何为反射?

如果说大家研究过框架的底层原理或者咱们自己写过框架的话，一定对反射这个概念不陌生。

反射之所以被称为框架的灵魂，主要是因为它赋予了我们在运行时分析类以及执行类中方法的能力。

通过反射你可以获取任意一个类的所有属性和方法，你还可以调用这些方法和属性。

反射的应用场景了解么?

像咱们平时大部分时候都是在写业务代码，很少会接触到直接使用反射机制的场景。

但是，这并不代表反射没有用。相反，正是因为反射，你才能这么轻松地使用各种框架。像 Spring/Spring Boot、MyBatis 等等框架中都大量使用了反射机制。

这些框架中也大量使用了动态代理，而动态代理的实现也依赖反射。

比如下面是通过 JDK 实现动态代理的示例代码，其中就使用了反射类 `Method` 来调用指定的方法。

```
public class DebugInvocationHandler implements InvocationHandler {
    /**
     * 代理类中的真实对象
     */
    private final Object target;

    public DebugInvocationHandler(Object target) {
        this.target = target;
    }


    public Object invoke(Object proxy, Method method, Object[] args) throws
InvocationTargetException, IllegalAccessException {
        System.out.println("before method " + method.getName());
        Object result = method.invoke(target, args);
        System.out.println("after method " + method.getName());
        return result;
    }
}
```

另外，像 Java 中的一大利器 **注解** 的实现也用到了反射。

为什么你使用 Spring 的时候 ，一个`@Component`注解就声明了一个类为 Spring Bean 呢？为什么你通过一个`@Value`注解就读取到配置文件中的值呢？究竟是怎么起作用的呢？

这些都是因为你可以基于反射分析类，然后获取到类/属性/方法/方法的参数上的注解。你获取到注解之后，就可以做进一步的处理。

谈谈反射机制的优缺点

优点： 可以让咱们的代码更加灵活、为各种框架提供开箱即用的功能提供了便利

缺点：让我们在运行时有了分析操作类的能力，这同样也增加了安全问题。比如可以无视泛型参数的安全检查（泛型参数的安全检查发生在编译时）。另外，反射的性能也要稍差点，不过，对于框架来说实际是影响不大的。相关阅读：Java Reflection: Why is it so slow?

反射实战

获取 Class 对象的四种方式

如果我们动态获取到这些信息，我们需要依靠 Class 对象。Class 类对象将一个类的方法、变量等信息告诉运行的程序。Java 提供了四种方式获取 Class 对象:

1.知道具体类的情况下可以使用：

```
Class alunbarClass = TargetObject.class;
```

但是我们一般是不知道具体类的，基本都是通过遍历包下面的类来获取 Class 对象，通过此方式获取 Class 对象不会进行初始化

2.通过 `Class.forName()`传入类的路径获取：

```
Class alunbarClass1 = Class.forName("cn.javaguide.TargetObject");
```

3.通过对象实例`instance.getClass()`获取：

```
TargetObject o = new TargetObject();
Class alunbarClass2 = o.getClass();
```

4.通过类加载器`xxxClassLoader.loadClass()`传入类路径获取:

```
class clazz = ClassLoader.LoadClass("cn.javaguide.TargetObject");
```

通过类加载器获取 Class 对象不会进行初始化，意味着不进行包括初始化等一些列步骤，静态块和静态对象不会得到执行

反射的一些基本操作

简单用代码演示一下反射的一些操作!

1.创建一个我们要使用反射操作的类 `TargetObject`。

```

```

```
package cn.javaguide;

public class TargetObject {
    private String value;

    public TargetObject() {
        value = "JavaGuide";
    }

    public void publicMethod(String s) {
        System.out.println("I love " + s);
    }

    private void privateMethod() {
        System.out.println("value is " + value);
    }
}
```

2.使用反射操作这个类的方法以及参数

```
package cn.javaguide;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class Main {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, IllegalAccessException, InstantiationException,
InvocationTargetException, NoSuchFieldException {
        /**
         * 获取TargetObject类的Class对象并且创建TargetObject类实例
         */
        Class<?> tagetClass = Class.forName("cn.javaguide.TargetObject");
        TargetObject targetObject = (TargetObject) tagetClass.newInstance();
        /**
         * 获取所有类中所有定义的方法
         */
        Method[] methods = tagetClass.getDeclaredMethods();
        for (Method method : methods) {
            System.out.println(method.getName());
        }
        /**
         * 获取指定方法并调用
         */
        Method publicMethod = tagetClass.getDeclaredMethod("publicMethod",
                String.class);

        publicMethod.invoke(targetObject, "JavaGuide");
        /**
         * 获取指定参数并对参数进行修改
```

```
             */
            Field field = tagetClass.getDeclaredField("value");
            //为了对类中的参数进行修改我们取消安全检查
            field.setAccessible(true);
            field.set(targetObject, "JavaGuide");
            /**
             * 调用 private 方法
             */
            Method privateMethod = tagetClass.getDeclaredMethod("privateMethod");
            //为了调用private方法我们取消安全检查
            privateMethod.setAccessible(true);
            privateMethod.invoke(targetObject);
        }
    }
```

输出内容：

```
publicMethod
privateMethod
I love JavaGuide
value is JavaGuide
```

注意：有读者提到上面代码运行会抛出 `ClassNotFoundException` 异常,具体原因是你没有下面把这段代码的包名替换成自己创建的 `TargetObject` 所在的包 。

```
Class<?> tagetClass = Class.forName("cn.javaguide.TargetObject");
```