Universe Scientific Publishing
Computer System Networking and Telecommunications
计算机系统网络和电信，2019,1(2)
ISSN:2661-3719 (Print); 2661-3727 (Online)

Java Reflection Mechanism and Its Application in Configurable Programming

Yujian CHEN Chun ZHAO Chenxi LI
School of Computer and Software, Jincheng College, Sichuan University, Chengdu, Sichuan, 611731

Abstract

Reflection is a key feature of Java as a dynamic language. It can dynamically acquire the internal information of classes and generate instances of classes or access members of classes during the running of programs. The high coupling caused by direct calls between classes in Java programming development will seriously affect the extensibility of programs. By introducing interfaces to reduce the coupling between programs and using configurable programming to improve the flexibility of programs, applications can have higher scalability in the face of changing requirements.

Key Words

Java Reflective, Interface-Oriented, Configurable Programming, Extensibility

DOI:10.18686/jsjxt.v1i2.642

Java 反射机制及在可配置式编程中的应用

陈宇坚 赵 春 李晨曦
四川大学锦城学院计算机与软件学院，四川，成都，611731

摘 要

反射是 Java 成为动态语言的关键性质，它可以在程序运行期间动态获取类的内部信息，并生成类的实例或访问类的成员。Java 程序设计开发中类之间的直接调用所带来的高度耦合会严重影响到程序的扩展性。通过引入接口降低程序之间的耦合、采用配置式编程提高程序的灵活性，可以让应用程序在面对需求变化时具备更高的扩展性。

关键字

Java 反射；面向接口；配置式编程；扩展性

1.引言

反射是指程序可以访问，检测和修改自身状态或行为的一种能力，它首先被程序语言的设计领域所采用，并在 Lisp 和面向对象方面取得了成绩。[1]它也是 Java 语言中的典型工具，允许程序员有效地创建可在运行时组装的更灵活的代码。源代码的组件可以相互动态链接，极大地提高了程序开发的效率。通过引入接口和采用配置式编程的方法，使程序在开发中应对需求的变更，降低程序模块间的耦合性，提高程序灵活性和扩展性。

2.反射技术

在程序开发中，使用某一个类时必须知道他的功能和使用方法，并在使用过程中需要对这个类进行实例化，之后再对这个实例化的类对象进行一系列操作。通过 new 方法来实例化对象的方式，称作正射。而反射，可以在运行时加载、探知、使用编译期间完全未知的 classes，[2]即在不知道初始化的类对象情况下，通过一个实例的引用，通过 JVM 逆向解析该类的各类信息，获悉其完整构造（但不包括 methods 定义），并生成其对象实体、或对其 fields 设值、或唤起其 methods。这种机制允许编程人员在对类未知的情况下，获取类相关信息的方式变得更加多样灵活，调用类中相应方法，是 Java 增加其灵活性与动态性的一种机制。[3]

3.反射机制中的类

3.1 java.lang.Class

在整个反射开发模型中，Class 类是 Reflection API 中的核心类，提供了数十多个的 Reflection APIs，[4]它是 Java 反射的来源。只能由系统创建对象，一个类在 JVM 中只有一个 Class 实例，通过 Class 可以完整得到一个类中所有的信息（属性、方法、构造器、内部类、父类、所在包、异常等等）。是所有可以通过对象反射出类的名称。下面简单的列举了常见的三种获取 class 实例的方式

第一种方式：

String str = ”hello word”;

Class cls1=str.getClass();

System.out.println(cls1);

此方法通过运行时类对象获取，但不能被子类所覆写，而且所有类的实例化对象都可以调用

第二种方式：

Class cls2= Java.lang.String.class;

System.out.println(cls2);

此方法通过调用运行时类本身的.class 属性获取，在一些开源框架中会被大量使用如 Hibernate、Mybatis 等

第三种方式：

Class cls3=Class.forName(”Java.lang.String”);

System.out.println(cls3);

此方法通过 Class 提供的静态方法获得，主要用于工厂类或驱动程序加载以设计 jdbc。

虽然这三种方式都可以获取 class 实例，但是在我们实际开发中，相比于第一种方式，第二种方式不需要获取指定操作类的实例化对象，您可以通过类的名称直接获取实例。但是两种方式都存在明显的弊端，即它们明确定义了一个存在的类，并且只能对其进行操作。而第三种方式最大的特点则是灵活，即操作的类可以不存在，只要在程序运行时添加上指定的类即可。

3.2 java.lang.reflect.Constructor

Constructor 是类的构造器，在 Constructor 类里面有一个专门负责实例化对象的方法来传递指定参数

例：Class cls=Class.forName("Reflection.Person");

Constructor<?> cons = cls.getConstructor();

Person p=(Person)cls.newInstance();

System.out.println(p);

通过以上一段代码，类之中必须提供无参构造方法的时候是最简单的，如果是有参构造，那么用户需要处理很多代码逻辑。若要调用指定的更多构造器，则需要使用 DeclaredConstructor 方法

3.3 java.lang.reflect.Method

在 class 方法中，提供有两种普通方法取得信息：

1)， getMethods（）:可以取得一个类之中所有定义的方法，包括自己定义的以及继承而来的方法

2)， getDeclaredMethods（）:可以取得本类之中定义的所有操作方法，与继承无关

Method. invoke（）:在使用该方法时，一定要保证已经存在了本类的实例化对象，最方便的一点是，这个实例化对象可以直接用 Object 替换，可以使用 Class 类反射直接实例化。而后通过 Object 类对象操作，省去了向下转型，提高了代码的质量。

3.4 java.lang.reflect.Field

在 Class 方法中提供了两种取得成员的方法：

1)，getFields():可以得到全部成员，包括继承而来的成员，但是无法取得私有。

2)，getDeclaredFields():可以获取本类的属性

4.反射在可配置式编程中的应用

在学生活动管理系统中，不同类型的活动有不同的宣传方式、举办方式和评奖方式。PartyAcivity类描述了一个晚会类活动，表示活动组织者的Organizer类通过直接对PartyActivity类的访问实现了晚会活动的组织执行。示例代码如下：

```
//晚会活动类
public class EveningParty {
    public void advertise() {
        System.out.println("这是晚会宣传方式");
    }
    public void organize(){
        System.out.println("这是晚会组织方式");
    }
    public void elect() {
        System.out.println("这是晚会评奖方式");
```

```
    }
}
//组织者类
public class Organizer {
    //组织晚会活动
    public void organizeActivity() {
            EvningParty ep = new EvningParty();   //创建活动
            ep.advertise();
            ep.organize();
            ep.elect();
    }
}
```

上述代码通过PartyActivity类与Oranizer类之间的直接交互实现了预期功能。代码简单、易于理解。考虑到需要组织英语竞赛活动的新需求，定义了如下英语竞赛活动类：

```
public class EnglishCompetition{
    public void advertise() {
        System.out.println("这是英语竞赛宣传方式");
    }
    public void organize(){
        System.out.println("这是英语竞赛组织方式");
    }
    public void elect() {
        System.out.println("这是英语竞赛评奖方式");
    }
}
//组织者类
public class Organizer {
    //组织英语竞赛活动
    public void organizeActivity() {
            EnglishCompetition ec = new EnglishCompetition ();   //创建活动
            ec.advertise();
            ec.organize();
            ec.elect();
    }
}
```

从上述代码可以看出，新活动的加入改变了Organizer类。这是因为它们之间的直接访问使得两者高度耦合，从而使得Organizer类不具备可扩展性。举办每一个活动，都会涉及到组织者类的修改，给程序的维护和扩展性带来了巨大的灾难，反复编写重复的代码也会使整个程序过于臃肿。

针对此问题，可以引入接口来隔离彼此的变化对对方的影响，有效降低双方的耦合性。同时可以利用Java反射机制来实现可配置式编程，从而极大提高程序的扩展性。代码示范如下：

首先创建一个活动接口，分别有宣传、举办和评奖三个方法：

```
public interface Activity{
    public void advertise();
    public void organize();
    public void elect();
}
```

不同活动均需实现活动接口：

```
class EnglishCompetition implements Activity{
    public void advertise() {
        System.out.println("这是英语竞赛宣传方式");
    }
    public void organize(){
        System.out.println("这是英语竞赛组织方式");
    }
    public void elect() {
        System.out.println("这是英语竞赛评奖方式");
    }
}
class EveningParty implements Activity{
    public void advertise() {
        System.out.println("这是晚会宣传方式");
    }
    public void organize(){
        System.out.println("这是晚会组织方式");
    }
    public void elect() {
        System.out.println("这是晚会评奖方式");
    }
}
```

将接口和即将开展的活动写入配置文件中

```
//配置文件片段
party=reflect.EveningParty
```

```
competition=reflect.EnglishCompetition
引入反射技术的组织者类：
public class Organizer {
    Property activities = new Properties();
    static{
        …… //读取配置文件数据到activities
    }
    public void organizeActivity(String
activityName){
        //获取具体活动实例
        Activity activity = getActivity(activityName);
        activity.advertise();
        activity.organize();
        activity.elect();
    }
//创建活动实例的工厂方法
public static Activity getActivity(String
activityName) throws Exception{
        String className =
activities.getProperty(activityName);
        //利用Java反射技术创建活动实例
        return
(Activity)Class.forName(className).newInstance();
}
```

这样做的好处是引入反射和配置文件后写出来的代码只需要在配置文件中修改想要办的活动名称，实现不同活动类型的接口，在用户类中的结构不会受到任何影响，用户只需根据自己的需求举办不同的活动。这种方式不仅解除了两个模块间耦合度高的问题，而且在系统以后整体的可维护性和功能扩展性上都有了很大的提升，也更贴近于现实中程序的开发。

然而缺点是引入反射后程序的性能问题一直被人们所诟病。如果在代码量较小的情况下，由于反射的特性导致了在加载字段和方法接入时性能会低于直接代码。因此更适用于逻辑复杂、对灵活性扩展性要求较高的系统框架中。另一个问题是模糊了程序内部发生的事情，允许代码执行不该执行的操作，破坏了Java程序的抽象性，使编码人员不清楚程序内部逻辑，一旦不能正常工作，可能会牵扯到整个系统的运行

5.总结

本文通过传统方法上一个程序中两个子模块的直接访问和交互与引入 Java 反射机制面向接口后采用配置式编程的方法实现两个子模块间相互访问和交互作对比，可以看出：Java 反射在可配置式编程中的应用更符合面向对象的原则而不是面向实现编程，这种方式可以将重复率高或功能近似的不同模块写入到配置文件中，通过反射机制调用类中需要使用到的方法。解除了程序各模块间耦合度高的问题，也提高了程序的可扩展性。充分利用这一特性，可以编写出灵活、低耦合、高重用的组件。[5]合理使用方能够写出更健壮、灵活的代码。

参考文献

[1] 刘凯立. Java 反射技术浅谈[J]. 科技信息. 2010(08)
[2] 王善发,吴道荣. Java 语言的反射机制[J]. 保山学院学报. 2011(05)
[3] 林围强. Java 反射机制的研究[J]. 信息系统工程. 2016(08)
[4] 龙天威,陈亚楠,陈彪. 浅谈 Java 反射机制[J]. 华南金融电脑. 2008(12)
[5] 王开，谭翼，周兰江. JAVA 中反射机制浅析及应用[J]. 计算机教育，2007（1）

作者简介

第一作者：陈宇坚（1997-），男，汉，四川省成都市，本科，四川大学锦城学院，研究方向：大数据技术开发应用。
第二作者（通讯作者）：赵春(1978-)，男，汉，四川成都，副教授，研究方向：互联网应用。
第三作者：李晨曦（1998-7），男，汉，贵州省贵阳市，本科，四川大学锦城学院，研究方向：大数据技术开发