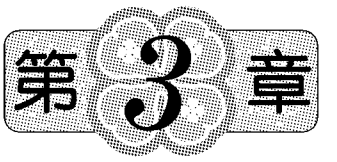

面向对象程序设计

3.1 面向对象程序设计的基本概念

ActionScript 是典型的面向对象的编程语言。前面已经多次使用了它的内置类，使用了这些类的属性、方法和事件，这些都是面向对象编程的概念。接下来，对面向对象程序设计（Object-Oriented Programing，OOP）进行更加深入的学习和应用，主要是学习如何创建自定义类以及在自定义类中实现各种面向对象程序设计的概念。这对理解游戏程序、组织复杂应用都有较大的作用。事实上，面向对象的程序设计思想贯穿于分析、设计、编码、调试的整个软件开发过程，是高效开发复杂软件、实现代码重用和便利维护的重要保障。

那么，什么是面向对象？面向对象是指用对象的观点去分析和思考问题，简而言之，就是将现实问题分解为一个个相对应的对象，每个对象为外界提供了一些必需的服务，至于这个服务的具体实现，则与外界无关，对象之间通过消息机制传递信息，彼此合作，从而完成整体的功能。

面向对象方法是针对传统的结构化程序设计语言无法解决所谓软件危机而提出的新的语言、工具和方法。面向对象思想是程序设计思想的一次飞跃。目前大多数程序设计语言都采用了面向对象的方法。这些语言提供的功能不尽相同，但基本上都满足面向对象的一些特性，如封装、继承和多态。

下面介绍面向对象程序设计中的几个基本概念。

1. 对象

对象（Object）是系统中用来描述客观事务的一个实体，它是构成系统的一个基本单元。对象由一组属性、行为（方法）构成。属性是用来描述对象静态特征的数据项，行为是用来描述对象动态特征的操作序列。另外，消息传递机制则表现为对象所能够处理的一组事件。

2. 类

类（Class）为属于该类的全部对象提供了抽象的描述，其内部主要包括属性和行为两个部分。属于某类的对象称为该类的一个实例。例如苹果 A、苹果 B 两个对象，都属于苹果这个类。

3. 封装

封装是面向对象方法的一个重要原则，就是把对象的属性和行为（也称为数据和基于数据的操作）结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节。封装有两个含义：第一个含义是把对象的全部属性和行为结合在一起，形成一个不可分割的独立单元，这就是高内聚；第二个含义也称做“信息隐蔽”，即尽可能隐蔽对象的内部细节，内部的实现尽量与外部无关，只保留有限的对外接口与外部发生联系，这就是低耦合。好的封装就是高内聚低耦合的封装。

4. 继承

继承是面向对象技术能够提高软件开发效率的重要原因之一。其定义是：特殊类的对象拥有其一般类的全部属性与行为，称做特殊类对一般类的继承。例如 Flash 中的 MovieClip 类的继承关系为：

MovieClip → Sprite → DisplayObjectContainer → InteractiveObject → DisplayObject → EventDispatcher→Object

MovieClip 类拥有 Sprite 类的所有属性和行为，而 Sprite 类拥有 DisplayObjectContainer 类的所有属性和方法，以此类推。

> 注 Object 类是所有类的基类。

5. 多态

多态性是指在一般类中定义的属性或行为，被特殊类继承之后，可以具有不同的数据类型和行为。这使得同一个属性和行为在一般类及其各个特殊类中具有不同的语义。假设矩形类和圆形类都继承自图形类，它们都有同一个继承自图形类的画线框的方法 draw()，但在矩形类对象和圆形类对象中使用 draw()方法的结果显然不同。事实上，是图形类的 draw 函数在矩形类和图形类定义中被重写了，也就是两者所调用的代码可以完全不同，只是名称相同而已。这就是多态。

6. 事件

事件就是面向对象编程中的消息传递，它是触发对象执行某些行为的原因。例如气球被刺而破，“气球”是对象，“破”是行为，“刺”就是事件。继承自 InteractiveObject 的类都具有接收事件的能力。常用的事件有鼠标事件、键盘事件、程序事件和定时触发事件。行为即是事件处理函数。

3.2 自定义类的实现

现在已经对类的使用非常熟悉了，接下来讲解如何在 ActionScript 中创建自定义类，以及在自定类中实现各种面向对象的概念。

值得一提的是，Flash 的内置类以及可视化的动画编辑环境已经提供了比较完善的游戏

创作基础，如果不是开发复杂的游戏，或者需要特殊的数据类型，则没有必要创建自定义类。直接使用面向过程的开发方法快速获得目标成果是 Flash 的一大优势。

ActionScript 实现自定义类的方法经过发展，有多种途径来实现，而且不同版本的 ActionScript 的方法不同。例如 AS1 常用的是原型属性（prototype）继承的方式来实现自定义类，甚至使用函数嵌套的方法来实现；AS2 增加了 class 类定义方法；而在 AS3 中，则又提供了一套更为完善的 class 类定义方法。虽然它仍然支持原型属性继承，但这样只是为了与前期版本兼容。所以使用 AS3 的类定义方法是大势所趋，这里仅介绍该版本标准的自定义类方法。

3.2.1　创建类的语法

创建类的语法如下：

```
class 类名称[ extends 超类]{
类成员
}
```

类的成员包括类的构造函数、属性和方法。

下面的代码创建一个机动车类。机动车成员如表 3-1 所示。

表 3-1　机动车成员

成　　员	说　　明
name：String	名称
numberPlate：String	车牌号
drive()	开启
stop()	停车

① 按 Ctrl+N 组合键，弹出“新建”对话框，选择“ActionScript 文件”选项，单击“确定”按钮，ActionScript 文件就是文本格式的类文件代码，Flash CS3 的界面变成一个输入和编辑代码的主窗口，不再有时间轴、舞台等概念。

② 在主窗口中输入以下代码：

```
class Vehicle {
    var name:String;
    var numberPlate:String;
    function drive() {
        trace("车开动了!");
    }
    function stop() {
        trace("车停止了!");
    }
}
```

③ 保存类文件：按 Ctrl+S 组合键保存文件，文件名为 Vehicle.as。自定义类必须放在.as 文件中，并且以类名作为文件名，包括大小写也必须相同，否则此类无效。

这样就完成了最初的一个自定义类。但是这样的自定义类无法在 Flash 文档中被引用，

因为它没有声明任何命名空间，Flash 文档无法感知它的存在，只有同一类包中的其他类能够引用到这个类。这就好像它是某个类包中的私有成员一样。为了能够在 Flash 文档中测试这个类的有效性，需要给它声明一个命名空间，使用 package 关键字是最常用的声明命名空间的方法。

将以上代码修改为：

```
package {
    public class Vehicle {
        public var name:String;
        public var numberPlate:String;
        public function drive() {
            trace(name+"开动了!");
        }
        public function stop() {
            trace(name +"停止了!");
        }
    }
}
```

按 Ctrl+S 组合键保存修改。

这里使用 package 关键字表示将此类“打包”到顶级路径下，然后在 Vehicle.as 的同一目录中新建 Flash 文档（AS3），并在动作面板中输入以下代码：


```
var myVehicle:Vehicle=new Vehicle();
myVehicle.name="标致 307";
myVehicle.drive();
myVehicle.stop();
```

保存文档，然后按 Ctrl+Enter 组合键测试影片，将得到如下输出：

```
标致 307 开动了!
标致 307 停止了!
```

代码解释

编译器发现有类声明时，如本例第一行代码，就会在类路径下寻找这个类的定义。默认的类路径地址为：程序安装目录\zh_cn\Configuration\ActionScript 3.0\Classes

Flash 文档的所在目录也是一个类路径，还可以添加或删除多个类路径。编译器在当前目录下发现 package 包，在里面找到类定义 Vehicle，于是根据代码声明创建 Vehicle 类的实例。之后的操作和操作系统自带的类完全一致，不再赘述。

3.2.2 构造函数

类的构造函数是一种特殊的函数，在使用 new 关键字创建类的实例时自动调用这个函数。构造函数的名称与包含它的类的名称相同。

例如为上面的 Vehicle 类添加构造函数，如代码中的黑体部分：

```
package vehicle{
    public class Vehicle {
```

```
        private var name:String;
        private var numberPlate:String;
        function Vehicle(nameValue:String,numberPlateValue:String) {
            name=nameValue;
            numberPlate=numberPlateValue;
            trace("车辆的名称是：" + name + "\n 车牌号是：" + numberPlate);
        }
        public function drive() {
            trace(name + "开动了!");
        }
        public function stop() {
            trace(name + "停止了!");
        }
    }
}
```

保存 Vehicle.as 文件于工作目录下的 vehicle 文件夹，并在工作目录下新建 Flash 文档 test.fla，然后输入以下代码：

```
import vehicle.Vehicle;
var myVehicle1:Vehicle=new Vehicle("东风标致 307","苏 E7CXX");
var myVehicle2:Vehicle=new Vehicle("江淮汽车","京 A5KXX");
```

按 Ctrl+Enter 组合键测试影片，会得到如下输出：

```
车辆的名称是：东风标致 307
车牌号是：苏 E7CXX
车辆的名称是：江淮汽车
车牌号是：京 A5KXX
```


可见构造函数被自动调用了。

3.2.3　包与目录、类路径的关系

package 的作用是指定类的命名空间。只有包含在 package 内的成员，才能被 Flash 文档使用。否则只能被同一命名空间的其他类使用。命名空间是用来控制所创建的属性、方法以及类等的可用性，并避免命名冲突。可以有多种方法定义和使用命名空间，package 是其中的一种。例如下面代码指定 Vehicle 类在 vehicle 命名空间中：

```
package vehicle
{
public class Vehicle{
//以下代码略
```

使用这些包中的类，需要在程序中显式导入（import）这些包，或者使用这些类时带上包前缀，这和 Flash CS3 自带的包是一样的用法。

package 可以嵌套，就像文档目录一样。事实上，package 结构就是类文件保存的目录结构。例如某个大型的应用程序，不仅有车辆以及它的派生类，还有各种与人员相关的类。把所有的车辆类存放在 vehicle 文件夹中，把所有与人员相关的类组织在 people 文件夹中，

如图 3-1 所示。

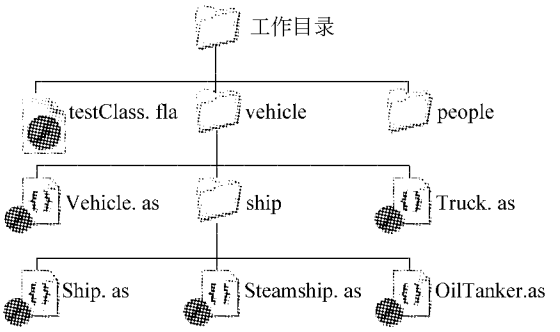

图 3-1　某工程的类文件组织方式

则 Vehicle、Truck 类都应该打包在 vehicle 包中，而 Ship、Steamship、OilTanker 类则应该打包在 vehicle.ship 包中。例如 OilTanker 类的开头部分应该如下：

```
package vehicle.ship
{
public class OilTanker{
//以下代码略
```

注意　这种目录结构，与类的继承关系不相关，例如 Truck 类可能继承自 Vehicle 而 Ship 类可能直接继承自 Object。

打包好这些类之后，将包放在类路径中。可以自行设置新的类路径，方法为选择【编辑】|【首选参数】选项，在弹出的设置面板中选择 ActionScript 选项，再单击“ActionScript 3.0 设置”按钮，单击“+”按钮，输入自定义的类路径即可。如图 3-2 所示，添加了一个新的类路径 C:/OpenFlashClasses，将自定义的类包放在这个路径之下，就可以被任何 Flash 文档使用。

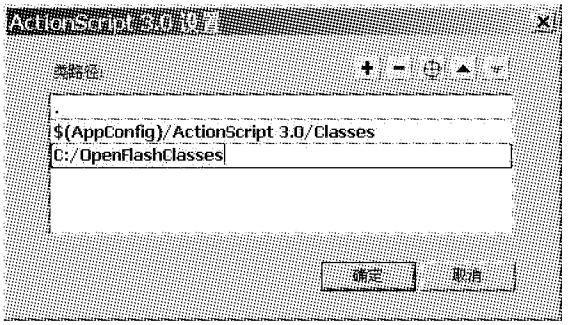

图 3-2　添加类路径

提示　发布 Flash 作品时不需要将这些类一起发布，因为文档使用到的类代码已经被自动打包在 SWF 文件中了。

3.2.4　访问控制关键字

用 public 标识的类或者类成员具有公共的访问权限，换句话说，标识为 public 的类或者类成员在任何位置可见。相对的，用 private 标识的类成员，只在类内部可见。这些控制类或者类成员的可见性的关键字被称为是访问控制关键字，如表 3-2 所示。

表 3-2　访问控制关键字

关　键　字	说　　明
internal	指定类、变量、常量或函数可用于同一包中的任何调用者
private	指定变量、常量、方法或命名空间仅可供定义它的类使用
protected	指定变量、常量、方法或命名空间只可用于定义它的类及该类的任何子类
public	指定类、变量、常量或方法可用于任何调用者

一个.as 文件可以包含多个类，但是只允许一个类声明为 public 访问权限。并且用这个类名作为.as 文件的文件名。其他类则作为包内部使用的类存在，Flash 文档无法访问到这些类。

默认情况下，类或成员使用 internal 访问权限。

表 3-2 中的关键字也被称为属性修饰符。除了这些修饰符外，ActionScript 还可以使用表 3-3 所列的属性修饰符。

表 3-3　属性修饰符

关　键　字	说　　明
dynamic	指定类的实例可具有在运行时添加的动态属性
final	指定不能覆盖方法或者不能扩展类
native	指定函数或方法由 FlashPlayer 以本机代码的形式实现
override	指定用一种方法替换继承的方法
static	指定变量、常量或方法属于类，而不属于类的实例

这些修饰符可以和访问控制关键字组合使用。

3.3　面向对象概念在 ActionScript 中的实现

3.3.1　封装

在完美的面向对象设计中，对象被看作包含（或封装）功能的“黑匣子”。程序员应当能够在仅知道对象的编程接口（属性、方法、接口和事件）的情况下无障碍地使用它们，至于其内部实现，程序员无需了解。封装使程序员可以在更高的抽象层次上考虑问题，并能提供可用于构建复杂系统的组织框架。

为了实现封装，ActionScript 用 private 将一些不让或者不需要外部对象访问的变量和函数设为私有。例如上例中的 name 和 numberPlate 就是私有变量，来自外部的访问将出错，

例如将 test.fla 的脚本修改为：

```
import vehicle.Vehicle;
var myVehicle:Vehicle=new Vehicle("东风标致 307","苏 E7CXX");
myVehicle.name="江淮汽车";
myVehicle.numberPlate="京 A5KXX";
```

对于构造函数来说，由于函数在 Vehicle 类内部，所以可以自由使用 name 和 numberPlate 属性，但是对于 test.fla 文档来说，Vehicle 类的 name 和 numberPlate 属性是不可见的，无法调用，如上面的例子，将无法编译通过。于是实现了封装的目的。

注意 如果没有添加访问控制修饰符，则 AS2 默认的修饰符是 public，而 AS3 默认的修饰符是 internal。

一个典型的封装的例子是将一些属性设为私有，然后通过 setter 和 getter 方法来存取这个属性，这样做的好处是当这个属性变量名称改变或者存取这个属性需要进行联动操作时，无须修改每处存取这个变量的代码而只要修改一下 setter 和 getter 函数即可。这两个函数比较特殊，在类内部是一个函数，但是在实例调用时，又以属性的方式被调用。

用下面的代码替换原来的 Vehicle 类代码。

```
package vehicle{
    public class Vehicle {
        private var _name:String;
        private var _numberPlate:String;
        private static  var _COUNT:Number = 0;
        public function Vehicle(nameValue:String, numberPlateValue:String) {
            _name = nameValue;
            _numberPlate = numberPlateValue;
            _COUNT++;
            trace("车辆的名称是："+_name+"\n 车牌号是："+_numberPlate);
        }
        public static function get COUNT ():Number {
            return _COUNT;
        }
        public function get name() {
            return _name;
        }
        public function set name(a:String) {
            _name = a;
        }
        public function get numberPlate():String {
            return _numberPlate;
        }
        public function set numberPlate(a:String) {
            _numberPlate = a;
        }
        public function drive() {
            trace(_name + "车开动了!");
        }
```

```
        public function stop() {
            trace(_name + "车停止了!");
        }
    }
}
```

按 Ctrl+S 组合键保存。这段代码多了一个静态属性_COUNT，用来表示车辆实例的总数，当用构造函数创建一个新的 vehicle 实例时会自动加 1。然后将_COUNT 以及原来的两个属性都改成用 setter 和 getter 存取方式。由于 COUNT 只有 getter 函数，所以对于类外部来说，它就成为只读的属性了。

修改 test.fla，输入以下代码：

```
import vehicle.Vehicle;
trace(Vehicle.COUNT);
var myVehicle1 = new Vehicle("东风标致 307", "苏 E7CXX");
trace(Vehicle.COUNT);
var myVehicle2 = new Vehicle("江淮汽车", "京 A5KXX");
trace(Vehicle. COUNT);
trace(myVehicle2.numberPlate);
myVehicle2.numberPlate = "沪 A703XX";
trace(myVehicle2.numberPlate);
```

测试影片，得到如下输出：

```
0
车辆的名称是：东风标致 307
车牌号是：苏 E7CXX
1
车辆的名称是：江淮汽车
车牌号是：京 A5KXX
2
京 A5KXX
沪 A703XX
```


注意　静态成员是所有类实例共用的，调用方法是使用类名前缀而不是实例名前缀。

3.3.2　继承

现在假设需要更细化车辆类。这个项目中有货车和客车。货车还有吨位属性，而客车还有座位数属性。货车直接采用车辆的开车方式，客车在车辆开车方式的基础上，还附加了自己的开车方式。

这里需要用到继承和多态两个面向对象程序设计的特色。通过继承，货车和客车都可以拥有车辆的所有属性和方法，改善了代码的重用性，极大地提高工作效率。在子类中可以添加新的属性或方法，也可以将原有的属性或方法重写，使与基类同名的属性或方法具有不同的含义。

本例的继承关系如图 3-3 所示。

为了实现上面提到的功能，需要使用继承和多态两个概念，那么具体如何实现？

继承的语法如下：

```
class 子类名 extends 超类名 {
类成员
}
```

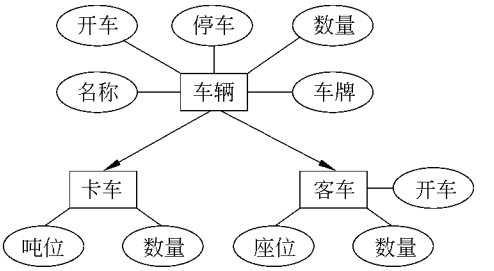

图 3-3　车辆类的继承关系

通过继承和重写，希望 Truck 拥有如表 3-4 所示的成员。

表 3-4　Truck 拥有的成员

成 员 名 称	说　　明
name:String	名称
numberPlate:String	车牌号
drive()	启动
stop()	停车
tonnage:Number	吨位
Truck. COUNT	Truck 实例的数量（只读）
Vechile. COUNT	所有车辆的数量（只读）

创建新的.as 文件，输入以下代码：

```
package vehicle{
    public class Truck extends Vehicle {
        private var _tonnage:Number;
        private static  var _COUNT:Number = 0;
        public function Truck(nameValue, numberPlateValue, tonnageValue) {
            super(nameValue, numberPlateValue);
            _tonnage = tonnageValue;
            _COUNT++;
        }
        public function get tonnage():Number {
            return _tonnage;
        }
        public static function get COUNT():Number {
            return _COUNT;
        }
        public function set tonnage(a:Number) {
            _tonnage = a;
        }
        //可以添加更多的属性和方法
    }
}
```

按 Ctrl+S 组合键将代码保存在 vehicle 目录下，文件名为 Truck.as。