1. 反射

1.1 反射的概述：

专业的解释（了解一下）：

是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；

对于任意一个对象，都能够调用它的任意属性和方法；

这种动态获取信息以及动态调用对象方法的功能称为Java语言的反射机制。

通俗的理解：（掌握）

- 利用**反射**创建的对象**可以无视修饰符**调用类里面的内容
- 可以跟**配置文件结合起来使用**，把要创建的对象信息和方法写在配置文件中。

  读取到什么类，就创建什么类的对象

  读取到什么方法，就调用什么方法

  此时当需求变更的时候不需要修改代码，只要修改配置文件即可。

1.2 学习反射到底学什么?

反射都是从class字节码文件中获取的内容。

- 如何获取class字节码文件的对象
- 利用反射如何获取构造方法（创建对象）
- 利用反射如何获取成员变量（赋值，获取值）

- 利用反射如何获取成员方法（运行）

1.3 获取字节码文件对象的三种方式

- Class这个类里面的静态方法forName（“全类名”） **（最常用）**
- 通过class属性获取
- 通过对象获取字节码文件对象

代码示例：

```
//1.Class这个类里面的静态方法forName
//Class.forName("类的全类名"):  全类名 = 包名 + 类名
Class clazz1 = Class.forName("com.itheima.reflectdemo.Student");
//源代码阶段获取 --- 先把Student加载到内存中，再获取字节码文件的对象
//clazz 就表示Student这个类的字节码文件对象。
//就是当Student.class这个文件加载到内存之后，产生的字节码文件对象


//2.通过class属性获取
//类名.class
Class clazz2 = Student.class;

//因为class文件在硬盘中是唯一的，所以，当这个文件加载到内存之后产生的对象也是唯一的
System.out.println(clazz1 == clazz2);//true


//3.通过Student对象获取字节码文件对象
Student s = new Student();
Class clazz3 = s.getClass();
```

```
System.out.println(clazz1 == clazz2);//true
System.out.println(clazz2 == clazz3);//true
```

1.4 字节码文件和字节码文件对象

java文件：就是我们自己编写的java代码。

字节码文件：就是通过java文件编译之后的class文件（是在硬盘上真实存在的，用眼睛能看到的）

字节码文件对象：当class文件加载到内存之后，虚拟机自动创建出来的对象。

这个对象里面至少包含了：构造方法，成员变量，成员方法。

而我们的反射获取的是什么？字节码文件对象，这个对象在内存中是唯一的。

1.5 获取构造方法

规则：

get表示获取

Declared表示私有

最后的s表示所有，复数形式

如果当前获取到的是私有的，必须要临时修改访问权限，否则无法使用

方法名	说明

方法名	说明
Constructor<?>[] getConstructors()	获得所有的构造（只能public修饰)
Constructor<?>[] getDeclaredConstructors()	获得所有的构造（包含private修饰)
Constructor getConstructor(Class<?>... parameterTypes)	获取指定构造（只能public修饰)
Constructor getDeclaredConstructor(Class<?>... parameterTypes)	获取指定构造（包含private修饰)

代码示例：

```
public class ReflectDemo2 {
    public static void main(String[] args) throws ClassNotFoundException, NoSuchMethodException {
        //1.获得整体（class字节码文件对象）
        Class clazz = Class.forName("com.itheima.reflectdemo.Student");


        //2.获取构造方法对象
        //获取所有构造方法（public）
        Constructor[] constructors1 = clazz.getConstructors();
        for (Constructor constructor : constructors1) {
            System.out.println(constructor);
        }

```

```
14
 System.out.println("======================");
15
16          //获取所有构造（带私有的）
17          Constructor[] constructors2 =
clazz.getDeclaredConstructors();
18
19          for (Constructor constructor :
constructors2) {
20              System.out.println(constructor);
21          }
22
 System.out.println("======================");
23
24          //获取指定的空参构造
25          Constructor con1 = clazz.getConstructor();
26          System.out.println(con1);
27
28          Constructor con2 =
clazz.getConstructor(String.class,int.class);
29          System.out.println(con2);
30
31
 System.out.println("======================");
32          //获取指定的构造(所有构造都可以获取到，包括public
包括private)
33          Constructor con3 =
clazz.getDeclaredConstructor();
34          System.out.println(con3);
35          //了解 System.out.println(con3 == con1);
36          //每一次获取构造方法对象的时候，都会新new一个。
37
38          Constructor con4 =
clazz.getDeclaredConstructor(String.class);
```

```
39          System.out.println(con4);
40      }
41  }
```

1.6 获取构造方法并创建对象

涉及到的方法：newInstance

代码示例：

```
1  //首先要有一个javabean类
2  public class Student {
3      private String name;
4
5      private int age;
6
7
8      public Student() {
9
10     }
11
12     public Student(String name) {
13         this.name = name;
14     }
15
16     private Student(String name, int age) {
17         this.name = name;
18         this.age = age;
19     }
20
21
22     /**
23      * 获取
24      * @return name
```

```
25          */
26         public String getName() {
27             return name;
28         }
29 
30         /**
31          * 设置
32          * @param name
33          */
34         public void setName(String name) {
35             this.name = name;
36         }
37 
38         /**
39          * 获取
40          * @return age
41          */
42         public int getAge() {
43             return age;
44         }
45 
46         /**
47          * 设置
48          * @param age
49          */
50         public void setAge(int age) {
51             this.age = age;
52         }
53 
54         public String toString() {
55             return "Student{name = " + name + ", age =
    " + age + "}";
56         }
57 }
```

```
58
59
60
61 //测试类中的代码：
62 //需求1：
63 //获取空参，并创建对象
64
65 //1.获取整体的字节码文件对象
66 Class clazz =
   Class.forName("com.itheima.a02reflectdemo1.Student"
   );
67 //2.获取空参的构造方法
68 Constructor con = clazz.getConstructor();
69 //3.利用空参构造方法创建对象
70 Student stu = (Student) con.newInstance();
71 System.out.println(stu);
72
73
74 System.out.println("==============================
   ==============");
75
76
77 //测试类中的代码：
78 //需求2：
79 //获取带参构造，并创建对象
80 //1.获取整体的字节码文件对象
81 Class clazz =
   Class.forName("com.itheima.a02reflectdemo1.Student"
   );
82 //2.获取有参构造方法
83 Constructor con =
   clazz.getDeclaredConstructor(String.class,
   int.class);
84 //3.临时修改构造方法的访问权限（暴力反射）
```

```
85 con.setAccessible(true);
86 //4.直接创建对象
87 Student stu = (Student) con.newInstance("zhangsan",
   23);
88 System.out.println(stu);
```

1.7 获取成员变量

规则：

get表示获取

Declared表示私有

最后的s表示所有，复数形式

如果当前获取到的是私有的，必须要临时修改访问权限，否则无法使用

方法名：

方法名	说明
Field[] getFields()	返回所有成员变量对象的数组（只能拿public的）
Field[] getDeclaredFields()	返回所有成员变量对象的数组，存在就能拿到
Field getField(String name)	返回单个成员变量对象（只能拿public的）
Field getDeclaredField(String name)	返回单个成员变量对象，存在就能拿到

代码示例：

```
public class ReflectDemo4 {
    public static void main(String[] args) throws ClassNotFoundException, NoSuchFieldException {
        //获取成员变量对象

        //1.获取class对象
        Class clazz = Class.forName("com.itheima.reflectdemo.Student");

        //2.获取成员变量的对象（Field对象)只能获取public修饰的
        Field[] fields1 = clazz.getFields();
        for (Field field : fields1) {
            System.out.println(field);
        }


 System.out.println("============================");

        //获取成员变量的对象（public + private）
        Field[] fields2 = clazz.getDeclaredFields();
        for (Field field : fields2) {
            System.out.println(field);
        }


 System.out.println("============================");
        //获得单个成员变量对象
        //如果获取的属性是不存在的，那么会报异常
```

```
25          //Field field3 = clazz.getField("aaa");
26
 //System.out.println(field3);//NoSuchFieldExcepti
on
27
28          Field field4 = clazz.getField("gender");
29          System.out.println(field4);
30
31
 System.out.println("============================
==");
32          //获取单个成员变量（私有）
33          Field field5 =
clazz.getDeclaredField("name");
34          System.out.println(field5);
35
36      }
37  }
38
39
40
41  public class Student {
42      private String name;
43
44      private int age;
45
46      public String gender;
47
48      public String address;
49
50
51      public Student() {
52      }
53
```

```
54      public Student(String name, int age, String
address) {
55          this.name = name;
56          this.age = age;
57          this.address = address;
58      }
59
60
61      public Student(String name, int age, String
gender, String address) {
62          this.name = name;
63          this.age = age;
64          this.gender = gender;
65          this.address = address;
66      }
67
68      /**
69       * 获取
70       * @return name
71       */
72      public String getName() {
73          return name;
74      }
75
76      /**
77       * 设置
78       * @param name
79       */
80      public void setName(String name) {
81          this.name = name;
82      }
83
84      /**
85       * 获取
```

```
86          * @return age
87          */
88         public int getAge() {
89             return age;
90         }
91 
92         /**
93          * 设置
94          * @param age
95          */
96         public void setAge(int age) {
97             this.age = age;
98         }
99 
100        /**
101         * 获取
102         * @return gender
103         */
104        public String getGender() {
105            return gender;
106        }
107 
108        /**
109         * 设置
110         * @param gender
111         */
112        public void setGender(String gender) {
113            this.gender = gender;
114        }
115 
116        /**
117         * 获取
118         * @return address
119         */
```

```
120     public String getAddress() {
121         return address;
122     }
123 
124     /**
125      * 设置
126      * @param address
127      */
128     public void setAddress(String address) {
129         this.address = address;
130     }
131 
132     public String toString() {
133         return "Student{name = " + name + ", age = " + age + ", gender = " + gender + ", address = " + address + "}";
134     }
135 }
136 
```

1.8 获取成员变量并获取值和修改值

方法	说明
void set(Object obj, Object value)	赋值
Object get(Object obj)	获取值

代码示例：

```
1 public class ReflectDemo5 {
2     public static void main(String[] args) throws ClassNotFoundException, NoSuchFieldException, IllegalAccessException {
```

```
        Student s = new Student("zhangsan",23,"广州");
        Student ss = new Student("lisi",24,"北京");

        //需求:
        //利用反射获取成员变量并获取值和修改值

        //1.获取class对象
        Class clazz = Class.forName("com.itheima.reflectdemo.Student");

        //2.获取name成员变量
        //field就表示name这个属性的对象
        Field field = clazz.getDeclaredField("name");
        //临时修饰他的访问权限
        field.setAccessible(true);

        //3.设置(修改)name的值
        //参数一：表示要修改哪个对象的name?
        //参数二：表示要修改为多少?
        field.set(s,"wangwu");

        //3.获取name的值
        //表示我要获取这个对象的name的值
        String result = (String)field.get(s);

        //4.打印结果
        System.out.println(result);

        System.out.println(s);
        System.out.println(ss);

    }
```

```
}


public class Student {
    private String name;
    private int age;
    public String gender;
    public String address;


    public Student() {
    }

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


    public Student(String name, int age, String gender, String address) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.address = address;
    }

    /**
     * 获取
     * @return name
     */
    public String getName() {
```

```
66          return name;
67      }
68
69      /**
70       * 设置
71       * @param name
72       */
73      public void setName(String name) {
74          this.name = name;
75      }
76
77      /**
78       * 获取
79       * @return age
80       */
81      public int getAge() {
82          return age;
83      }
84
85      /**
86       * 设置
87       * @param age
88       */
89      public void setAge(int age) {
90          this.age = age;
91      }
92
93      /**
94       * 获取
95       * @return gender
96       */
97      public String getGender() {
98          return gender;
99      }
```

```
100
101     /**
102      * 设置
103      * @param gender
104      */
105     public void setGender(String gender) {
106         this.gender = gender;
107     }
108
109     /**
110      * 获取
111      * @return address
112      */
113     public String getAddress() {
114         return address;
115     }
116
117     /**
118      * 设置
119      * @param address
120      */
121     public void setAddress(String address) {
122         this.address = address;
123     }
124
125     public String toString() {
126         return "Student{name = " + name + ", age =
" + age + ", gender = " + gender + ", address = "
+ address + "}";
127     }
128 }
129
```

1.9 获取成员方法

规则：

get表示获取

Declared表示私有

最后的s表示所有，复数形式

如果当前获取到的是私有的，必须要临时修改访问权限，否则无法使用

方法名	说明
Method[] getMethods()	返回所有成员方法对象的数组（只能拿public的）
Method[] getDeclaredMethods()	返回所有成员方法对象的数组，存在就能拿到
Method getMethod(String name, Class<?>... parameterTypes)	返回单个成员方法对象（只能拿public的）
Method getDeclaredMethod(String name, Class<?>... parameterTypes)	返回单个成员方法对象，存在就能拿到

代码示例：

```
1 public class ReflectDemo6 {
2     public static void main(String[] args) throws
ClassNotFoundException, NoSuchMethodException {
3         //1.获取class对象
4         Class<?> clazz =
Class.forName("com.itheima.reflectdemo.Student");
5
```

```
6
7           //2.获取方法
8           //getMethods可以获取父类中public修饰的方法
9           Method[] methods1 = clazz.getMethods();
10          for (Method method : methods1) {
11              System.out.println(method);
12          }
13
14
 System.out.println("============================");
15          //获取所有的方法（包含私有）
16          //但是只能获取自己类中的方法
17          Method[] methods2 =
clazz.getDeclaredMethods();
18          for (Method method : methods2) {
19              System.out.println(method);
20          }
21
22
 System.out.println("============================");
23          //获取指定的方法（空参）
24          Method method3 = clazz.getMethod("sleep");
25          System.out.println(method3);
26
27          Method method4 =
clazz.getMethod("eat",String.class);
28          System.out.println(method4);
29
30          //获取指定的私有方法
31          Method method5 =
clazz.getDeclaredMethod("playGame");
32          System.out.println(method5);
33      }
34  }
```

```
35
```

1.10 获取成员方法并运行

方法

Object invoke(Object obj, Object... args)：运行方法

参数一：用obj对象调用该方法

参数二：调用方法的传递的参数（如果没有就不写）

返回值：方法的返回值（如果没有就不写）

代码示例：

```
1  package com.itheima.a02reflectdemo1;
2
3  import
   java.lang.reflect.InvocationTargetException;
4  import java.lang.reflect.Method;
5
6  public class ReflectDemo6 {
7      public static void main(String[] args) throws
   ClassNotFoundException, NoSuchMethodException,
   InvocationTargetException, IllegalAccessException
   {
8          //1.获取字节码文件对象
9          Class clazz =
   Class.forName("com.itheima.a02reflectdemo1.Student
   ");
10
11         //2.获取一个对象
12         //需要用这个对象去调用方法
13         Student s = new Student();
```

```
14
15          //3.获取一个指定的方法
16          //参数一：方法名
17          //参数二：参数列表，如果没有可以不写
18          Method eatMethod =
clazz.getMethod("eat",String.class);
19
20          //运行
21          //参数一：表示方法的调用对象
22          //参数二：方法在运行时需要的实际参数
23          //注意点：如果方法有返回值，那么需要接收invoke的
结果
24          //如果方法没有返回值，则不需要接收
25          String result = (String)
eatMethod.invoke(s, "重庆小面");
26          System.out.println(result);
27
28      }
29  }
30
31
32
33  public class Student {
34      private String name;
35      private int age;
36      public String gender;
37      public String address;
38
39
40      public Student() {
41
42      }
43
44      public Student(String name) {
```

```
45          this.name = name;
46      }
47
48      private Student(String name, int age) {
49          this.name = name;
50          this.age = age;
51      }
52
53      /**
54       * 获取
55       * @return name
56       */
57      public String getName() {
58          return name;
59      }
60
61      /**
62       * 设置
63       * @param name
64       */
65      public void setName(String name) {
66          this.name = name;
67      }
68
69      /**
70       * 获取
71       * @return age
72       */
73      public int getAge() {
74          return age;
75      }
76
77      /**
78       * 设置
```

```
79          * @param age
80          */
81         public void setAge(int age) {
82             this.age = age;
83         }
84 
85         public String toString() {
86             return "Student{name = " + name + ", age =
" + age + "}";
87         }
88 
89         private void study(){
90             System.out.println("学生在学习");
91         }
92 
93         private void sleep(){
94             System.out.println("学生在睡觉");
95         }
96 
97         public String eat(String something){
98             System.out.println("学生在吃" + something);
99             return "学生已经吃完了，非常happy";
100         }
101 }
```

面试题：

你觉得反射好不好？好，有两个方向

第一个方向：无视修饰符访问类中的内容。但是这种操作在开发中一般不用，都是框架底层来用的。

第二个方向：反射可以跟配置文件结合起来使用，动态的创建对象，动态的调用方法。

1.11 练习泛型擦除（掌握概念，了解代码）

理解：（掌握）

集合中的泛型只在java文件中存在，当编译成class文件之后，就没有泛型了。

代码示例：（了解）

```
package com.itheima.reflectdemo;

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

public class ReflectDemo8 {
    public static void main(String[] args) throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
        //1.创建集合对象
        ArrayList<Integer> list = new ArrayList<>();
        list.add(123);
//        list.add("aaa");

        //2.利用反射运行add方法去添加字符串
        //因为反射使用的是class字节码文件

        //获取class对象
        Class clazz = list.getClass();

        //获取add方法对象
```

```
        Method method = clazz.getMethod("add", Object.class);

        //运行方法
        method.invoke(list,"aaa");

        //打印集合
        System.out.println(list);
    }
}

```

1.12 练习：修改字符串的内容（掌握概念，了解代码）

在这个练习中，我需要你掌握的是字符串不能修改的真正原因。

字符串，在底层是一个byte类型的字节数组，名字叫做value

```
private final byte[] value;
```

真正不能被修改的原因：final和private

final修饰value表示value记录的地址值不能修改。

private修饰value而且没有对外提供getvalue和setvalue的方法。所以，在外界不能获取或修改value记录的地址值。

如果要强行修改可以用反射：

代码示例：（了解）

```
String s = "abc";
String ss = "abc";
// private final byte[] value= {97,98,99};
```

```
// 没有对外提供getvalue和setvalue的方法，不能修改value记录的地址值
// 如果我们利用反射获取了value的地址值。
// 也是可以修改的，final修饰的value
// 真正不可变的value数组的地址值，里面的内容利用反射还是可以修改的，比较危险

//1.获取class对象
Class clazz = s.getClass();

//2.获取value成员变量（private）
Field field = clazz.getDeclaredField("value");
//但是这种操作非常危险
//JDK高版本已经屏蔽了这种操作，低版本还是可以的
//临时修改权限
field.setAccessible(true);

//3.获取value记录的地址值
byte[] bytes = (byte[]) field.get(s);
bytes[0] = 100;

System.out.println(s);//dbc
System.out.println(ss);//dbc
```

1.13 练习，反射和配置文件结合动态获取的练习（重点）

需求: 利用反射根据文件中的不同类名和方法名，创建不同的对象并调用方法。

分析:

①通过Properties加载配置文件

②得到类名和方法名

③通过类名反射得到Class对象

④通过Class对象创建一个对象

⑤通过Class对象得到方法

⑥调用方法

代码示例：

```
public class ReflectDemo9 {
    public static void main(String[] args) throws IOException, ClassNotFoundException, NoSuchMethodException, InvocationTargetException, InstantiationException, IllegalAccessException {
        //1.读取配置文件的信息
        Properties prop = new Properties();
        FileInputStream fis = new FileInputStream("day14-code\\prop.properties");
        prop.load(fis);
        fis.close();
        System.out.println(prop);

        String classname = prop.get("classname") + "";
        String methodname = prop.get("methodname") + "";

        //2.获取字节码文件对象
        Class clazz = Class.forName(classname);

        //3.要先创建这个类的对象
```

```
        Constructor con = clazz.getDeclaredConstructor();
        con.setAccessible(true);
        Object o = con.newInstance();
        System.out.println(o);

        //4.获取方法的对象
        Method method = clazz.getDeclaredMethod(methodname);
        method.setAccessible(true);

        //5.运行方法
        method.invoke(o);


    }
}

配置文件中的信息：
classname=com.itheima.a02reflectdemo1.Student
methodname=sleep
```

1.14 利用发射保存对象中的信息（重点）

```
public class MyReflectDemo {
    public static void main(String[] args) throws IllegalAccessException, IOException {
    /*
        对于任意一个对象，都可以把对象所有的字段名和值，保存到文件中去
    */
```

```
        Student s = new Student("小A",23,'女',167.5,"睡觉");
        Teacher t = new Teacher("播妞",10000);
        saveObject(s);
    }

    //把对象里面所有的成员变量名和值保存到本地文件中
    public static void saveObject(Object obj) throws IllegalAccessException, IOException {
        //1.获取字节码文件的对象
        Class clazz = obj.getClass();
        //2. 创建IO流
        BufferedWriter bw = new BufferedWriter(new FileWriter("myreflect\\a.txt"));
        //3. 获取所有的成员变量
        Field[] fields = clazz.getDeclaredFields();
        for (Field field : fields) {
            field.setAccessible(true);
            //获取成员变量的名字
            String name = field.getName();
            //获取成员变量的值
            Object value = field.get(obj);
            //写出数据
            bw.write(name + "=" + value);
            bw.newLine();
        }

        bw.close();

    }
}
```

```
public class Student {
```

```
    private String name;
    private int age;
    private char gender;
    private double height;
    private String hobby;

    public Student() {
    }

    public Student(String name, int age, char gender, double height, String hobby) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.height = height;
        this.hobby = hobby;
    }

    /**
     * 获取
     * @return name
     */
    public String getName() {
        return name;
    }

    /**
     * 设置
     * @param name
     */
    public void setName(String name) {
        this.name = name;
    }

```

```
35      /**
36       * 获取
37       * @return age
38       */
39      public int getAge() {
40          return age;
41      }
42
43      /**
44       * 设置
45       * @param age
46       */
47      public void setAge(int age) {
48          this.age = age;
49      }
50
51      /**
52       * 获取
53       * @return gender
54       */
55      public char getGender() {
56          return gender;
57      }
58
59      /**
60       * 设置
61       * @param gender
62       */
63      public void setGender(char gender) {
64          this.gender = gender;
65      }
66
67      /**
68       * 获取
```

```
     * @return height
     */
    public double getHeight() {
        return height;
    }

    /**
     * 设置
     * @param height
     */
    public void setHeight(double height) {
        this.height = height;
    }

    /**
     * 获取
     * @return hobby
     */
    public String getHobby() {
        return hobby;
    }

    /**
     * 设置
     * @param hobby
     */
    public void setHobby(String hobby) {
        this.hobby = hobby;
    }

    public String toString() {
        return "Student{name = " + name + ", age = " + age + ", gender = " + gender + ", height = " + height + ", hobby = " + hobby + "}";
```

```
    }
}
```

```
public class Teacher {
    private String name;
    private double salary;

    public Teacher() {
    }

    public Teacher(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    /**
     * 获取
     * @return name
     */
    public String getName() {
        return name;
    }

    /**
     * 设置
     * @param name
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * 获取
```

```
     * @return salary
     */
    public double getSalary() {
        return salary;
    }

    /**
     * 设置
     * @param salary
     */
    public void setSalary(double salary) {
        this.salary = salary;
    }

    public String toString() {
        return "Teacher{name = " + name + ", salary = " + salary + "}";
    }
}

```

2. 动态代理

2.1 好处：

无侵入式的给方法增强功能

2.2 动态代理三要素：

1，真正干活的对象

2，代理对象

3，利用代理调用方法

切记一点：代理可以增强或者拦截的方法都在接口中，接口需要写在newProxyInstance的第二个参数里。

2.3 代码实现：

```
public class Test {
    public static void main(String[] args) {
    /*
        需求：
            外面的人想要大明星唱一首歌
             1. 获取代理的对象
                代理对象 = ProxyUtil.createProxy(大明星的对象);
             2. 再调用代理的唱歌方法
                代理对象.唱歌的方法("只因你太美");
     */
        //1. 获取代理的对象
        BigStar bigStar = new BigStar("鸡哥");
        Star proxy = ProxyUtil.createProxy(bigStar);

        //2. 调用唱歌的方法
        String result = proxy.sing("只因你太美");
        System.out.println(result);
    }
}
```

```
/*
*
* 类的作用：
*        创建一个代理
```

```
5  *
6  * */
7  public class ProxyUtil {
8      /*
9      *
10     * 方法的作用：
11     *         给一个明星的对象，创建一个代理
12     *
13     *  形参：
14     *         被代理的明星对象
15     *
16     *  返回值：
17     *         给明星创建的代理
18     *
19     *
20     *
21     * 需求：
22     *   外面的人想要大明星唱一首歌
23     *   1. 获取代理的对象
24     *       代理对象 = ProxyUtil.createProxy(大明星的对
   象);
25     *   2. 再调用代理的唱歌方法
26     *       代理对象.唱歌的方法("只因你太美");
27     * */
28     public static Star createProxy(BigStar bigStar)
   {
29         /* java.lang.reflect.Proxy类：提供了为对象产生代
   理对象的方法：
30
31          public static Object
   newProxyInstance(ClassLoader loader, Class<?>[]
   interfaces, InvocationHandler h)
32         参数一：用于指定用哪个类加载器，去加载生成的代理类
```

```
33          参数二：指定接口，这些接口用于指定生成的代理长什么，也就是有哪些方法
34          参数三：用来指定生成的代理对象要干什么事情*/
35          Star star = (Star) Proxy.newProxyInstance(
36                  ProxyUtil.class.getClassLoader(),//参数一：用于指定用哪个类加载器，去加载生成的代理类
37                  new Class[]{Star.class},//参数二：指定接口，这些接口用于指定生成的代理长什么，也就是有哪些方法
38                  //参数三：用来指定生成的代理对象要干什么事情
39                  new InvocationHandler() {
40                      @Override
41                      public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
42                          /*
43                          * 参数一：代理的对象
44                          * 参数二：要运行的方法 sing
45                          * 参数三：调用sing方法时，传递的实参
46                          * */
47
 if("sing".equals(method.getName())){
48                              System.out.println("准备话筒，收钱");
49                          }else if("dance".equals(method.getName())){
50                              System.out.println("准备场地，收钱");
51                          }
52                          //去找大明星开始唱歌或者跳舞
53                          //代码的表现形式：调用大明星里面唱歌或者跳舞的方法
```

```
                        return method.invoke(bigStar,args);
                    }
                }
        );
        return star;
    }
}
```

```
public interface Star {
    //我们可以把所有想要被代理的方法定义在接口当中
    //唱歌
    public abstract String sing(String name);
    //跳舞
    public abstract void dance();
}
```

```
public class BigStar implements Star {
    private String name;


    public BigStar() {
    }

    public BigStar(String name) {
        this.name = name;
    }

    //唱歌
    @Override
    public String sing(String name){
        System.out.println(this.name + "正在唱" + name);
```

```
16          return "谢谢";
17      }
18
19      //跳舞
20      @Override
21      public void dance(){
22          System.out.println(this.name + "正在跳舞");
23      }
24
25      /**
26       * 获取
27       * @return name
28       */
29      public String getName() {
30          return name;
31      }
32
33      /**
34       * 设置
35       * @param name
36       */
37      public void setName(String name) {
38          this.name = name;
39      }
40
41      public String toString() {
42          return "BigStar{name = " + name + "}";
43      }
44  }
45
```

2.4 额外扩展

动态代理，还可以拦截方法

比如：

在这个故事中，经济人作为代理，如果别人让邀请大明星去唱歌，打篮球，经纪人就增强功能。

但是如果别人让大明星去扫厕所，经纪人就要拦截，不会去调用大明星的方法。

```
/*
* 类的作用：
*        创建一个代理
* */
public class ProxyUtil {
    public static Star createProxy(BigStar bigStar)
{
        public static Object
newProxyInstance(ClassLoader loader, Class<?>[]
interfaces, InvocationHandler h)
        Star star = (Star) Proxy.newProxyInstance(
                ProxyUtil.class.getClassLoader(),
                new Class[]{Star.class},
                new InvocationHandler() {
                    @Override
                    public Object invoke(Object
proxy, Method method, Object[] args) throws
Throwable {

 if("cleanWC".equals(method.getName())){
                            System.out.println("拦
截，不调用大明星的方法");
                            return null;
                        }
```

```
                        //如果是其他方法，正常执行
                        return
method.invoke(bigStar,args);
                    }
                }
        );
        return star;
    }
}
```

2.5 动态代理的练习

对add方法进行增强，对remove方法进行拦截，对其他方法不拦截也不增强

```
public class MyProxyDemo1 {
    public static void main(String[] args) {
        //动态代码可以增强也可以拦截
        //1.创建真正干活的人
        ArrayList<String> list = new ArrayList<>();

        //2.创建代理对象
        //参数一：类加载器。当前类
名.class.getClassLoader()
        //                    找到是谁，把当前的类，加载到
内存中了，我再麻烦他帮我干一件事情，把后面的代理类，也加载到内
存

        //参数二：是一个数组，在数组里面写接口的字节码文件对
象。
        //                    如果写了List，那么表示代
理，可以代理List接口里面所有的方法，对这些方法可以增强或者拦
截
```

```
13          //                          但是，一定要写ArrayList真实实现的接口
14          //                          假设在第二个参数中，写了MyInter接口，那么是错误的。
15          //                          因为ArrayList并没有实现这个接口，那么就无法对这个接口里面的方法，进行增强或拦截
16          //参数三：用来创建代理对象的匿名内部类
17          List proxyList = (List) Proxy.newProxyInstance(
18                  //参数一：类加载器
19
                 MyProxyDemo1.class.getClassLoader(),
20                  //参数二：是一个数组，表示代理对象能代理的方法范围
21                  new Class[]{List.class},
22                  //参数三：本质就是代理对象
23                  new InvocationHandler() {
24                      @Override
25                      //invoke方法参数的意义
26                      //参数一：表示代理对象，一般不用（了解）
27                      //参数二：就是方法名，我们可以对方法名进行判断，是增强还是拦截
28                      //参数三：就是下面第三步调用方法时，传递的参数。
29                      //举例1：
30                      //list.add("阿玮好帅");
31                      //此时参数二就是add这个方法名
32                      //此时参数三 args[0] 就是 阿玮好帅
33                      //举例2：
34                      //list.set(1, "aaa");
35                      //此时参数二就是set这个方法名
36                      //此时参数三  args[0] 就是 1 args[1]"aaa"
```

```
37                      public Object invoke(Object
proxy, Method method, Object[] args) throws
Throwable {
38                          //对add方法做一个增强，统计耗时
时间
39                          if
(method.getName().equals("add")) {
40                              long start =
System.currentTimeMillis();
41                              //调用集合的方法，真正的添加
数据
42                              method.invoke(list,
args);
43                              long end =
System.currentTimeMillis();
44                              System.out.println("耗时
时间: " + (end - start));
45                              //需要进行返回，返回值要跟真
正增强或者拦截的方法保持一致
46                              return true;
47                          }else
if(method.getName().equals("remove") && args[0]
instanceof Integer){
48                              System.out.println("拦截
了按照索引删除的方法");
49                              return null;
50                          }else
if(method.getName().equals("remove")){
51                              System.out.println("拦截
了按照对象删除的方法");
52                              return false;
53                          }else{
54                              //如果当前调用的是其他方法，
我们既不增强，也不拦截
```

```
55
 method.invoke(list,args);
56                                  return null;
57                              }
58                          }
59                      }
60          );
61
62          //3.调用方法
63          //如果调用者是list，就好比绕过了第二步的代码，直接添加元素
64          //如果调用者是代理对象，此时代理才能帮我们增强或者拦截
65
66          //每次调用方法的时候，都不会直接操作集合
67          //而是先调用代理里面的invoke，在invoke方法中进行判断，可以增强或者拦截
68          proxyList.add("aaa");
69          proxyList.add("bbb");
70          proxyList.add("ccc");
71          proxyList.add("ddd");
72
73          proxyList.remove(0);
74          proxyList.remove("aaa");
75
76
77          //打印集合
78          System.out.println(list);
79      }
80  }
```