# 反射

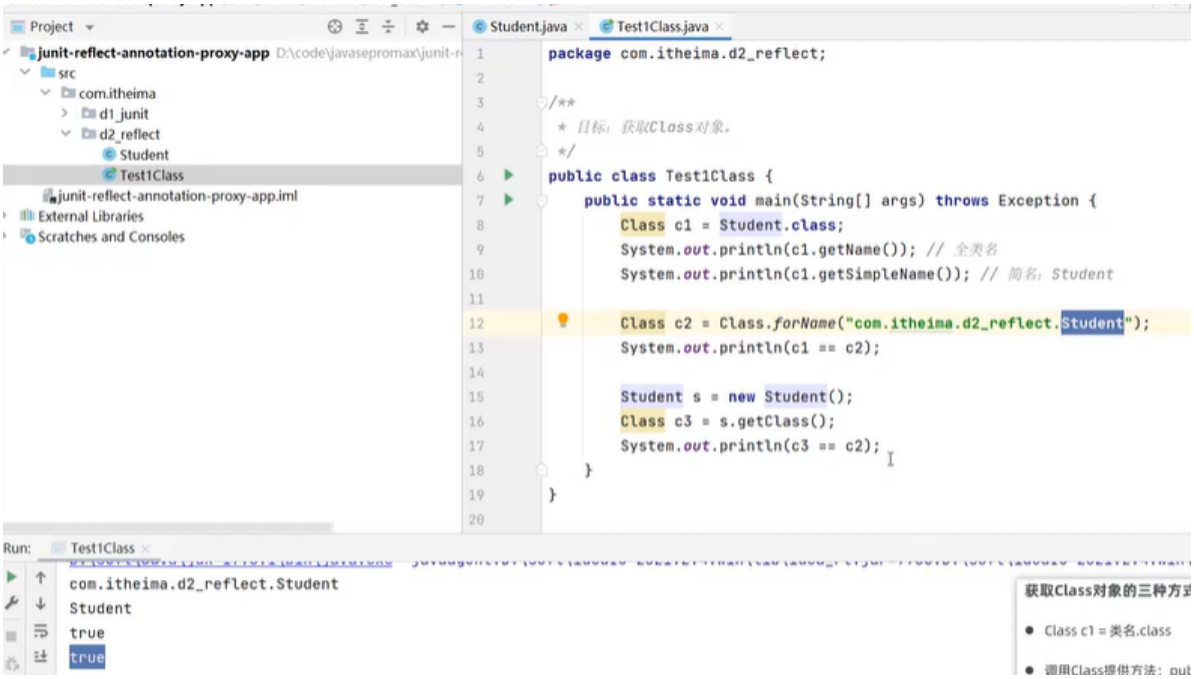
## 一、认识反射

### 反射（Reflection）

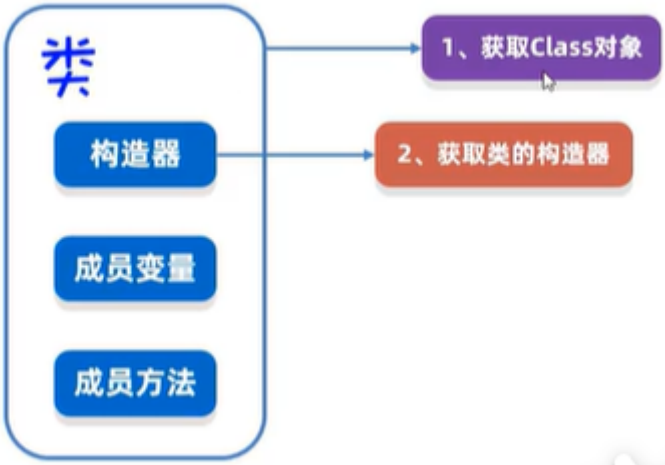
- 反射就是：加载类，并允许以编程的方式解剖类中的各种成分（成员变量、方法、构造器等）。


### 获取Class对象的三种方式

- Class c1 = 类名.class
- 调用Class提供方法： public static Class forName(String package);
- Object提供的方法： public Class getClass(); Class c3 = 对象.getClass();


Project
junit-reflect-annotation-proxy-app
src
com.itheima
d1_junit
d2_reflect
Student
Test1Class
junit-reflect-annotation-proxy-app.iml
External Libraries
Scratches and Consoles
Student.java
Test1Class.java
package com.itheima.d2_reflect;
/**
* 目标：获取Class对象。
*/
public class Test1Class {
public static void main(String[] args) throws Exception {
Class c1 = Student.class;
System.out.println(c1.getName()); // 全类名
System.out.println(c1.getSimpleName()); // 简名：Student
Class c2 = Class.forName("com.itheima.d2_reflect.Student");
System.out.println(c1 == c2);
Student s = new Student();
Class c3 = s.getClass();
System.out.println(c3 == c2);
}
}
Run:
Test1Class
com.itheima.d2_reflect.Student
Student
true
true
获取Class对象的三种方式
Class c1 = 类名.class

## 二、获取类的构造器


类
构造器
成员变量
成员方法
1、获取Class对象
2、获取类的构造器

## 获取类的构造器、并对其进行操作

- Class提供了从类中获取构造器的方法。

| 方法 | 说明 |
| --- | --- |
| Constructor<?>[] getConstructors() | 获取全部构造器（只能获取public修饰的） |
| Constructor<?>[] getDeclaredConstructors() | 获取全部构造器（只要存在就能拿到） |
| Constructor<T> getConstructor(Class<?>... parameterTypes) | 获取某个构造器（只能获取public修饰的） |
| Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes) | 获取某个构造器（只要存在就能拿到） |

## 获取类构造器的作用：依然是初始化对象返回

| Constructor提供的方法 | 说明 |
| --- | --- |
| T newInstance(Object... initargs) | 调用此构造器对象表示的构造器，并传入参数，完成对象的初始化并返回 |
| public void setAccessible(boolean flag) | 设置为true，表示禁止检查访问控制（暴力反射） |

EXP:

```
public class Data {
private String a;

    public Data(int a,int b){

    }

    public Data(){

    }



    public void p(String c){

    }

    public String q(Object obj){

        return a;
    }

}
```

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



public class w {
```

```
    public static void main(String[] args) {
    //1、反射第一步，先获得这个类的class对象
    Class c=Data.class;
    //2、获取类的全部的构造器
    Constructor[] Constructors=c.getDeclaredConstructors();
    //3、遍历构造对象
for(Constructor i:Constructors){
    System.out.println(i.getName()+"---
>"+i.getParameterCount());//getParameterCount为构造器的数量
}

 //1、反射第一步，先获得这个类的class对象
 Class d=Data.class;
//2、获取某个构造器，无参构造器
try {
    Constructor con=d.getDeclaredConstructor();
    System.out.println(con.getName()+"--->"+con.getParameterCount());
} catch (NoSuchMethodException | SecurityException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

//3、获取有参构造器
try {
    Constructor cons=d.getDeclaredConstructor(int.class,int.class);
    System.out.println(cons.getName()+"--->"+cons.getParameterCount());
} catch (NoSuchMethodException | SecurityException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}
```

Putout:

```
Data--->2
Data--->0
Data--->0
Data--->2
```

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



public class w {

    public static void main(String[] args) throws InstantiationException,
IllegalAccessException, IllegalArgumentException, InvocationTargetException {
```

```
 //1、反射第一步，先获得这个类的class对象
 Class d=Data.class;
//2、获取某个构造器，无参构造器
try {
    Constructor con=d.getDeclaredConstructor();
    System.out.println(con.getName()+"--->"+con.getParameterCount());
    con.setAccessible(true);
    Data data=(Data) con.newInstance();
    System.out.println(data);
} catch (NoSuchMethodException | SecurityException e) {
    // TODO Auto-generated catch block

    e.printStackTrace();
}

//3、获取有参构造器
try {
    Constructor cons=d.getDeclaredConstructor(int.class,int.class);
    System.out.println(cons.getName()+"--->"+cons.getParameterCount());
    cons.setAccessible(true);
    Data data=(Data) cons.newInstance(2,3);
    System.out.println(data);
} catch (NoSuchMethodException | SecurityException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}




}
```

```
public class Data {
private int a;
private int b;
private String c;

    public Data(int a,int b){
        this.a=a;
        this.b=b;
System.out.println("有参数");
    }

    public Data(){
System.out.println("无参数");
    }



    public void p(String c){

    }
```

```
    public String q(Object obj){

        return c;
    }

    public String toString() {
        return "Data{" +
                "a=" + a +
                ", b=" + b +
                '}';
    }
}
```

```
Data--->0
无参数
Data{a=0, b=0}
Data--->2
有参数
Data{a=2, b=3}
```

# 三、获取成员变量

**获取类的成员变量**

- Class提供了从类中获取成员变量的方法。

| 方法 | 说明 |
| --- | --- |
| public Field[] getFields() | 获取类的全部成员变量（只能获取public修饰的） |
| public Field[] getDeclaredFields() | 获取类的全部成员变量（只要存在就能拿到） |
| public Field getField(String name) | 获取类的某个成员变量（只能获取public修饰的） |
| public Field getDeclaredField(String name) | 获取类的某个成员变量（只要存在就能拿到） |

**获取到成员变量的作用：依然是赋值、取值。**

| 方法 | 说明 |
| --- | --- |
| void set(Object obj, Object value): | 赋值 |
| Object get(Object obj) | 取值 |
| public void setAccessible(boolean flag) | 设置为true，表示禁止检查访问控制（暴力反射） |

```
import java.lang.reflect.*;

public class w {

    public static void main(String[] args)throws Exception {

 //1、反射第一步，先获得这个类的class对象
 Class d=Data.class;
 //2、获取全部成员变量
 Field[] fields=d.getDeclaredFields();
//3、遍历这个成员变量
for(Field i:fields){
    System.out.println(i.getName()+"--->"+i.getType());
}

//4、定位某个成员变量
Field field1=d.getDeclaredField("a");
System.out.println(field1.getName()+"--->"+field1.getType());

//赋值
Data data=new Data();
field1.setAccessible(true);
field1.set(data,1);
System.out.println(data);

//取值
int a=(int) field1.get(data);
System.out.println(a);


}

}
```

```
public class Data {
private int a;
private int b;
private String c;

    public Data(int a,int b){
        this.a=a;
        this.b=b;
System.out.println("有参数");
    }

    public Data(){
System.out.println("无参数");
    }



    public void p(String c){
```

```
    }

    public String q(Object obj){

        return c;
    }

    public String toString() {
        return "Data{" +
                "a=" + a +
                ", b=" + b +
                '}';
    }
}
```

```
a--->int
b--->int
c--->class java.lang.String
a--->int
无参数
Data{a=1, b=0}
1
```

# 四、获取成员方法

- Class提供了从类中获取成员方法的API。

| 方法 | 说明 |
| --- | --- |
| Method[] getMethods() | 获取类的全部成员方法（只能获取public修饰的） |
| Method[] getDeclaredMethods() | 获取类的全部成员方法（只要存在就能拿到） |
| Method getMethod(String name, Class<?>... parameterTypes) | 获取类的某个成员方法（只能获取public修饰的） |
| Method getDeclaredMethod(String name, Class<?>... parameterTypes) | 获取类的某个成员方法（只要存在就能拿到） |

**成员方法的作用：依然是执行**

| Method提供的方法 | 说明 |
| --- | --- |
| public Object invoke(Object obj, Object... args) | 触发某个对象的该方法执行。 |
| public void setAccessible(boolean flag) | 设置为true，表示禁止检查访问控制（暴力反射） |

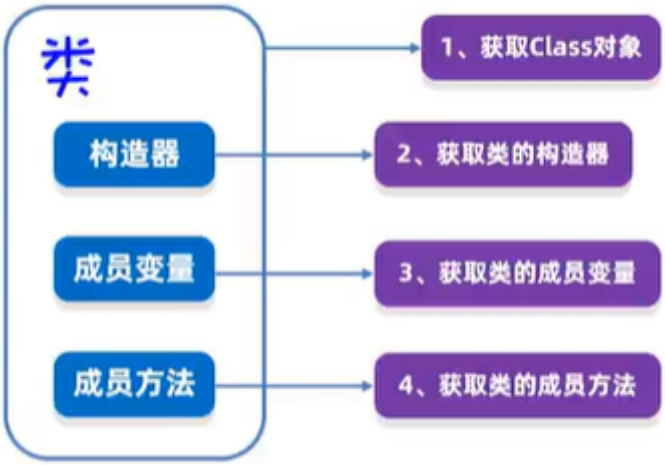
```
public class w {

    public static void main(String[] args)throws Exception {
```

```
 //1、反射第一步，先获得这个类的class对象
 Class d=Data.class;
 //2、获取类的成员方法
 Method[] methods=d.getDeclaredMethods();
 //3、遍历数组中每个方法对象
 for(Method i:methods){
    System.out.println(i.getName()+"-->"
    +i.getParameterCount()+"-->"
    +i.getReturnType());
 }
//4、获取某个成员方法
Method method=d.getDeclaredMethod("q",Object.class);
//5、输出
System.out.println(method.getName()+"-->"
    +method.getParameterCount()+"-->"
    +method.getReturnType());

Data data=new Data();
String p1 = (String) method.invoke(data, "Java1");
System.out.println(p1);

}

}
```

```
public class Data {
private int a;
private int b;
private String c;

    public Data(int a,int b){
        this.a=a;
        this.b=b;
System.out.println("有参数");
    }

    public Data(){
System.out.println("无参数");
    }



    public void p(String c){
        System.out.println("ppp");
    }

    public String q(Object obj){
 System.out.println("qqq");
        return c;
    }

    public String toString() {
```

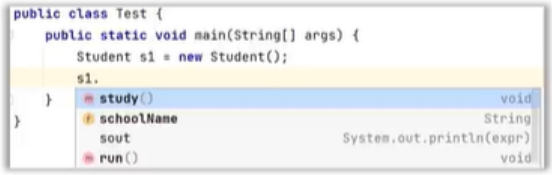
```
        return "Data{" +
                "a=" + a +
                ", b=" + b +
                '}';
    }
}
```

# 五、反射的作用




**反射的作用？**

- 基本作用：可以得到一个类的全部成分然后操作。
- 可以破坏封装性。




- 最重要的用途是：适合做Java的框架，基本上，主流的框架都会基于反射设计出一些通用的功能。

**需求：**

- 对于任意一个对象，该框架都可以把对象的字段名和对应的值，保存到文件中去。

**实现步骤**

① 定义一个方法，可以接收任意对象。

② 每收到一个对象后，使用反射获取该对象的Class对象，然后获取全部的成员变量。

③ 遍历成员变量，然后提取成员变量在该对象中的具体值。

④ 把成员变量名、和其值，写出到文件中去即可。