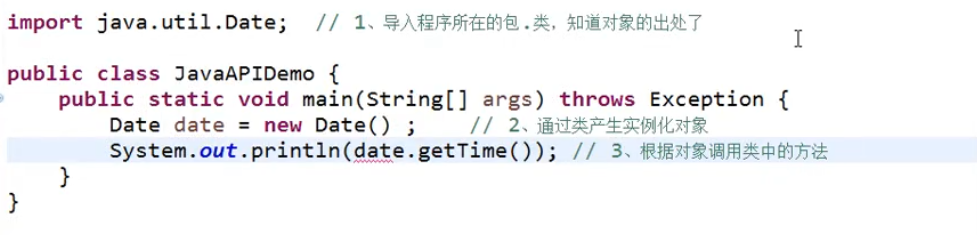
1. 反射简介

在Java语言中之所以会有如此多的开源技术支撑，很大一部分原因来自Java最大的特征——反射机制。如果不能灵活的使用反射机制进行项目的开发与设计，那么可以说你还并没有接触到Java的精髓。

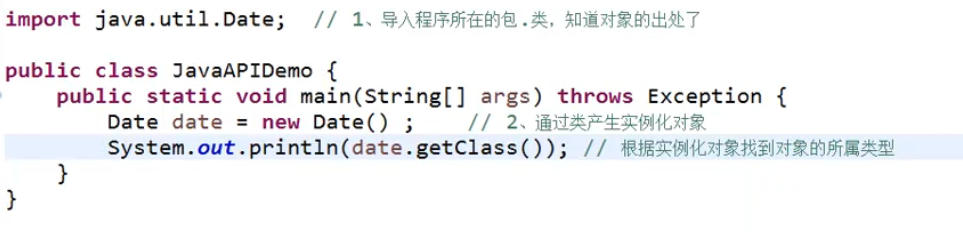
所有的技术实现的目标只有一点:重用性

对于反射技术首先考虑的是“反”与“正”的操作，所谓正就是当我们要使用一个类的时候，一定是先导入程序所在的包，然后根据类进行对象的实例化，并且依靠对象调用的类的方法。但是如果说反:就是根据实例化对象反推出其类型。

范例:正向操作

如果要想实现反的处理操作，那么我们首先需要采用Object类中的一个操作方法

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

范例:观察getClass()的使用方法；

getClass()可以帮助使用者找到对象的根源。

1. Class类对象的三种实例化模式

反射之中所有核心操作都是通过Class对象展开，可以说Class类是反射的根源所在，但是这个类如果要想获取他的实例化对象，可以使用三种方式完成。

首先来观察java.lang.Class类的定义:

public final class Class<T>

extends [Object](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Object.html)

implements [Serializable](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/io/Serializable.html), [GenericDeclaration](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/reflect/GenericDeclaration.html), [Type](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/reflect/Type.html), [AnnotatedElement](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/reflect/AnnotatedElement.html), [TypeDescriptor.OfField](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/invoke/TypeDescriptor.OfField.html)<[Class](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Class.html)<?>>, [Constable](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/constant/Constable.html)

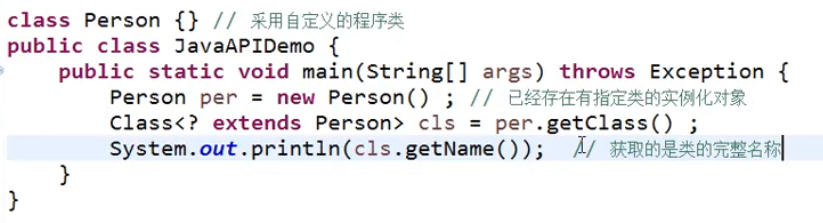
从JDK1.5开始Class类在定义的时候可以使用泛型进行标记，这样的用法主要是希望可以避免所谓的向下转型。

范例:三种实例化形式

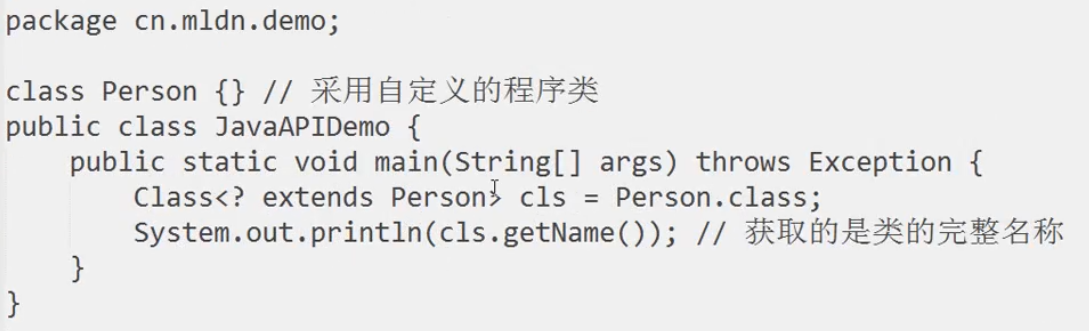
1. 【Object类支持】Object类对象可以根据实例化对象获取Class对象

·public final Class<?> getClass();

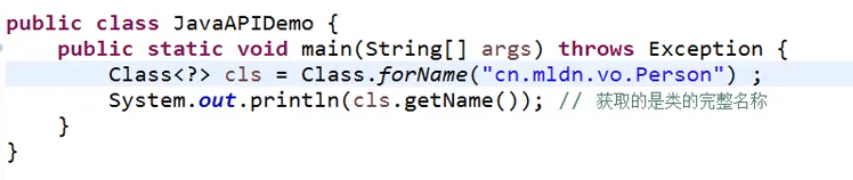
注意:这个方法获取Class类对象，则必须产生指定类的实例化对象

后才能获得。

1. 【JVM直接支持】采用 类.class 的形式实例化

·特点:如果想要使用这种方法，则必须导入程序所在对象的开发包。

3、【Class类支持】在Class类之中提供一个static方法

 ·加载类:public static [Class](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Class.html)<?> forName​([String](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/String.html) className) throws [ClassNotFoundException](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/ClassNotFoundException.html)

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。

经过一系列的分析以后发现虽然获得了Class类的实例化对象，但是依然觉得这个对象的获取的意义不大，所以为了进一步理解反射的核心意义所在，下面将通过几个案例进行程序说明(都是在实际开发中一定会用到的)。

1. 反射实例化对象

获取Class对象之后最大的意义不在于只是一个对象实例化的形式，更重要的是Class类提供有一个对象的反射实例化方法（代替了关键字new）:

·在JDK1.9以前的实例化：public T newInstance() throws Exception

·在JDK1.9之后：clazz.getDeclaredConstructor().newInstance();

范例:通过newInstance（）方法实例化Person对象

|  |
| --- |
| public class JavaDemo1 {  public static void main(String[] args) throws Exception {  Class<?> aClass = Class.forName("com.company.反射机制.entity.Person");  //实例化对象，1.9之后被废除  Object o = aClass.newInstance();  //实例化对象，1.9之后的版本  Object instance = aClass.getDeclaredConstructor().newInstance();  }  } |
| |  |  | | --- | --- | | 执行结果 |  | |

现在通过反射实现的对象实例化处理，依然要调用对象的无参构造方法，其本质等价于“类 对象 = new 类（）”，也就是所相当于隐含了关键字new，而直接使用String字符串进行代替。因为JDK1.9之前的newInstance只能调用无参构造，所以很多开发者认为其描述的不准确，于是将其变换了形式。

·aClass.getDeclaredConstructor().newInstance();