第二十二章 反射机制

1. 反射的概述

1. 什么是反射
    a. 定义
        i. Java代码在运行的过程中，对任意的一个类，都可以获得他的所有成员和继承关系
        ii. 对任意的对象，都可以访问它的属性和方法
        iii. 这种动态获取信息和动态调用的方式称为Java的反射机制
    b. 原理
        i. Java字节码在运行前，会加载到JVM中，形参类的Class字节码文件对象
        ii. 反射机制操作Class对象来获取类的结构信息（字段，构造器，方法和继承关系等）
        iii. 利用这些信息可以创建对象，访问方法和属性
2. 反射的作用
    a. 优点
        i. 提高程序的扩展性，降低了耦合性，提高程序的自适应能力
        ii. 创建对象和访问成员不用硬编码
    b. 缺点
        i. 反射降低了程序的性能，主要应用在对扩展性要求比较高的框架上，比如：SSM，SpringBoot等
        ii. 反射模糊了程序的内部逻辑，降低了可读性
        iii. 打破了Java封装机制，私有成员其他类也可以访问

2. 获得Class对象的四种方式

代码块

```
1  package com.powernode.reflection04;
2
3  class Teacher{
4      static {
5          System.out.println("Teacher.static initializer");
6      }
7  }
```

```
public class Test {
    public static void main(String[] args) throws Exception {
        //方式一：通过类名称.class获得class对象
        Class<Teacher> class1 = Teacher.class;

        //方式二：通过对象.getClass获得Class对象
        Teacher teacher = new Teacher();
        Class<? extends Teacher> class2 = teacher.getClass();

        //方式三：Class.forName("包名 + 类名")
        Class<?> class3 = Class.forName("com.powernode.reflection04.Teacher");

        //方式四：通过类加载器获得Classd对象
        //获得系统的类加载器
        ClassLoader systemClassLoader = ClassLoader.getSystemClassLoader();
        Class<?> class4 =
systemClassLoader.loadClass("com.powernode.reflection04.Teacher");

        //因为类的字节码文件对象只有一个,所有都是true
        System.out.println(class1 == class2);
        System.out.println(class2 == class3);
        System.out.println(class3 == class4);
        /**
         * 总结:
         *    1.方式一：通过类名称获得，不执行静态块
         *    2.方式二：通过对象获得，执行静态块
         *    3.方式三：通过包名 + 类名获得，执行静态块
         *    4.方式四：通过包名 + 类名获得，不执行静态块
         *    5.方式一和方式二，耦合性高，方式三和方式四耦合性低
         */
    }
}
```

3. 操作构造器

代码块

```
 package com.powernode.reflection05;

import java.lang.reflect.Constructor;
import java.lang.reflect.Modifier;

class Teacher{
    private String name;
    private int age;

```

```
10      public Teacher() {
11          System.out.println("Teacher.Teacher");
12      }
13
14      protected Teacher(String name) {
15          this.name = name;
16          System.out.println("name = " + name);
17      }
18
19      private Teacher(String name, int age) {
20          this.name = name;
21          this.age = age;
22          System.out.println("name = " + name + ", age = " + age);
23      }
24  }
25  public class Test {
26      public static void main(String[] args) throws Exception {
27          /**
28           * - 构造器包含的元素
29           *    1.修饰符
30           *    2.名称
31           *    3.参数类型
32           *    4.调用构造器
33           */
34          //1.获得Class文件对象
35          Class<Teacher> teacherClass = Teacher.class;
36          //2.用于获取该类所有【public】构造方法
37          //Constructor<?>[] constructors = teacherClass.getConstructors();
38          //3.用于获取该类【所有的】构造方法
39          Constructor<?>[] declaredConstructors =
    teacherClass.getDeclaredConstructors();
40          for (Constructor<?> constructor : declaredConstructors) {
41              System.out.println("修饰符: " +
    Modifier.toString(constructor.getModifiers()));
42              //System.out.println("构造器名称: " +
    constructor.getName());com.powernode.reflection05.Teacher
43              System.out.println("构造器名称: " +
    constructor.getDeclaringClass().getSimpleName());//Teacher
44              //4.获得参数列表
45              Class<?>[] parameterTypes = constructor.getParameterTypes();
46              for (Class<?> parameterType : parameterTypes) {
47                 // System.out.println("参数类型: " + parameterType.getName());
48                  System.out.println("参数类型: " +
    parameterType.getSimpleName());
49              }
50          }
51          System.out.println("---------调用构造器---------");
```

```
        Class<Teacher> teacherClass1 = Teacher.class;
        //1.调用无参构造器
        Teacher teacher = teacherClass1.newInstance();
        //2.调用有参构造器
        //2.1 获得构造器对象
        Constructor<Teacher> declaredConstructor = teacherClass1.getDeclaredConstructor(String.class);
        //2.2 通过构造器对象调用构造器
        Teacher teacher1 = declaredConstructor.newInstance("zs");
        //3.调用私有参构造器
        //3.1获得构造器对象
        Constructor<Teacher> declaredConstructor1 = teacherClass1.getDeclaredConstructor(String.class, int.class);
        //3.2  私有方法，在调用之前要先设置访问权限为（true），否则抛出：IllegalAccessException
        declaredConstructor1.setAccessible(true);
        //3.3 通过构造器对象调用构造器
        Teacher teacher2 = declaredConstructor1.newInstance("zs", 23);

    }
}
```

4. 操作字段

4.1 获得字段信息

代码块

```
package com.powernode.reflection06;

import java.lang.reflect.Field;
import java.lang.reflect.Modifier;

class Teacher{
    public String name = "zs";
    private int age = 23;
    public static String city = "北京";
}
public class Test {
    public static void main(String[] args) throws Exception {
        /**
         * - 字段
         *   1.修饰符
         *   2.数据类型
         *   3.字段名称
```

```
18              *   4.值
19              *     1.取值
20              *     2.赋值
21              */
22             //1.获得字节码文件对象
23             Class<?> aClass = Class.forName("com.powernode.reflection06.Teacher");
24             //2.通过字节码文件对象获得字段数组
25             //Field[] fields = aClass.getFields();//只能获得public修饰的字段
26             Field[] declaredFields = aClass.getDeclaredFields();//获得所有的字段对象
27             for (Field field : declaredFields) {
28                 //3.获得字段信息
29                 int modifiers = field.getModifiers();
30                 System.out.println("修饰符: " + Modifier.toString(modifiers));
31                 System.out.println("数据类型: " + field.getType());
32                 System.out.println("字段名称: " + field.getName());
33                 //4.判断字段是否为私有
34                 if (Modifier.isPrivate(modifiers)) {
35                     //如果为私有，设置访问权限为true
36                     field.setAccessible(true);
37                 }
38                 System.out.println("值: " + field.get(aClass.newInstance()));
39             }
40         }
41     }
```

4.2 字段取值和赋值

代码块

```
1   package com.powernode.reflection07;
2
3   import java.lang.reflect.Field;
4
5   class Teacher{
6       public String name = "zs";
7       private int age = 23;
8       public static String city = "北京";
9   }
10  public class Test {
11      public static void main(String[] args) throws Exception {
12          /**
13           * - 字段
14           *   1.修饰符
15           *   2.数据类型
16           *   3.字段名称
17           *   4.值
```

```
18              *      1.取值
19              *      2.赋值
20              */
21             //1.获得字节码文件对象
22             Class<?> aClass = Class.forName("com.powernode.reflection07.Teacher");
23             System.out.println("==========实例字段---取值和赋值===========");
24             //1.通过字段名称获得字段对象
25             Field name = aClass.getField("name");
26             //2.创建对象
27             Object teacher = aClass.newInstance();
28             //3.取值
29             System.out.println(name.get(teacher));
30             //4.赋值
31             name.set(teacher,"ls");
32             System.out.println(name.get(teacher));
33             //5.私有字段的值
34             //5.1获得私有字段对象
35             Field age = aClass.getDeclaredField("age");
36             //5.2设置访问权限
37             age.setAccessible(true);
38             //5.3取值
39             System.out.println(age.get(teacher));
40             //5.4赋值（修改值）
41             age.set(teacher,33);
42             System.out.println(age.get(teacher));
43             System.out.println("==========静态字段---取值和赋值===========");
44             //1.获得静态字段对象
45             Field city = aClass.getField("city");
46             //2.获得值
47             System.out.println(city.get(null));
48             //3.赋值
49             city.set(null,"南京");
50             System.out.println(city.get(null));
51
52         }
53     }
```

5. 操作方法

5.1 获得方法信息

代码块

```
1   package com.powernode.reflection08;
2
```

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

class Teacher{
    public void method01(){
        System.out.println("Teacher.method01");
    }
    public int method02(int x,int y){
        System.out.println("Teacher.method02(int x,int y)");
        return x + y;
    }
    private void method03(String x,int y){
        System.out.println("Teacher.method03(String x,int y)");
    }
    public static void method04(){
        System.out.println("Teacher.method04");
    }
}
public class Test {
    public static void main(String[] args) {
        /**
         * - 方法
         *   1.修饰符
         *   2.返回类型
         *   3.方法名称
         *   4.形参列表
         *   5.方法调用
         */
        //1.获得字节码文件对象
        Class<Teacher> teacherClass = Teacher.class;
        /*Method[] methods = teacherClass.getMethods();//public 修饰的方法*/
        Method[] declaredMethods = teacherClass.getDeclaredMethods();
        for (Method method : declaredMethods) {
            System.out.println("修饰符:" +
Modifier.toString(method.getModifiers()));
            System.out.println("返回类型:" +
method.getReturnType().getSimpleName());
            System.out.print("方法名称:" + method.getName());
            Class<?>[] parameterTypes = method.getParameterTypes();
            System.out.print(parameterTypes.length == 0?"()\n":"(");
            for (int i = 0; i < parameterTypes.length; i++) {
                Class<?> parameterType = parameterTypes[i];
                if (i == parameterTypes.length -1) {
                    System.out.println(parameterType.getSimpleName() + ")");
                }else{
                    System.out.print(parameterType.getSimpleName() + ",");
```

```
                }

            }
        }
    }
}
```

5.2 方法调用

代码块

```
package com.powernode.reflection09;

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

class Teacher{
    public void method01(){
        System.out.println("Teacher.method01");
    }
    public int method02(int x,int y){
        System.out.println("Teacher.method02(int x,int y)");
        return x + y;
    }
    private void method03(String x,int y){
        System.out.println("Teacher.method03(String x,int y)");
    }
    public static void method04(){
        System.out.println("Teacher.method04");
    }
}
public class Test {
    public static void main(String[] args) throws Exception {
        /**
         * - 方法
         *   1.修饰符
         *   2.返回类型
         *   3.方法名称
         *   4.形参列表
         *   5.方法调用
         */
        System.out.println("===========实例方法调用=================");
        //1.获得字节码文件对象
        Class<Teacher> teacherClass = Teacher.class;
        //2.通过字节码文件对象拿到方法对象
        Method method01 = teacherClass.getMethod("method01");
```

```
        //3.创建对象
        Teacher teacher = teacherClass.newInstance();
        //4.调用方法(无参方法)
        method01.invoke(teacher);
        //5.调用有参方法
        Method method02 = teacherClass.getMethod("method02", int.class,
int.class);
        int sum = (int) method02.invoke(teacher, 10, 20);
        System.out.println("sum = " + sum);
        //6.私有方法调用
        //6.1获得私有方法对象
        Method method03 = teacherClass.getDeclaredMethod("method03",
String.class, int.class);
        //6.2设置访问权限
        method03.setAccessible(true);
        method03.invoke(teacher, "zs", 20);
        System.out.println("===========静态方法调用=================");
        Method method04 = teacherClass.getMethod("method04");
        method04.invoke(null);
    }
}
```

6. 获得类的继承和实现关系

代码块

```
package com.powernode.reflection10;
interface Itf01{}
interface Itf02{}
class Person{}
class Teacher extends Person implements Itf01,Itf02{}
public class Test {
    public static void main(String[] args) {
        //1.拿到字节码文件对象
        Class<Teacher> teacherClass = Teacher.class;
        //2.获得父类
        Class<? super Teacher> superclass = teacherClass.getSuperclass();
        System.out.println(superclass.getName());
        System.out.println(superclass.getSimpleName());
        //3.获得接口
        Class<?>[] interfaces = teacherClass.getInterfaces();
        for (Class<?> anInterface : interfaces) {
            System.out.println(anInterface.getSimpleName());
        }
        //4.获得包名
        System.out.println(teacherClass.getPackageName());
```

```
        System.out.println(teacherClass.getPackage().getName());

    }
}
```

7. 项目案例

1. config.properties

代码块

```
# 数据库连接配置
db.url=jdbc:mysql://localhost:3306
db.uname=root
db.pwd=123
# 配置当前使用的数据库
db.type=com.powernode.reflection11.MySQLDriver
#db.type=com.powernode.reflection11.OracleDriver
```

代码块

```
package com.powernode.reflection11;

public class Test {
    public static void main(String[] args) throws ClassNotFoundException {
        /**
         * 1.应用案例:
         *   1.服务器启动后，通过用户名和密码自动连接数据库
         *   2.支持数据库的切换
         *   3.如上操作，不能修改代码，只允许在配置文件中修改。
         * 2.main方法运行相当于服务器启动
         */
        Class.forName("com.powernode.reflection11.DriverManager");
    }
}
```

代码块

```
package com.powernode.reflection11;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
```

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

public class DriverManager {
    static {
        String[] nameAndPwdByConfigFile = AnalysisConfigFile.getNameAndPwdByConfigFile();
        String name = nameAndPwdByConfigFile[0];
        String pwd = nameAndPwdByConfigFile[1];

        //拿到配置文件中的数据类
        String dbType = AnalysisConfigFile.getDBType();
        try {
            Class<?> aClass = Class.forName(dbType);
            Object o = aClass.newInstance();
            Method method = aClass.getDeclaredMethod("getConnection", String.class, String.class);
            method.invoke(o, name, pwd);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        } catch (InstantiationException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        } catch (NoSuchMethodException e) {
            throw new RuntimeException(e);
        } catch (InvocationTargetException e) {
            throw new RuntimeException(e);
        }


    }
}
```

代码块

```
package com.powernode.reflection11;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

//解析配置文件
public class AnalysisConfigFile {
```

```
    private static String path =
"./day21/src/com/powernode/reflection11/config.properties";
    /**
     * 获得用户名和密码
     * @return
     */
    public static String [] getNameAndPwdByConfigFile(){

        try (
                //1.创建文件读取对象，并指定需要读取的文件
                FileInputStream fileInputStream = new FileInputStream(path);
        ){
            //2.创建Propetries对象
            Properties properties = new Properties();
            properties.load(fileInputStream);
            String name = properties.getProperty("db.uname");
            String pwd = properties.getProperty("db.pwd");
            String [] str = {name,pwd};
            return str;
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    public static String getDBType() {

        try (
                FileInputStream fileInputStream = new FileInputStream(path);
        ){
            Properties properties = new Properties();
            properties.load(fileInputStream);
            return properties.getProperty("db.type");
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

代码块

```
package com.powernode.reflection11;
```

```
2
3  public class MySQLDriver {
4      public boolean getConnection(String name,String pwd){
5          //如果配置文件中的用户名和密码与数据库中的用户名和密码匹配上就可以登录成功
6          if (name.equals("root") && pwd.equals("123")) {
7              System.out.println("MySQL登录成功! ");
8              return true;
9          }
10         System.out.println("MySQL登录失败! ");
11         return false;
12     }
13 }
```

代码块

```
1  package com.powernode.reflection11;
2
3  public class OracleDriver {
4      public boolean getConnection(String name,String pwd){
5          //如果配置文件中的用户名和密码与数据库中的用户名和密码匹配上就可以登录成功
6          if (name.equals("root") && pwd.equals("123")) {
7              System.out.println("Oracle登录成功! ");
8              return true;
9          }
10         System.out.println("Oracle登录失败! ");
11         return false;
12     }
13 }
```

8. 总结

1. I/O流：字节流和字符流的备份
2. 反射：最好都敲一遍，项目案例选择敲