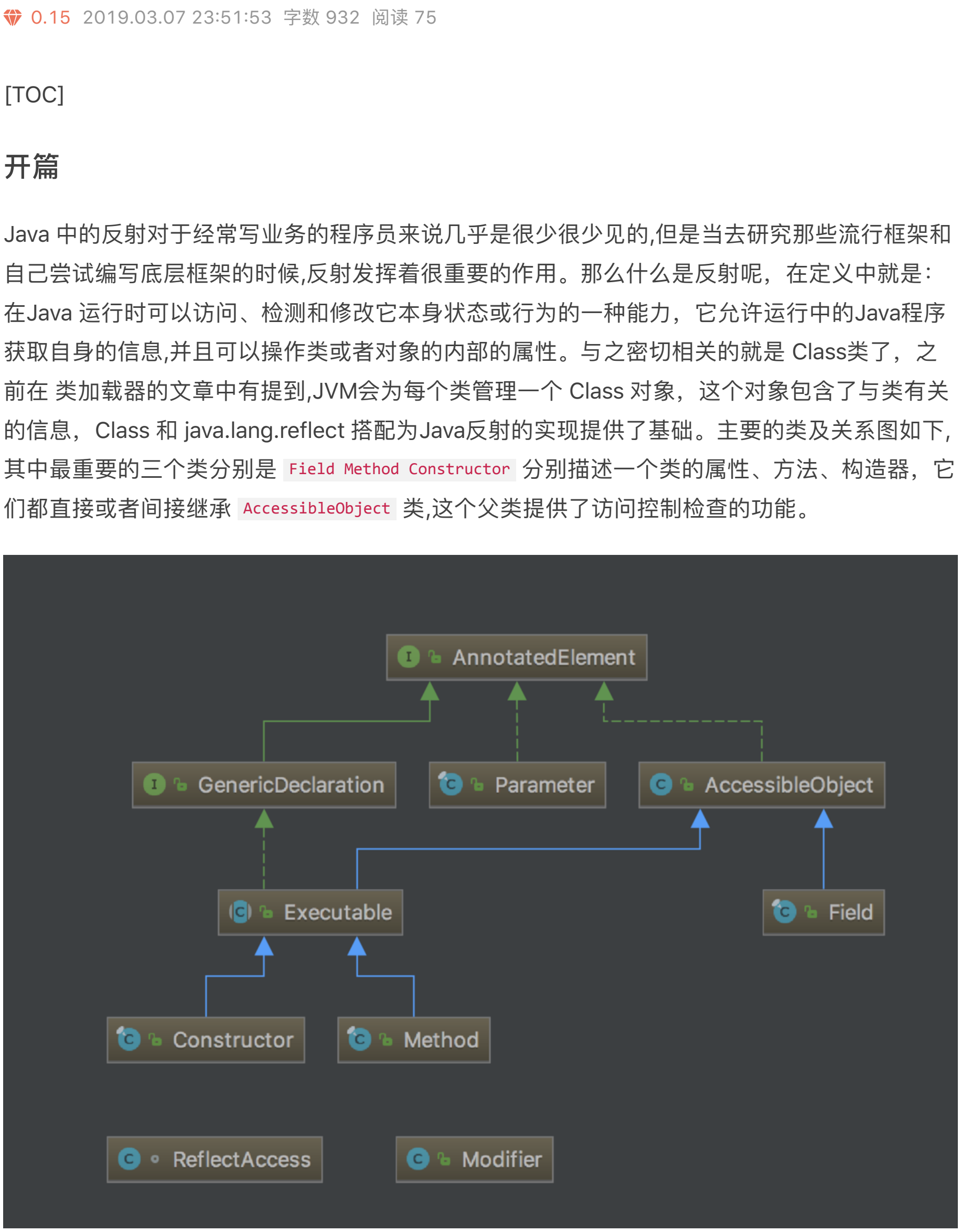
Java 反射 Reflection

0.15 2019.03.07 23:51:53 字数 932 阅读 75

[TOC]

开篇

Java 中的反射对于经常写业务的程序员来说几乎是很少很少见的,但是当去研究那些流行框架和自己尝试编写底层框架的时候,反射发挥着很重要的作用。那么什么是反射呢，在定义中就是：在Java 运行时可以访问、检测和修改它本身状态或行为的一种能力，它允许运行中的Java程序获取自身的信息,并且可以操作类或者对象的内部的属性。与之密切相关的就是 Class类了，之前在 类加载器的文章中有提到,JVM会为每个类管理一个 Class 对象，这个对象包含了与类有关的信息，Class 和 java.lang.reflect 搭配为Java反射的实现提供了基础。主要的类及关系图如下,其中最重要的三个类分别是 `Field Method Constructor` 分别描述一个类的属性、方法、构造器，它们都直接或者间接继承 `AccessibleObject` 类,这个父类提供了访问控制检查的功能。



Java反射架构

通过Class获取类的信息

```
1  package cn.xisole.sky.reflect;
2
3  import lombok.Data;
4
5  /**
6   * OrderDO 订单实体类
```

华为云 828 | 企业上云节 限时秒杀 云服务器 1. 注册送价值188 注册参与
isoleHero 拥有32钻 (约
华为云 0元试用 数据库、安全、 立即体验

```java
 */
@Data
public class OrderDO {
    /**
     * 订单id
     */
    private String orderId;
    /**
     * 会员id
     */
    private String customerId;
    /**
     * 会员id
     */
    private String shopId;
    /**
     * 订单的创建时间
     */
    private Long createTime;

    public OrderDO(String orderId, String customerId, String shopId, Long createTime) {
        this.orderId = orderId;
        this.customerId = customerId;
        this.shopId = shopId;
        this.createTime = createTime;
    }

    public OrderDO() {
    }
}
// 子类

package cn.xisole.sky.reflect;

import lombok.Data;

/**
 * TakeoutOrderDO 外卖订单实体类
 *
 * @author yupao
 * @since 2019/3/7 下午11:06
 */
@Data
public class TakeoutOrderDO extends OrderDO{
    /**
     * 配送地址
     */
    private String deliveryAddress;
    /**
     * 客户电话
     */
    private String customerPhone;
    /**
     * 客户姓名
     */
    private String customerName;
    public TakeoutOrderDO(String orderId, String customerId, String shopId, Long createTime) {
        super(orderId, customerId, shopId, createTime);
    }

    /**
     * 定义一个方法
     */
    public void printInfo(){
        System.out.println(this);
    }

    public TakeoutOrderDO() {
        super();
    }

    @Override
    public String toString() {
```

```
        return "TakeoutOrderDO{" +
                "deliveryAddress='" + deliveryAddress + '\'' +
                ", customerPhone='" + customerPhone + '\'' +
                ", customerName='" + customerName + '\'' +
                '}' + super.toString();
    }
}

//测试类
package cn.xisole.sky.reflect;

import java.lang.reflect.*;

/**
 * MainReflect
 *
 * @author yupao
 * @since 2019/3/7 下午11:12
 */
public class MainReflect {
    public static void main(String[] args) throws ClassNotFoundException, IllegalAccessException,
        /**
         * 一:获取 Class 对象的三种方式
         */
        /**
         * 通过 Class.forName 直接获取 Class 对象
         */
        Class c1 = Class.forName("cn.xisole.sky.reflect.TakeoutOrderDO");
        /**
         * 通过类直接获取 Class 对象
         */
        Class c2 = TakeoutOrderDO.class;

        /**
         * 通过实例的 getClass 方法获得 Class 对象
         */
        TakeoutOrderDO takeoutOrderDO = new TakeoutOrderDO("orderId","customerId","shopId",111L);
        Class c3 = takeoutOrderDO.getClass();

        System.out.println(c1 + "\n" + c2 + "\n" + c3);

        System.out.println("c1==c2==c3:"+(c1==c3 && c1==c3));
        /**
         * 二:通过反射来创建类的实例
         */
        /**
         * 1. 通过Class对象的 newInstance 方法来创建Class对象的实例
         */
        TakeoutOrderDO takeoutOrderDO1 = (TakeoutOrderDO)c1.newInstance();
        /**
         * 2. 通过Class对象获取指定的 Constructor对象，再调用Constructor对象的newInstance方法创建实例
         */
        Constructor constructor = c1.getConstructor(String.class,String.class,String.class,Long.
        TakeoutOrderDO takeoutOrderDO2 = (TakeoutOrderDO) constructor.newInstance("param1","paran
        System.out.println(takeoutOrderDO1 + "\n" + takeoutOrderDO2);
        System.out.println("takeoutOrderDO1 == takeoutOrderDO2:"+(takeoutOrderDO1 == takeoutOrder

        /**
         * 三:通过反射获取类的信息
         */
        System.out.println("c1的父类:"+c1.getSuperclass().getName());
        System.out.println("c1的构造器:"+c1.getConstructor());
        System.out.println("c1的属性:");
        Field[] fields = c1.getDeclaredFields();
        for(Field field:fields){
            System.out.println(field.getType() +" "+ field.getName());
        }
        System.out.println("c1的方法:");
        Method[] methods = c1.getDeclaredMethods();
        for (Method method : methods){
            System.out.println(method.getName());
        }

```

```
156
157         }
158
159     }
160
```

上面的Demo主要展示了如何获取Class类以及如果根据Class得到对象的实例。可以发现,当我们拿到了 Class 类的时候,我们就直接掌控了这个类了。无论是获取类的属性还是构造器还是方法，都只是Class对象的管理的一部分而已，因为在JVM内部，一个个的类就是一个个的Class。此外我们还可以对获取到的结果进行进一步的需要的操作，比如说可以得到Methon，然后Methon可以执行Invoke方法去执行等等诸如这种操作。

小结

反射的优点：

- 可扩展性:应用程序可以利用全限定名创建可扩展对象的实例，来使用来自外部的用户自定义类。
- 方便实时查看类的成员,这点不用说了就是干这个的
- 调试器和测试工具:调试器需要能够检查一个类里的私有成员。测试工具可以利用反射来自动地调用类里定义的可被发现的 API 定义，以确保一组测试中有较高的代码覆盖率。

反射的缺点:
尽管反射非常强大，但也不能滥用。如果一个功能可以不用反射完成那么最好不用。在我们使用反射技术时，有一些缺点需要了解下:

- 性能开销： 反射涉及了动态类型的解析，所以 JVM 无法对这些代码进行优化。因此，反射操作的效率要比那些非反射操作低得多。我们应该避免在经常被执行的代码或对性能要求很高的程序中使用反射。
- 安全限制： 使用反射技术要求程序必须在一个没有安全限制的环境中运行。
- 内部暴露： 由于反射允许代码执行一些在正常情况下不被允许的操作（比如访问私有的属性和方法），所以使用反射可能会导致意料之外的副作用，这可能导致代码功能失调并破坏可移植性。反射代码破坏了抽象性，因此当平台发生改变的时候，代码的行为就有可能也随着变化。

3人点赞 >

Java基础

isoleHero
拥有32钻 (约4.95元)

关注

"未来可期"

赞赏

推荐阅读

更多精彩内容 >

深入理解Java类型信息(Class对象)与反射机制

深入理解Class对象 RRTI的概念以及Class对象作用 认识Class对象之前，先来了解一个概念，RTTI（...

架构师springboot

公共技术点之 Java 反射 Reflection

1. 了解 Java 中的反射 1.1 什么是 Java 的反射 Java 反射是可以让我们在运行时获取类的函数、...

Ten_Minutes

【转】公共技术点之 Java 反射 Reflection

项目：，分析者：Mr.Simple，校对者：Trinea本文为 Android 开源项目源码解析 公共技术点中的 ...

zizi192

「转载」Java 反射 Reflection

一、了解 Java 中的反射 1. 什么是 Java 的反射 Java反射是在程序运行时获取类的函数、属性、父类和...

liwei_happyman

国画班

ambermm