反射：一般来说，我们定义一个变量或者实例化一个类，我们都需要先找到类名，才能定义，那么现在如果想通过一个对象找到他的类型？

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

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **public** **class** ClassDemo01 {  **public** **static** **void** main(String[] args) {  Person p=**new** Person();  System.*out*.println(p.getClass().getName());  }  } |

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，如果想实例化此类，有三种方式：

1. public final Class <?> getClass().
2. 通过class类形式：类.class

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **public** **class** ClassDemo01 {  **public** **static** **void** main(String[] args) {  Class<?> c=Person.**class**;  System.*out*.println(c.getName());  }  } |

是通过Class类的静态方法：public static [Class](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html)<?> **forName**([String](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) className) throws [ClassNotFoundException](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/ClassNotFoundException.html)

我们说虽然有三种方法，但是我们最常用的是使用forName()方法，其次是类.class的形式

|  |
| --- |
| public static [Class](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html)<?> **forName**([String](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) className)  throws [ClassNotFoundException](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/ClassNotFoundException.html) |

public [T](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html) **newInstance**()throws [InstantiationException](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/InstantiationException.html),[IllegalAccessException](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/IllegalAccessException.html)

上面的方法是用来创建一个该类的新实例的

下面就是通过Class进行类的实例化的方法

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **public** **class** ClassDemo01 {  **public** **static** **void** main(String[] args) {  Class<?> c=**null**;  **try** {  c=Class.*forName*("com.wanczy.ClassDemo.Person");  } **catch** (ClassNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  **try** {  Person p=(Person) c.newInstance();  System.*out*.println(p.say());  } **catch** (InstantiationException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IllegalAccessException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

以上的方式存在一个问题，如果Person类没有无参的构造函数，那么我们就无法成功实例化

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **public** **class** Person {  **private** String name;  **private** **int** 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** Person(String name,**int** age) {  **this**.name=name;  **this**.age=age;  }  **public** String say(){  **return** "这是一个Person";  }  }  java.lang.InstantiationException: com.wanczy.ClassDemo.Person  at java.lang.Class.newInstance0(Class.java:340)  at java.lang.Class.newInstance(Class.java:308)  at com.wanczy.ClassDemo.ClassDemo01.main(ClassDemo01.java:13) |

所以说在严格的开发过程中，我们创建一个类，一般都要跟上他的无参构造方法.

我们可以通过将目标类中的构造方法转换为一个Constructor类的对象数组来实现我们对象的实例化，Constructor类 提供了有参数的newInstance方法

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **import** java.lang.reflect.Constructor;  **public** **class** ClassDemo01 {  **public** **static** **void** main(String[] args) **throws** Exception{  Class<?> c=**null**;  Person p=**null**;  c=Class.*forName*("com.wanczy.ClassDemo.Person");  Constructor<?> cus[]=c.getConstructors();  p = (Person) cus[0].newInstance("张三",30);  System.*out*.println(p.say());  }  } |

反射机制的重点就是利用forName（）方法来实例化对象，以达到解耦合的目的。

获得Person的父类：

public [Class](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html)<? super [T](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html)> **getSuperclass**()

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **import** java.lang.reflect.Constructor;  **public** **class** ClassDemo01 {  **public** **static** **void** main(String[] args) **throws** Exception{  Class<?> c=Class.*forName*("com.wanczy.ClassDemo.Person");  Class<?> sp=c.getSuperclass();  System.*out*.println(sp.getName());  }  } |

获得接口的方法：

public [Class](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Class.html)<?>[] **getInterfaces**()

取得Person类的所有方法的方法：getMethod()

取得Person类的所有属性的方法：getField()

|  |
| --- |
| **package** com.wanczy.ClassDemo;  **import** java.lang.reflect.Method;  **public** **class** ClassDemo01 {  **public** **static** **void** main(String[] args) **throws** Exception{  Class<?> c=Class.*forName*("com.wanczy.ClassDemo.Person");  Method met=c.getMethod("say");  System.*out*.println(met.invoke(c.newInstance()));  }  } |

以上的代码可以不进行实例化而直接调用该类的方法。

有一个生产者，他一次能生产50个产品，然后有12个消费者，一次能够购买5个产品。

要大家写一个多线程来实现这个过程，当产品不够的时候提示“正在生产，请等待”。