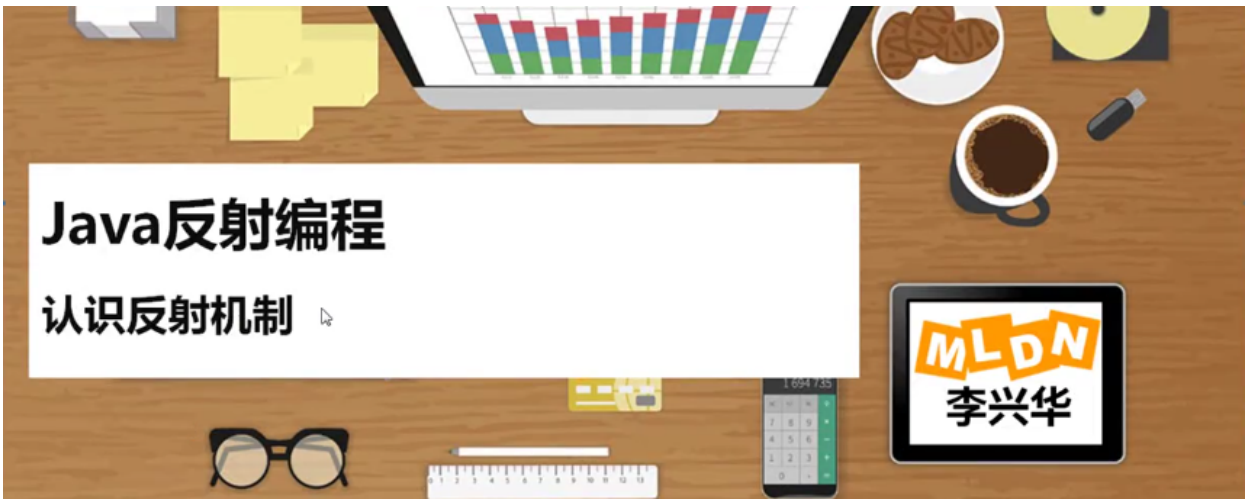


# 2、具体内容

在Java语言里面之所会有如此众多的开源技术支撑，很大的一部分是来自于Java最大的特征——反射机制，如果不能够灵活去使用反射机制进行项目的开发与设计，那么可以说并未接触到Java的精髓。

重邮的技术实现的目标只有一点，重用性。

对于反射技术首先来考虑的是“反”与“正”的操作，所谓的“正”操作指的是当我们要使用一个类的时候，一定要先导入程序所在的包，而后根据类进行对象实例化，并且依靠对象调用类中的方法。但是如果说“反”，根据实例化对象反推出其类型。

**范例：正向操作**

```
import java.util.Date;      // 1、导入程序所在的包.类，知道对象的出处了
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Date date = new Date() ;        // 2、通过类产生实例化对象
        System.out.println(date.getTime()); // 3、根据对象调用类中的方法
    }
}
```

如果要想实现反的处理操作，那么首先要采用的就是Object类中所提供的一个的方法：

·获取Class对象信息：public final Class<?> getClass();

**范例：观察Class对象的使用**

```
import java.util.Date;      // 1、导入程序所在的包.类，知道对象的出处了
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Date date = new Date() ;        // 2、通过类产生实例化对象
        System.out.println(date.getClass()); // 根据实例化对象找到对象的所属类型
    }
}
```

getClass()可以帮助使用者找到对象的根源。


**博客：** https://www.cnblogs.com/HOsystem/p/14116443.html

# 2、具体内容

反射之中所有的核心操作都是通过Class类对象展开的，可以说Class类时反射操作的根源所在，但是这个类如果要想获取它的实例化对象，可以采用三种方式完成，首先来观察java.lang.Class类的定义：

```
public final class Class<T> extends Object implements Serializable, GenericDeclaration,
Type, AnnotatedElement
```

从JDK1.5开始Class类在定义的时候可以使用泛型进行标记，这样的用法主要是希望可以避免所谓的向下转型。下面通过具体的操作讲解三种实例化关系。

1、【Object类支持】Object类可以根据实例化对象获取Class对象：public final Class<?> getClass()：

·这种方式有一个不是缺点的缺点：如果现在只是想获得Class类对象，则必须产生指定类对象后才可以获得；

```
class Person {} // 采用自定义的程序类
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Person per = new Person() ; // 已经存在有指定类的实例化对象
        Class<? extends Person> cls = per.getClass() ;
        System.out.println(cls.getName()); // 获取的是类的完整名称
    }
}
```

2、【JVM直接支持】采用“类.class”的实例化：

·特点：如果要采用此种模式，则必须导入程序所对应的开发包

```
class Person {} // 采用自定义的程序类
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Class<? extends Person> cls = Person.class;
        System.out.println(cls.getName()); // 获取的是类的完整名称
```

```
    }
}
```

3、【Class类支持】在Class类里面提供有一个static方法：

·加载类：public static Class<?> forName(**String** className) throws ClassNotFoundException；

```
package cn.mldn.vo;
public class Person {
}
```

```
package cn.mldn.demo;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Class<?> cls = Class.forName("cn.mldn.vo.Person") ;
        System.out.println(cls.getName()); // 获取的是类的完整名称
    }
}
```

这种模式最大的特点是可以直接采用字符串的形式定义要使用的类型，并且程序中不需要编写任何的import语句。如果此时要使用的程序类不存在则会抛出“java.lang.ClassNotFoundException”异常。