反射：正常情况下，要使用一个类，必须找到此类，并且实例化对象，必须现有类再有对象，那么如果现在通过对象找到类呢？

在Object类中有一个方法：public final Class<?> getClass()

所有的对象都可以使用getClass()方法，现在要返回一个Class，就是通过Class反射完成：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo {  **public** **static** **void** main(String[] args) {  Person p = **new** Person();  System.*out*.println(p.getClass().getName());  }  } |

## Class类：

在反射机制中，Class类是一个操作的源头，所有的反射操作都要由此展开，但是如果想实例化此类，有三种方式完成：

·通过Object类中的getClass方法：public final Class<?> getClass()

·通过class类形式：类.class

·通过Class类的静态方法：public static Class<?> forName(String className) throws ClassNotFoundException

第一种形式：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo01 {  **public** **static** **void** main(String[] args) {  Person p = **new** Person();  Class<?> c = p.getClass();  System.*out*.println(c.getName());  }  } |

第二种形式：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo02 {  **public** **static** **void** main(String[] args) {  Class<?> c = Person.**class**;  System.*out*.println(c.getName());  }  } |

第三种方式：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo03 {  **public** **static** **void** main(String[] args) {  Class<?> c = **null**;  **try** {  c = Class.*forName*("com.jjm.Person");  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  }  System.*out*.println(c.getName());  }  } |

第三种：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo03 {  **public** **static** **void** main(String[] args) {  Class<?> c = **null**;  **try** {  c = Class.*forName*("com.jjm.Person");  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  }  System.*out*.println(c.getName());  }  } |

虽然有三种方法，但是最常用的是forName()方法，其次就是类.class形式

### 通过Class 类实例化对象：

在Class类中实例化对象是一种最常用的操作，以后所有的框架程序都是以这个原理实现的。

正常情况下，通过new关键字实例化对象，现在我们通过Class完成，操作步骤如下：

·根据packeg.calss实例化class 对象

·通过Class类中的方法：public T newInstance() throws InstantiationException, IllegalAccessException

下面来看代码：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo04 {  **public** **static** **void** main(String[] args) {  Class<?> c = **null**;  **try** {  c = Class.*forName*("com.jjm.Person");  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  }  **try** {  Person p = (Person) c.newInstance();  System.*out*.println(p.say());  } **catch** (InstantiationException e) {  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  }  }  } |

现在对象实例化成功了，以上代码开发中可能会经常使用到，但是以上程序有一个问题，现在要将Person类改一下：

|  |
| --- |
| **package** com.jjm;  **public** **class** Person {  **private** String name;  **private** **int** age;  **public** Person(String name, **int** age) {  **super**();  **this**.name = name;  **this**.age = age;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String say(){  **return** "Hello World";  }  } |

此类中现在有一个有参构造，那现在我调用之前的程序，看看会有什么问题？

|  |
| --- |
| java.lang.InstantiationException: com.jjm.Person  at java.lang.Class.newInstance0(Class.java:340)  at java.lang.Class.newInstance(Class.java:308)  at com.jjm.PersonClassDemo04.main(PersonClassDemo04.java:12) |

此时程序实例化异常，对象实例化失败，因为类中并没有无参构造方法，所以出现了以上的错误。

### 实例化时调用指定的构造方法

以上的操作中类必须要有一个无参的构造方法，而如果现在没有无参的构造方法，则必须指定构造方法，并向指定的构造方法传递指定的参数才能进行实例化操作：

如果想要指定使用构造方法，必须得到一个类的构造方法，可以使用以下方法的都类的构造方法：public Constructor<?>[] getConstructors() throws SecurityException

以上的方法可以得到一个类中的全部的构造方法，会议Constructor对象数组的形式返回，Constructor是一个类此类，在java.lang.reflect包中，以后凡是看见在java.lang.reflect中的类，都是属于反射操作类，java.lang.reflect是反射操作包

取得构造方法之后，可以通过Constructor类中有一个方法进行实例化：public T newInstance(Object... initargs) throws InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException

|  |
| --- |
| **package** com.jjm;  **import** java.lang.reflect.Constructor;  **public** **class** PersonClassDemo05 {  **public** **static** **void** main(String[] args) **throws** Exception{  Class<?> c = **null**;  Person p = **null**;  c = Class.*forName*("com.jjm.Person");  Constructor<?> cus[] = c.getConstructors();  p = (Person)cus[0].newInstance("姜建民",30);  System.*out*.println(p.say());    }  } |

以上的操作相对于第一种给对象实例化方式非常的麻烦，所以在以后的开发中，都要习惯性的给类中加一个无参的构造方法。

### 通过Class 类取得完整的结构

Class 有很多的方法，可以轻易的取得一个类的全部得定义的构造方法，普通方法，常量，变量，实现的接口，继承的类。

#### 取得Person类中所有的构造方法：

|  |
| --- |
| **package** com.jjm;  **import** java.lang.reflect.Constructor;  **public** **class** PersonClassDemo06 {  **public** **static** **void** main(String[] args) **throws** Exception{  Class<?> c = **null**;  c = Class.*forName*("com.jjm.Person");  Constructor<?> cus[] = c.getConstructors();//取得所有的构造方法  **for** (**int** i = 0; i < cus.length; i++) {  System.*out*.println(cus[i]);  }    }  } |

#### 取得Person类的父类：

在Class 类中，存在以下的方法，取得一个类继承的父类：public Class<? super T> getSuperclass()：

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo7 {  **public** **static** **void** main(String[] args) **throws** Exception {  Class<?> c = Class.*forName*("com.jjm.Person");  Class<?> sp = c.getSuperclass();  System.*out*.println(sp.getName());  }  } |

#### 取得Person类实现的接口：

Class类中定义了一个方法，可以取得类所实现的全部接口：

public Class<?>[] getInterfaces()

|  |
| --- |
| **package** com.jjm;  **public** **class** PersonClassDemo8 {  **public** **static** **void** main(String[] args) **throws** Exception {  Class<?> c = Class.*forName*("com.jjm.Person");  Class<?> sp[] = c.getInterfaces();  **for** (**int** i = 0; i < sp.length; i++) {  System.*out*.println(sp[i]);  }  }  } |

#### 取得Person类实全部的方法：getMethod()

取得方法的返回类型：

#### 取得Person类实全部的属性：getField()

#### 通过反射调用类中的方法：

正常情况下，类中的方法是对象名.方法（），类名。方法（），在反射机制中，通过反射也可以调用类中的方法。

首先我们知道，Class可以找到类中的方法，只要传递可变参数就可以调用方法：

public Object invoke(Object obj,

Object... args)

throws IllegalAccessException,

IllegalArgumentException,

InvocationTargetException

|  |
| --- |
| **package** com.jjm;  **import** java.lang.reflect.Method;  **public** **class** PersonClassDemo9 {  **public** **static** **void** main(String[] args) **throws** Exception {  Class<?> c = Class.*forName*("com.jjm.Person");  Method met = c.getMethod("say");//取得指定的方法  met.invoke(c.newInstance());  }  } |

以上调用了没有参数，没有返回值类型的类的方法，

|  |
| --- |
| **package** com.jjm;  **import** java.lang.reflect.Method;  **public** **class** PersonClassDemo10 {  **public** **static** **void** main(String[] args) **throws** Exception {  Class<?> c = Class.*forName*("com.jjm.Person");  Method met = c.getMethod("say1");//取得指定的方法  String str = (String)met.invoke(c.newInstance());  System.*out*.println(str);  }  } |

以上取得了方法的返回值

|  |
| --- |
| **package** com.jjm;  **import** java.lang.reflect.Method;  **public** **class** PersonClassDemo11 {  **public** **static** **void** main(String[] args) **throws** Exception {  Class<?> c = Class.*forName*("com.jjm.Person");  Method met = c.getMethod("say2",String.**class**,**int**.**class**);//取得指定的方法(参数)  String str = (String)met.invoke(c.newInstance(),"姜建民",30);  System.*out*.println(str);  }  } |