2、具体内容

多态性是面向对象中的第三大主要特征，多态性是在继承性的继承之上扩展出来的概念，也就是说可以实现父子类之间的互相转换处理。

■多态性的基本概念

在Java之中对于多态性有两种实现的模式：

• 方法的多态性：

|- **方法的重载**：同一个方法名称可以根据传入的参数的类型或个数的不同实现不同功能的执行；

|- **方法的覆写**：同一个方法可能根据使用子类的不同有不同的实现；

方法的重载：	**方法的覆写：**

class Message
{
public void **print**(){
System.out.println("www");
}
public void **print**(String str) {
System.out.println(str);
}
}
	class Message {
	public void **print**() {
	System.out.println("www.mldn.cn") ;
	}
	}
	class DataBaseMessage extends Message {
	public void **print**() {
	System.out.println("Oracle数据库连接信息....") ;
	}
	}
	public class JavaDemo {
	public static void main(String args[]) {
	DataBaseMessage msg = new DataBaseMessage() ;
	msg.print() ;
	}
	}

• 对象的多态性：父子实例之间的转换处理，它有两种模式：

|- **对象向上转型**：父类 父类实例 = 子类实例、自动完成转换；

|- 对象向上转型：子类 子类实例 = (子类)父类实例、强制完成转换

从实际的转型处理来讲，大部分情况下考虑最多的一定是对象的向上转型（90%）、对于对象的向下转型往往都在使用子类特殊功能（子类可以对父类进行功能扩充）的时候要采用向下转型（3%）、还有一些时候是不会考虑转型的（String类、7%）。

■对象向上转型（接收或返回参数的统一性）

对象转型的处理属于多态性，而这一特性必须在继承性的基础上实现。

范例：观察一个简单代码

```
class Message {
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
class DataBaseMessage extends Message {
	public void print() {
		System.out.println("Oracle数据库连接信息....") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		DataBaseMessage msg = new DataBaseMessage() ;
		msg.print() ;
	}
}
```

本程序是一个最简单的方法覆写操作实现，整体的程序之中可以发现，由于现在实例化的是子类对象，并且子类对象覆写了父类中的print()方法，所以调用的是被覆写过的方法。

范例：观察向上转型

```
class Message {
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
class DataBaseMessage extends Message {
	public void print() {
		System.out.println("Oracle数据库连接信息....") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Message msg = new DataBaseMessage() ;  // 向上转型
		msg.print() ;
	}
}
```

那么这个时候就需要进行思考了，向上转型这种操作有什么主要的用处呢？

范例：向上转型

```
class Message {
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
class DataBaseMessage extends Message {
	public void print() {
		System.out.println("Oracle数据库连接信息....") ;
	}
}
class WebServerMessage extends Message {
	public void print() {
		System.out.println("WEB服务器连接信息....") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		fun(new DataBaseMessage()) ;	// Message msg = new DataBaseMessage()
		fun(new WebServerMessage()) ;	// Message msg = new WebServerMessage()
	}
	public static void fun(Message msg) {	// 不管现在传递是那一个子类都可以接收
		msg.print() ;
	}
}
```

向上转型的主要特点在于，可以对参数进行统一的设计。但是为什么此时不适用重载来解决当前问题呢？

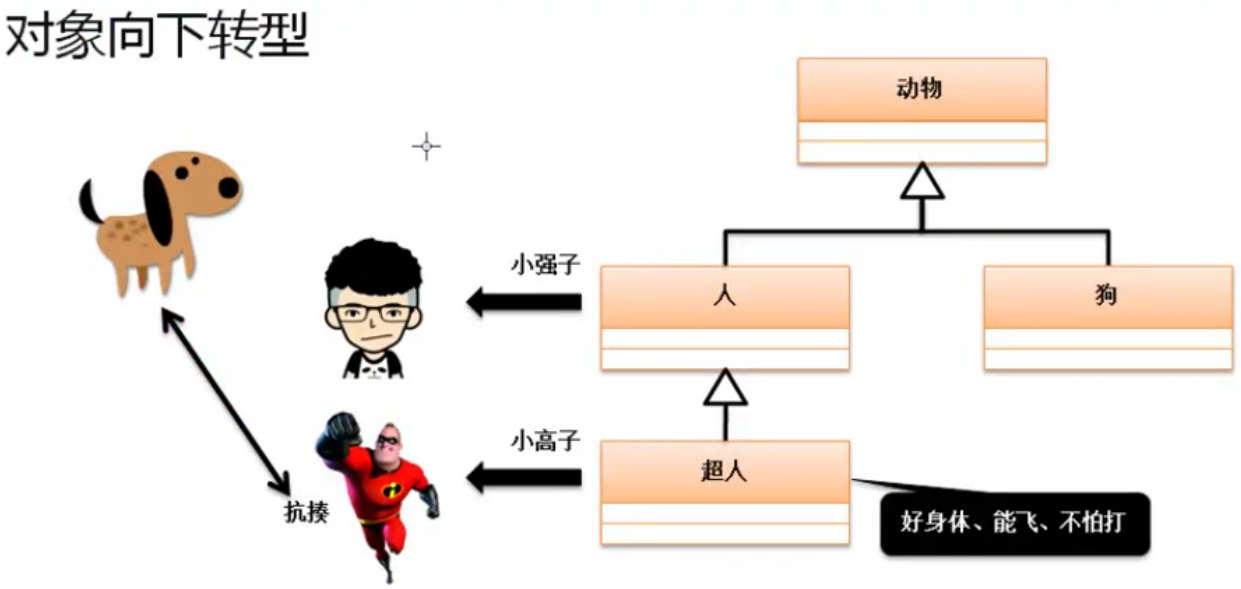
```
class Message {
	public void print() {
		System.out.println("www.mldn.cn") ;
	}
}
class DataBaseMessage extends Message {
	public void print() {
		System.out.println("Oracle数据库连接信息....") ;
	}
}
class WebServerMessage extends Message {
	public void print() {
		System.out.println("WEB服务器连接信息....") ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		fun(new DataBaseMessage()) ;	// Message msg = new DataBaseMessage()
		fun(new WebServerMessage()) ;	// Message msg = new WebServerMessage()
	}
```

```
    public static void fun(DataBaseMessage msg) {    // 不管现在传递是那一个子类都可以接收
        msg.print() ;
    }
    public static void fun(WebServerMessage msg) { // 不管现在传递是那一个子类都可以接收
        msg.print() ;
    }
}
```

现在的操作利用了重载解决了当前的设计，的确可以实现与之前完全一样的效果。但是在进行程序类设计的时候除了满足于当前的要求之外，还需要做出可维护性的设计，如果说现在随着项目的发展，Message产生了3W个子类，那么这个时候每当扩充一个Message子类之后就需要追加一个fun()的方法重载，这样就对程序的维护性造成很大影响。

■对象向下转型

向下转型主要特点在于需要使用到一些子类自己特殊的定义处理。



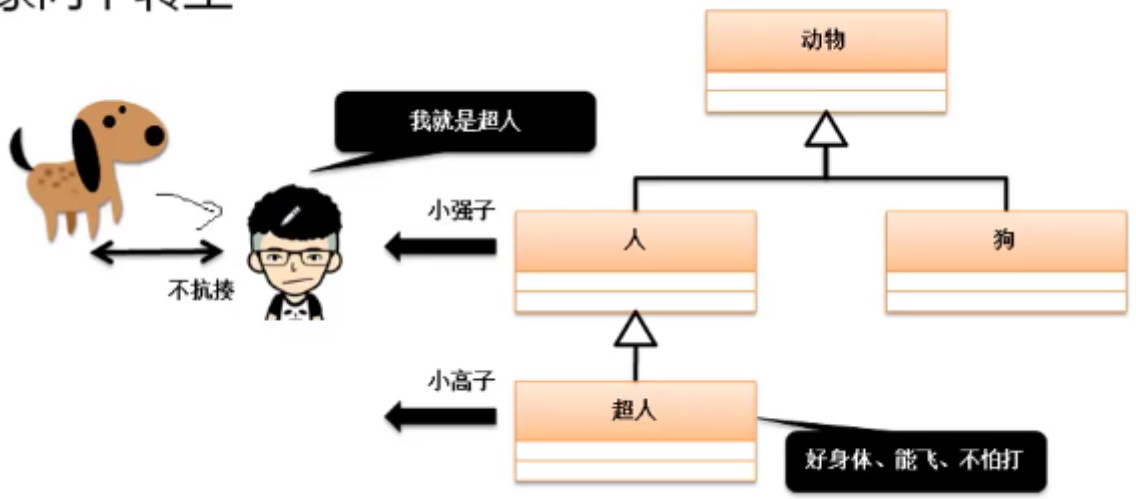
范例：实现向下转型

```
class Person {
    public void print() {
        System.out.println("一个正常的人类行为，吃饭、睡觉、繁衍。") ;
    }
}
class SuperMan extends Person {
    public String fly() {
        return "我可以飞。。。" ;
    }
    public String fire() {
        return "我可以喷火。。。" ;
    }
}
public class JavaDemo {
```

```
    public static void main(String args[]) {
        System.out.println("---------- 正常状态下的超人应该是一个普通人的状态 ----------") ;
        Person per = new SuperMan() ; // 向上转型
        per.print() ;
        System.out.println("---------- 外星怪兽狗骚扰地球，准备消灭人类 ----------") ;
        SuperMan man = (SuperMan) per ;             // 向下转型
        System.out.println(man.fly()) ;
        System.out.println(man.fire()) ;
    }
}
```

向上描述的是一些公共的特征，而向下描述的是子类自己特殊的定义环境，但是需要明确的是，向下转型并不是一件安全的事情。因为在向下转型之前一定要首先发生向上转型。

对象向下转型



范例：观察错误的程序

```
class Person {
    public void print() {
        System.out.println("一个正常的人类行为，吃饭、睡觉、繁衍。") ;
    }
}
class SuperMan extends Person {
    public String fly() {
        return "我可以飞。。。" ;
    }
    public String fire() {
        return "我可以喷火。。。" ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        System.out.println("---------- 正常状态下的人应该是一个普通人的状态 ----------") ;
        Person per = new Person() ; // 不转型
        per.print() ;
        System.out.println("---------- 外星怪兽狗骚扰地球，准备消灭人类 ----------") ;
        SuperMan man = (SuperMan) per ;      // SuperMan类与Person类
```

```
    }
}
```

```
Exception in thread "main" java.lang.ClassCastException: Person cannot be cast to SuperMan
```

以后只要是发生对象的向下转型之前一定要首先发生向上转型，两个没有任何关系的实例如果要发生强制转换，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。

■instanceof关键字

通过分析可以发现向下转型本身是一件存在有安全隐患的操作，所以为了保证向下转型的正确性，往往需要在进行转型之前进行判断，判断某个实例是否是某个类的对象，这个就需要通过instanceof语法来实现，语法如下：

```
对象 instanceof 类
```

该判断将返回一个boolean类型，如果是true表示实例是指定类对象。

范例：观察instanceof的使用

```
class Person {
    public void print() {
        System.out.println("一个正常的人类行为，吃饭、睡觉、繁衍。");
    }
}
class SuperMan extends Person {
    public String fly() {
        return "我可以飞。。。";
    }
    public String fire() {
        return "我可以喷火。。。";
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Person per = new Person(); // 不转型
        System.out.println(per instanceof Person);  // true
        System.out.println(per instanceof SuperMan);    // false
    }
}
```

范例：观察instanceof关键字

```
class Person {
    public void print() {
        System.out.println("一个正常的人类行为，吃饭、睡觉、繁衍。");
    }
}
class SuperMan extends Person {
    public String fly() {
```

```
        return "我可以飞。。。" ;
    }
    public String fire() {
        return "我可以喷火。。。" ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Person per = new SuperMan() ; // 向上转型
        System.out.println(per instanceof Person) ;   // true
        System.out.println(per instanceof SuperMan) ;     // true
    }
}
```

所以在日后进行项目的开发过程之中，如果要进行对象的向下转型，最好先判断一次。

```
class Person {
    public void print() {
        System.out.println("一个正常的人类行为，吃饭、睡觉、繁衍。") ;
    }
}
class SuperMan extends Person {
    public String fly() {
        return "我可以飞。。。" ;
    }
    public String fire() {
        return "我可以喷火。。。" ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Person per = new SuperMan() ; // 向上转型
        if (per instanceof SuperMan) {
            SuperMan man = (SuperMan) per ;
            System.out.println(man.fly()) ;
            System.out.println(man.fire()) ;
        }
    }
}
```

在以后进行一些完整性的程序开发的过程之中，对于转型之前一定要使用instanceof判断。