**多态的简介**

1. **简介**

多态性时面向对象的三大主要特征，多态是在继承的基础上扩展来的，也就是说可以实现父子类之间的互相转换处理。

1. **多态性的基本概念**
   1. 在Java中对于多态有两种实现模式

· 方法的多态性：

方法的重载:同一个方法名称可以更具传入的参数的类型或个数实现不同的功能

方法的覆写: 子类对父类方法的扩展，不同的子类在覆写父类方法时也要根据实际情况来实现(同一个方法可能根据子类的不同有不同的功能。)

· 对象多态性:父子实例之间的转换处理，他有两种模式:

-对象向上转型: 父类 父类实例 = 子类实例 (这是自动完成转换的)

-对象向下转型: 子类 子类实例 = (子类) 父类实例 (这是强制完成转换)

在实际开发中，大部分情况考虑最多的是对象的向上转型(90%)，而向下转型不到%3。剩下百分之七是一些不需要考虑转型的类(String 类 )；

范例:(JavaDemo.java)

|  |
| --- |
| class Message {  public void println(){  System.out.println("父类方法");  }  }  class DataBaseMessage extends Message{  public void println(){  System.out.println("Oracle数据库相关信息");  }  }  public class JavaDemo{  public static void main(String[] args) {  DataBaseMessage msg = new DataBaseMessage();  msg.println();  }  } |

本程序是一个最简单的方法覆写来实现操作的，由于new 的实例对象是子类DataBaseMessage 而子类调用的方法是它覆写过的方法。

1. **观察向上转型:（向上转型的目标是为了参数的统一设计）**

|  |
| --- |
| class Message {  public void println(){  System.out.println("父类方法");  }  }  class DataBaseMessage extends Message{  public void println(){  System.out.println("Oracle数据库相关信息");  }  }  public class JavaDemo{  public static void main(String[] args) {  Message msg = new DataBaseMessage();  msg.println();  }  } |

通过上述我们无法发现向上转型的用处，那么再观察下面这段代码。

|  |
| --- |
| class Message {  public void println(){  System.out.println("父类方法");  }  }  class DataBaseMessage extends Message{  public void println(){  System.out.println("Oracle数据库相关信息");  }  }  class WebService extends Message{  public void println(){  System.out.println("WebService服务启动");  }  }  public class JavaDemo2{  public static void main(String[] args) {  fun(new DataBaseMessage());  fun(new WebService());  }  public static void fun(Message msg){ //不管传递哪一个子类都可以  msg.println();  }  } |

思考一下 如果fun 中的Message 换成子类并且 其它子类都通过方法的重载来实现，那么会出现什么情况？当我们有300 个甚至更多的子类时，我们需要重载 300 个方法！因此向上转型的特点就是对参数的统一设计。 而这样看来，重载虽然能实现同样的功能，但不我们不能只是考虑到功能的实现，还应该考虑到维护的问题。因此我们可以知道向上转型比用重载更加便于维护。

1. **对象向下转型**

向下转型的主要特点在于需要使用到一些子类自己特殊的定义处理。

范例:

|  |
| --- |
| class Person {  public void println(){  System.out.println("我是person类");  }  }  class SupperMan extends Person{  public void fly(){  System.out.println("我是SupperMan：我可以飞");  }  public void fir(){  System.out.println("我是person类：我可以喷火");  }  }  public class JavaDemo2{  public static void main(String[] args) {  Person per = new SupperMan();  per.println();  SupperMan sup = (SupperMan)per;  sup.fly();  sup.fir();  }  } |

向下转型的目的就是调用子类特有的方法，如果直接调用per.fly(),将会找不到方法，因为per 定义的是person类。

但是我们必须清楚向下转型是不安全的，因为再向下转型之前必须进行向上转型。

范例(错误代码):（其余类与上面正确代码一样）

|  |
| --- |
| public class JavaDemo3{  public static void main(String[] args) {  Person per = new Person();  per.println();  SupperMan sup = (SupperMan)per;  sup.fly();  sup.fir();  }  } |

此时会报错:Exception in thread "main" java.lang.ClassCastException: Person cannot be cast to SupperMan 。**以后只要是向下转型必须先进行向上转型，两个没有任何关系的对象如果要强制进行转换那么就会出现 ClassCastException 异常。所以向下转型并不是一件安全的事情。**

1. **Instanceof关键字**

|  |  |
| --- | --- |
| **实例对象 Instanceof 类** | **判断对象是否属于这个类** |

* 1. 为了保证在向下转型的安全性，我们在必须在向下转型之前进行一个判断。

范例:

|  |
| --- |
| class Person {  public void println(){  System.out.println("我是person类");  }  }  class SupperMan extends Person{  public void fly(){  System.out.println("我是SupperMan：我可以飞");  }  public void fir(){  System.out.println("我是person类：我可以喷火");  }  }    public class JavaDemo4{  public static void main(String[] args) {  Person per = new SupperMan();  System.out.println(per instanceof Person); //true  if (per instanceof Person) {  SupperMan sup = (SupperMan)per;  sup.fly();  sup.fir();  }  }  } |