1.2、实例化
```

```
        Object obj = cls.newInstance();

        //2、调用方法
        //2.1、通过类对象获取要调用的方法
        Method method = cls.getMethod(methodName);
        //2.2、通过方法对象执行该方法
        method.invoke(obj); //obj.watchTV()，前提是obj表示的是一个Person对象
    }
}
```

Student

```java
package reflect;

import reflect.annotations.AutoRunClass;
import reflect.annotations.AutoRunMethod;

@AutoRunClass
public class Student {

    @AutoRunMethod
    public void study() {
        System.out.println("student：good good study，day day up！");
    }
}
```

ReflectDemo5

```
package reflect;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

//调用有参方法
public class ReflectDemo5 {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, NoSuchMethodException,
InvocationTargetException {
        Class cls = Class.forName("reflect.Person");
        Object obj = cls.newInstance();

        Method method = cls.getMethod("say", String.class);
        method.invoke(obj, "你好！");

        Method method2 = cls.getMethod("say", String.class, int.class);
        method2.invoke(obj, "嘿嘿", 3);
    }
}
```

ArgsDemo

```java
package reflect;

import java.util.Arrays;

//可变长参数
public class ArgsDemo {
    public static void main(String[] args) {
        dosome(1, "a", "b", "c"); //dosome(new String[]{"a","b"});
    }

    //可变长参数会被编译器改为数组。
    //可变长参数只能是方法的【最后一个】参数。（暗示可变长参数只能有一个）
    public static void dosome(int d, String... s) { //(String[] s)
        System.out.println(s.length);
        System.out.println(Arrays.toString(s));
    }
}
```

ReflectDemo6

```
package reflect;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class ReflectDemo6 {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, NoSuchMethodException,
InvocationTargetException {
        //Person p = new Person();
        //p.hehe(); //编译不通过！hehe是私有方法！

        Class cls = Class.forName("reflect.Person");
        Object obj = cls.newInstance();

        //Class的方法：getMethod()、getMethods()都只能获取到类对象表示的类的public方法
        //Method method = cls.getMethod("hehe");
        Method method = cls.getDeclaredMethod("hehe"); //获取本类自己定义的方法（包含private方法）
        method.setAccessible(true); //强行打开访问权限，使private可以被外部访问，不过慎用！
        method.invoke(obj);
        method.setAccessible(false); //用完就关！
    }
}
```

Test

```
package reflect;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

/**
 * 自动调用Person类中所有无参方法：
 * <p>
 * Method的方法：
 * int getParameterCount()：
 * 返回当前Method表示的方法的参数个数
 */
public class Test {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, InvocationTargetException {
        Class cls = Class.forName("reflect.Person");
        Object obj = cls.newInstance();

        Method[] methods = cls.getDeclaredMethods();
        for (Method method : methods) {
            //判断是否为无参且public且方法名含字母a的方法
            if (method.getParameterCount() == 0 && method.getModifiers() == Modifier.PUBLIC &&
method.getName().contains("a")) {
```

```
                method.invoke(obj);
            }
        }
    }
}
```

Test2

```
package reflect;

import java.io.File;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.net.URISyntaxException;

//自动调用与当前类Test2在同一个包中所有类中的无参且public的方法
public class Test2 {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, URISyntaxException, InvocationTargetException {
        File dir = new File(Test2.class.getResource(".").toURI()); //定位当前类所在的包（目录）
        File[] subs = dir.listFiles(f -> f.getName().endsWith(".class"));
        String packageName = Test2.class.getPackage().getName();
        for (File sub : subs) {
            //通过每一个class文件的文件名得到类名
            //根据Test2这个类的包名来拼接其它同包类的完全限定名
            //加载该类的类对象
            Class cls = Class.forName(packageName + "." + sub.getName().split("\\.")[0]);
//substring(0,fileName.indexOf("."))
            Object obj = cls.newInstance(); //实例化对象
            Method[] methods = cls.getDeclaredMethods(); //根据类对象获取该类中所有定义的方法
            for (Method method : methods) {
```

```
                if (method.getParameterCount() == 0 && method.getModifiers() ==
Modifier.PUBLIC) {
                    method.invoke(obj);
                }
            }
        }
    }
}
```

AutoRunClass

```java
package reflect.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * 注解：
 * 注解被大量应用于反射中，可以辅助反射机制做更多灵活的操作。
 * 后期使用的框架基本都是使用注解进行配置的。
 * <p>
 * 在注解上我们有两个常用的注解用来说明当前注解的相关特性：
 * <p>
 * @Target：用于说明我们定义的注解可以被应用到哪里。
 * 该注解可以传递的参数为ElementType类型，用来表示可标注的位置。
 * 常见的有：
 * ElementType.TYPE：在类上用
 * ElementType.METHOD：在方法上用
 * ElementType.FIELD：在属性上用
 * 如果不指定@Target，则当前注解可以被应用在任何注解可使用的位置上。
 * <p>
 * @Retention：当前注解的保留级别，有三个可选值 RetentionPolicy.SOURCE：当前注解仅保留在源代码中，编译后的字节码文件中没有该注解
```

```
 * RetentionPolicy.CLASS：保留到编译后的字节码文件中，但是反射机制不可访问
 * RetentionPolicy.RUNTIME：保留到字节码文件中且反射机制可以访问
 * 不指定时，默认保留级别为CLASS
 */
@Target(ElementType.TYPE) //多个时的数组形式：{ElementType.TYPE,......}
@Retention(RetentionPolicy.RUNTIME)
public @interface AutoRunClass {

}
```

AutoRunMethod

```java
package reflect.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface AutoRunMethod {
    int value() default 2;
    //String name();

    /*
    注解中定义参数的语法：
    类型 参数名() default 默认值; //“default 默认值”是可写可不写的

    如果当前注解只有一个参数，那么参数名应当叫value。
    此时当我们在外面使用该注解时，可以使用@AutoRunMethod(3)进行传参。
    此时注解中value的值为3。

    如果该参数名不叫value，例如，定义时如下：
    public @interface AutoRunMethod{
        int num();
```

```
}
```
此时当我们在外面使用该注解时，可以使用：
```
@AutoRunMethod(num=3) //传参时格式为：参数名=参数值
```
进行传参。此时注解中num的值为3。

只有当参数名叫value时，传参时才可以将参数名忽略。

以上特性仅限于只有一个参数时。

当有多个参数时，例如，定义时如下：
```
public @interface AutoRunMethod{
    int value();
    String name();
}
```
使用注解传递参数时，参数名都不可以忽略，必须写为name=value的格式，且顺序无所谓。
例如：
```
@AutoRunMethod(value=1,name="张三")
@AutoRunMethod(name="张三",value=1)
```
以上都正确

参数可以使用default指定默认值，那么在使用注解时，可以不指定该参数的值，使用的是默认值。
```
public @interface AutoRunMethod{
   int value() default 1;
}
```
使用该注解时：
```
@AutoRunMethod(3)：此时value的值为3
@AutoRunMethod：此时value的值为1（默认值）
 */
```

```
}
```

ReflectDemo7

```java
package reflect;

import reflect.annotations.AutoRunClass;

//反射机制访问注解
public class ReflectDemo7 {
    public static void main(String[] args) throws ClassNotFoundException {
        Class cls = Class.forName("reflect.Person");
        /*
            Class、Method等常用的反射对象都提供了一个方法：
            boolean isAnnotationPresent(Class cls)：
            用于判断是否被参数指定的注解标注过（参数为注解的类对象）
         */
        boolean flag = cls.isAnnotationPresent(AutoRunClass.class);
        System.out.println(flag);
    }
}
```

Test3

```
package reflect;

import reflect.annotations.AutoRunClass;
import reflect.annotations.AutoRunMethod;

import java.io.File;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.net.URISyntaxException;

//输出与当前类Test3在同包中被@AutoRunClass标注过的类的类名
public class Test3 {
    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, URISyntaxException, InvocationTargetException {
        File dir = new File(Test3.class.getResource(".").toURI());
        File[] subs = dir.listFiles(f -> f.getName().endsWith(".class"));
        String packageName = Test3.class.getPackage().getName();
        for (File sub : subs) {
            Class cls = Class.forName(packageName + "." + sub.getName().split("\\.")[0]);
            if (cls.isAnnotationPresent(AutoRunClass.class)) {
                System.out.println(cls.getName() + "被@AutoRunClass标注了");
                Object obj = cls.newInstance(); //类对象的实例对象的实例化
                Method[] methods = cls.getDeclaredMethods(); //输出该类中所有被@AutoRunMethod标注的
方法
```

```
                for (Method method : methods) {
                    if (method.isAnnotationPresent(AutoRunMethod.class)) {
                        System.out.println(cls.getName() + "." + method.getName() + "()被
@AutoRunMethod标注了");
                        AutoRunMethod arm = method.getAnnotation(AutoRunMethod.class);
                        int value = arm.value();
                        for (int i = 0; i < value; i++) {
                            method.invoke(obj);
                        }
                    }
                }
            }
        }
    }
}
```

ReflectDemo8

```
package reflect;

import reflect.annotations.AutoRunMethod;

import java.lang.reflect.Method;

//获取注解中的参数
public class ReflectDemo8 {
    public static void main(String[] args) throws ClassNotFoundException, NoSuchMethodException
{
        Class cls = Class.forName("reflect.Person");
        Method method = cls.getMethod("sayHello");
        if (method.isAnnotationPresent(AutoRunMethod.class)) {
            AutoRunMethod arm = method.getAnnotation(AutoRunMethod.class); //获取该方法上的注解
@AutoRunMethod
            int value = arm.value(); //通过注解对象获取value参数的值
            System.out.println(value);
        }
    }
}
```