文章编号:1009-2552(2017)06-0052-06 DOI:10.13274/j.cnki.hdzj.2017.06.012

基于面向对象编程的代码复用技术

杨 浩[1],杨陟卓[2]

(1. 榆林职业技术学院,陕西 榆林 719000; 2. 山西大学计算机信息与技术学院,太原 030006)

摘 要:程序代码复用是软件开发中的关键技术,充分利用代码复用技术不仅可以降低程序编写成本,更重要的是可以提高程序执行效率。文中从类的封装、继承、多态性、重载等核心技术入手,通过抽象类、虚方法、类的转换、接口、接口和类的转换、覆盖和多重继承等技术,以Delphi面向对象的程序设计为例,研究如何在具有相同基类的一组派生类及不具有相同基类的一组派生类等几种情况下实现代码复用,并通过实例验证了代码复用的可行性。

关键词:面向对象编程;代码复用;封装;多态;接口

中图分类号:TP311.1 **文献标识码**:A

Technology of code-reuse based on object oriented programming

YANG Hao[1], YANG Zhi-zhuo[2]

(1. Yulin Vocational and Technical College, Yulin 719000, Shaanxi Province, China;
2. School of Computer Information Technology, Shanxi University, Taiyuan 030006, China)

Abstract: Program code reuse is a key technology in software development, making full use of code reuse technology can not only reduce the cost of programming, but also can improve the efficiency of program execution. Taking Delphi object oriented programming as an example, proceed with key characteristic of class, encapsulation, inheriting, polymorphism and overload, using the technology of abstract class, virtual method, class conversion, interface, interface and class conversion, coverage and multiple inheritance, this paper presents how to carry out code reuse when following situation appeared: a group of derived classes that have or not the same base class. It also verifies the feasibility of code reuse through the relevant examples.

Key words: object oriented programming; code-resuse; encapsulate; polymorphism; interface

0 引言

代码复用是程序编写中的一项非常重要的技术,特别是在面向对象的编程中,可以说科学、合理的代码复用是评价程序质量的重要指标之一。然而,仍有一些非常高效的代码复用技术,由于所涉及的技术难度较大,而不易被程序员所掌握、应用,对代码复用水平还停留函数(或过程)调用的较低层次(大多数情况是拷贝代码)。由于不能深入到面向对象编程的内部技术,巧妙利用其强大功能和核心机制进行代码复用,致使所编写的软件急剧膨胀,执行性能直线下滑。本文以Delphi面向对象的编程为例,从抽象类、虚方法、类转换、接口、接口和类转换、覆盖及多重继承等技术方面揭示了面向对象编程了代码服用的核心机制,全面剖析了面向对象编程中的代码复用技术,并进行了仿真实验。

1 利用多态、覆盖机制实现代码复用

1.1 实现思想及示例

类的多态性实现原理依赖于类的抽象方法和虚方法,同时也和类的继承密切相关。因此通常先定义一个底层的对象,然后将其中的某些方法定义为抽象方法(抽象方法首先必须是虚方法或动态方

收稿日期:2016-05-23
基金项目:国家自然科学基金(61502287);榆林市科技计划项目(2014CXY-02)
作者简介:杨浩(1977-),硕士研究生,高级工程师,研究方向为信息化教学,工程优化。

法)，也就是说，只是定义了接口，而没有定义具体的实现细节。照这样的思路，可以通过定义多个派生对象，然后在这些对象中实现祖先类中未曾真正实现的细节，这样，先前定义的那些底层类就具有了多态的特性。这种机制的好处是，在使用这些类的时，只需定义一套代码(一般把该段代码编成一个函数或过程)，就可以实现多种功能(各功能通过覆盖技术实现)。而唯一需要修改的就是创建对象实例的那一部分代码[1]。

Delphi 面向对象的程序设计中，一般把基类中虚(Virtual)方法、动态(Dynamic)方法声明为抽象(Abstract)方法并作为接口留给派生类去实现，此时，基类也就相应地变成了抽象基类，然后利用覆盖(Override)技术和晚绑定实现机制即可实现多态[2]。一般不能将特定类型的值赋予不同类型的变量。但是，在面向对象的程序里有个例外。那就是允许将一个派生类的值赋给一个类型为基类的变量，但是反过来却是不行的。这对通过多态技术实现代码复用是很关键的，这在面向对象的程序设计中被称为类的类型转换(向上转型)。下面通过示例程序来说明基于多态、覆盖技术的代码复用的实现过程。

示例引用的单元文件 Unit_Example(其中包含了类的定义和实现)如下:

```
unitUnit_Example;
interface
type
Tbook = class( TObject)
public
BkName: string;
PBHouse: string;
PBDate: string;
Athr: string;
constructor create; virtual;
function ShowIntro: pchar; virtual; abstract;  //定义的抽象方法，为多态作准备
end;
TFl_book = class( TBook)
public
    constructor create; override;
    function ShowIntro: pchar; override;  //用覆盖实现多态
end;
TD_book = class( TBook)
public
    constructor create; override;
    function ShowIntro: pchar; override;  //用覆盖实现多态
end;
implementation
constructor TBook. create;
begin
BkName: = 'Fl';
…    //初始化数据成员
end;
constructorTFl_book. create;
begin
inherited;
end;
  constructorTD_book. create;
begin
BkName: = 'delphi';
…    //初始化数据成员
end;
functionTFl_book. ShowIntro;
begin
result: = 'The bookFL mainly. . . ';
end;
functionTD_book. ShowIntro;
begin
result: = 'The book DELPHI mainly. . . ';
end; end.
```

其中，Fl 类和 TD_book 类是基类 Tbook 的两个派生类，现在通过类的多态性及相关技术，在具有同一基类的一组派生类(Fl 和 TD_book)中实现代码复用。其窗体代码文件为:

```
unit UFrmIntroduction;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, StdCtrls;
type
  TForm1  =  class( TForm)
    …
    procedureBT_1Click( Sender:  TObject) ;
    procedureBT_2Click( Sender:  TObject) ;
  private
    { 声明私有成员 }
  public
    { 声明公有成员 }
```

```
    end;
var
    Form1: TForm1;
implementation
usesUnit_Example;
procedure TForm1. FlClick( Sender: TObject) ;
var
    Fl: TFl_book;
begin
Fl: = TFl_book. create;
    Ed_1. Text: = Fl. BkName;
    Ed_2. Text: = Fl. Athr;
    Ed_3. Text: = Fl. PBHouse;
    Ed_4. Text: = Fl. PBDate;
    Ed_5. Text: = Fl. ShowIntro;
end; end;
procedure TForm1. DelClick( Sender: TObject) ;
var
    Del: TD_book;
begin
Del: = TD_book. create;
    Ed_1. Text: = Del. BkName;
    Ed_2. Text: = Del. Athr;
    Ed_3. Text: = Del. PBHouse;
    Ed_4. Text: = Del. PBDate;
    Ed_5. Text: = Del. ShowIntro;
     end; end; end.
```

运行结果如图 1 所示。

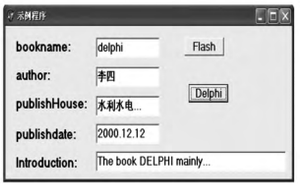

图 1　程序运行结果

1.2　代码改进方法

上面的代码固然能实现预期的功能，但没有实现代码复用，不是高质量的程序代码。不难发现按钮 Fl 和 Del 的 OnClick 事件中代码很相似，其不同点是：在按钮 Fl 的 Onclick 事件中程序访问的是类 TFl_book 的数据成员和方法，而在按钮 Del 的 OnClick 代码中程序访问的是类 TD_book 的数据成员和方法。由于类 TFl_book 和 TD_book 的父类都是基类 Tbook，由类的向上转型可知，可以通过定义一个函数 showIntroduction (book: Tbook) ，并用基类 Tbook 的实例 book 作为函数参数，即可实现代码的复用[3]。改进后的代码为：

```
    unit UfrmIntroduction;
    interface
    uses
      Windows, Messages, SysUtils, Variants, Clas-
ses, Graphics, Controls, Forms,
      Dialogs, StdCtrls,Unit_Example;
    type
      Tform1 = class( Tform)
       …
        procedure showIntroduction( book: Tbook) ;
        procedureFlClick( Sender: Tobject) ;
        procedureDelClick( Sender: Tobject) ;
      private
        { Private declarations }
      public
    { Public declarations }
      end;
    var
      Form1: Tform1;
    implementation
    procedure Tform1. showIntroduction ( book:
Tbook) ;
    begin
      Ed_1. Text: = book. BkName;
      Ed_2. Text: = book. Athr;
      Ed_3. Text: = book. PBHouse;
      Ed_4. Text: = book. PBDate;
      Ed_5. Text: = book. ShowIntro;
    end;
    procedure Tform1. FlClick( Sender: Tobject) ;
    begin
    showIntroduction( TFl_book. create) ;
    end;
    procedure Tform1. DelClick( Sender: Tobject) ;
    begin
      showINtroduction( TD_book. create) ;
    end;   end.
```

改进后，按钮 Fl 和 Del 的 OnClick 事件代码中只用一条语句即可完成与原代码同样的程序功能。这就是基于多态、覆盖技术的代码复用思想。

2　利用接口技术实现代码复用

在上述基于多态、覆盖技术的代码复用中，代码复用的前提是具有相同父类类(Tbook) 的一组派生

类(TFl_book 和 TD_book),虽然累的层次分明,但在一定程度上限制了多态的应用,继而限制了程序代码的复用潜力。因为在实际应用中,往往要在不具有相同父类的一组派生类中实现相似的程序功能,此时,必须借助接口技术来实现。

通过接口技术实现上例所述代码复用相对容易,再次不在啰嗦,下面以更复杂情况下的代码复用技术,来说明基于接口技术的代码复用思想。

2.1 接口概念

接口与类相似却又有不同,接口可包含方法和属性,却不能包含数据成员。接口的方法都是公有的(public),而且接口的方法不能有接口自己来实现,必须留给实现该接口的类来实现其方法。与C++不同,Delphi 可以继承多个基类,但不支持派生类,但依然可以用多接口继承实现类似于C++中多重继承功能。

2.2 接口引用技术及接口与类的转换

Delphi 允许接口类型的变量来引用任何实现该接口的实例,使得程序可以通过特定变量调用接口的方法,而无需知道接口功能的具体实现细节。但有两个限制条件:一是接口类型的表达式只允许访问接口定义的方法及属性,而不能访问实现接口类的其它任何成员;二是特定接口类型的表达式不允许引用实现了其派生类接口的类实例,除非这个类或其继承类实现了此祖先接口[4-5]。

2.3 实验示例

下面本文通过示例,来在一组不具有相同父类的派生类之间实现代码复用。首先把需要共享的程序代码定义为一个或多个接口的方法,这样,即可用该组类依次实现接口的方法,然后利用覆盖、重载等技术实现代码复用。

将上例中单元文件 Unit_Example 做如下调整,其中定义了接口:

```
unitUnit_Example;
interface
type
IIntro = interface          //定义一个接口
function showIntro( ) : pchar;
end;
Tbook = class( TInterfacedObject,IIntro)
public
BkName: string;
Athr: string;
PBHouse: string;
PBDate: string;
constructor create; virtual;
function ShowIntro: pchar; virtual; abstract;  //定义的抽象方法为多态作准备
end;
TFl_book = class( Tbook)
public
   constructor create; override;
   function ShowIntro: pchar; override;  end;  //用覆盖实现多态
TD_book = class( TBook)
public
constructor create; override;
function ShowIntro: pchar; override;     //覆盖实现多态 end;
TRoom = class( TInterfacedobject,IIntro)
public
  name: string;
  Size: string;
  position: string;
  constructor create; virtual;
  function showIntro: pchar; virtual; abstract;
end;
TOffice = class( TRoom)
public
 constructor create; override;
 function showIntro: pchar; override;
 end;
TClassroom = class( TRoom)
public
 constructor create; override;
 function showIntro: pchar; override;
 end;
implementation
constructor TBook. create;
begin
BkName: = 'Fl';
{数据成员初始化}
end;
  constructorTFl_book. create;
begin
inherited;
end;
  constructorTD_book. create;
begin
BkName: = 'Del';
```

```
{数据成员初始化}
end;
functionTFl_book. ShowIntro;
begin
result: = 'This book Fl mainly. . . ';
end;
functionTD_book. ShowIntro;
begin
result: = 'The bookDEL mainly. . . ';
end;
constructor TRoom. create;
begin
name: = ' office';
{数据成员初始化}
end;
constructor TOffice. create;
begin
inherited;
end;
constructor TClassroom. create;
begin
name: = ' classroom';
{数据成员初始化}
end;
function TOffice. showIntro;
begin
result: = ' This office is used as. . . ';
end;
 function TClassroom. showIntro;
begin
result: = ' This classroom is used as. . . ';
end; end.
```

在该单元文件中，类 Fl **和** TD_book **是基类** Tbook **的两个派生类，类** Toffice **和** Tclassroom **是基类** Troom **的两个派生类，现在通过接口技术在不具有相同基类的一组派生类(** Fl、TD_book、Toffice **和** Tclassroom**) 中实现代码共享。窗体文件示例如下:**

```
Unit UFrmIntroduction;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls,Unit_Example, ExtCtrls;
type
  TForm1 = class( TForm)
  …
    procedureFlClick( Sender: TObject) ;
    procedureDelClick( Sender: TObject) ;
    procedure OfficeClick( Sender: TObject) ;
    procedure classroomClick( Sender: TObject) ;
  private  //下列两个函数用重载实现代码复用
  procedure showIntroduction( book: Tbook) ; overload;
   procedure showIntroduction( room: TRoom) ; overload;
  end;
  var
    Form1: TForm1;
  implementation
  procedure TForm1. showIntroduction ( book: Tbook) ;
  begin
    Ed_1. Text: = book. BkName; //通过重载实现代码复用
    Ed_2. Text: = book. Athr;
    Ed_3. Text: = book. PBHouse;
    Ed_4. Text: = book. PBDate;
    Ed_5. Text: = book. ShowIntro;
  end;
  procedure Tform1. showIntroduction ( room: TRoom) ; //用重载技术实现代码复用
  begin
  Ed_6. Text: = room. name;
  Ed_7. Text: = room. Size;
  Ed_8. Text: = room. position;
  Ed_9. Text: = room. showIntro;
  end;
  procedure TForm1. FlClick( Sender: TObject) ;
  begin
  showIntroduction( TFl_book. create) ;
  //通过重载实现代码复用
  end;
  procedure TForm1. DelClick( Sender: TObject) ;
  begin
  showINtroduction( TD_book. create) ;
  //通过重载实现代码复用
  end;
  procedure TForm1. OfficeClick ( Sender: TObject) ;
  begin
```

```
showINtroduction( TOffice. create);    //通过重载实现代码复用
end;
procedure TForm1. classroomClick( Sender: TObject);
begin
  showIntroduction( Tclassroom. create); //通过重载实现代码复用
end; end.
```

这样,在任何需要该程序功能的编程场合,都可通过调用 showINtroduction( ) 函数实现特定接口方法。例如,上例中的四个按钮分别用一条语句就实现了相应的功能。由于使用了函数重载技术,系统会根据函数参数类型自动查找适合的函数入口,以保证函数的正确调用,这就是在一组不具有相同父类的派生类之间实现代码共享的思想。运行结果如图 2 所示。

图 2　程序运行结果

当然,也可以用类来实现本例所示的代码复用,但是由于 TBook 类和 TRoom 类之间没有任何实质性的关系,如果将一个派生类强加给这两个类,虽然实现了代码复用,但类的关系不伦不类,破坏了程序的易读性。

3　通过 DLL、COM/COM + 及封装技术实现软件复用

3.1　封装的概念

封装也是面向对象编程中的一项非常重要的技术,它隐藏了与用户无关的内部技术的复杂性。这样,无论所使用的对象如何复杂,从使用者的角度看来,都是一个易操作的“黑盒子”。该“黑盒子”对外提供一个公共接口,通过该接口即可轻松实现组件化的代码复用。

3.2　用 DLL 和 COM/COM + 实现代码复用的思想

在面向对象的程序设计中,当特定对象的调用者利用预定义的接口关联到某对象的服务或数据时,无须知道该服务的具体实现细节,即用户使用对象时,不必知道对象内部是如何运行的。这样,新的应用就可以使用已开发软件中使用的对象,减少了类似程序功能在新项目中分析、设计和编程的工作量,从而实现了代码的复用[6]。

DLL 或 COM/COM + 是程序功能相对独立的一些特定的程序模块,其中被封装的程序代码可以是类、也可以是软件包(或类的集合) ,核心思想是通过接口来实现行对象之间的互相调用。用 DLL 或 COM/COM + 实现代码共享的应用范围非常广泛,尤其在在. NET 平台上可以完美地实现跨语言、跨平台的软件共享。有关利用 DLL 或 COM/COM + 技术实现代码复用的书籍很多,这里不在详说。

4　用类引用作为参数实现代码复用

在 Delphi 等诸多面向对象的程序设计语言中,对象的引用如同指针,指向的是一个类对象在内存中的地址。在软件设计的工程实践中,对象的引用也非常普遍,但是,类引用却并不多见。其实类引用技术不但能提高编程的灵活性,而且更已于实现代码复用,进而减少编程工作量,提高软件开发效率[7-8]。

设 TcontrolClass 为一个类引用,它指向所有控件类的父类(TControl) :

```
TcontrolClass = class of Tcontrol;
```

现在要动态创建一个控件的实例,但本文无法预料到程序运行到什么时候需要创建什么样的控件。这时,可以通过把类引用作为函数参数,然后在程序运行中动态调用该函数,来实现上述程序功能。如可以定义如下过程:

```
procedure CreateControl( control: TcontrolClass);
begin
{ 创建控件实例的代码}
end;
```

只要在需要创建特定控件的时候调用 CreateControl(control: TcontrolClass) 过程,并用具体的控件类替换类引用参数 control 即可。这样不仅提高了程序编写的灵活性,而且实现了代码的复用,省去了编写几十行甚至上百行代码工作量,缩小了程序规模。

5　结束语

软件规模越来越大,代码复用在软件架构中显得越来越重要,本文全面剖析了具有相同父类的一组派生类的代码复用技术、不具有相同父类的一组派生类的代码复用技术、跨语言、跨平台的代码复用技术、动态创建控件的代码复用技术等,并通过程序示例验证了上述代码复用技术的可行性。这些代码复用技术可为程序员编写出更加精悍的程序助一臂之力。

(下转第 61 页)

3.2 识别率统计

为了全面的验证系统的语音识别率，实验选择在一个普通布置的家庭中进行测试。考虑到实验次数、实验距离、语音命令长度等因素对实验结果的影响，测试次数设为 500 次，测试距离分为 1 ~ 5m 五个距离级别，每级测试 100 次，每个距离级别下对长度为 1、2、3、4 的语音命令分别进行 25 次测试。最终统计结果如下，其中测试距离与识别率统计结果如表 2 所示，语音命令长度与识别率统计结果如表 3 所示。

表 2 测试距离与识别率

每个距离级别 100 次，共计 500 次				
距离/m	识别	未识别	错误	识别率
1	96	3	1	96.0%
2	93	4	3	93.0%
3	89	8	3	89.0%
4	88	10	2	88.0%
5	85	11	4	85.0%

表 3 语音命令长度与识别率

每级长度测试 125 次，共计 500 次				
命令长度	识别	未识别	错误	识别率
1	122	2	1	97.6%
2	119	5	1	95.2%
3	111	10	4	88.8%
4	99	19	7	79.2%

3.3 统计结果分析

通过统计数据可知，该系统识别率与测试距离成负相关关系，3m 距离是用户日常使用范围，识别率为 89%，在 5m 范围内综合识别率可达 85%，完全可以满足用户日常使用电视的需求。语音命令的长度与最终识别率也有很大的关系，当命令长度为 1 时，识别率高达 97.6，但随着长度的增加，识别率急剧下降，当命令长度为 4 时，识别率只有 79.2%，识别率较低影响使用效果。因此在使用系统时，应尽量设置长度较短的关键词，另外也可以使用多对一的设置方法，即使用多个关键词与一个控制指令建立对应关系，这样可以有效增大系统的语音识别率。

另外，在测试过程中发现声音大小，使用方向对识别率也有较大影响。声音越大，系统接收到的信号信噪比也就越大，语音命令的识别率越高；用户正对系统发出命令，声波直接输入 MIC，与加大声音原理相似，也有助于提升识别率。因此在实际使用时，为获得较好的控制效果，应尽量正对系统并适当提高音量。

4 结束语

本文介绍的智能电视的非特定语音控制系统，采用 STM32 单片机和 LD3320 专用语音识别芯片相结合的方式，实现了对电视的非特定语音控制，具有结构简单、稳定可靠、使用方便等优点，在 5m 范围内实现 85% 的识别率。另外系统的移植性很强，可以通过简单修改用于其他产品的控制，使用范围特别广泛，具有良好的市场前景。

参 考 文 献：

[1] 苏鹏，周风余，陈磊. 基于 ST32 的嵌入式语音识别模块设计 [J]. 单片机与嵌入式系统应用，2011，11(2)：42 - 45.

[2] 周艳萍. 机器人嵌入式语音识别系统设计与开发 [D]. 广州：华南理工大学，2012：34 - 38.

[3] 高维深. 基于 HMM/ANN 混合模型的非特定人语音识别研究 [D]. 成都：电子科技大学，2013：24 - 29.

[4] 刘任平，侯瑞真，方英兰，等. 拟人机器人语音识别系统的硬件设计 [J]. 计算机时代，2013(1)：1 - 3.

[5] 蔺鹏. 基于语音识别技术的家居环境控制系统设计 [J]. 兰州工业学院学报，2013，20(3)：7 - 9.

[6] 禹琳琳. 语音识别技术及应用综述 [J]. 现代电子技术，2013，36(13)：1 - 3.

[7] 杨海燕，景新幸，曾招华. 基于 DSP 开发板的语音识别系统的研究 [J]. 计算机测量与控制，2013，21(1)：12 - 14.

责任编辑：张荣香

(上接第 57 页)

参 考 文 献：

[1] 张园园，刘琪. 军用型号项目软件复用成本度量模型 [J]. 计算机工程与应用，2015(5)：234 - 239.

[2] 刘艺. Delphi 面向对象编程思想 [M]. 北京：机械工艺出版社，2003.

[3] 何炎祥，杨健康. 基于群体智慧的软件开发间层模型及其架构实现 [J]. 计算机科学，2015，42(1)：175 - 179.

[4] 杜欣，汪春燕，倪友聪. 基于规则的软件体系结构层性能优化模型 [J]. 计算机科学，2015，42(10)：189 - 192.

[5] 唐克，王猛. Web 应用软件系统的性能分析与优化 [J]. 电脑开发与应用，2014，27(7)：42 - 44.

[6] 王海强. 软件复用与软件构件技术 [J]. 硅谷，2015，8(3)：49 - 51.

[7] 张晓宇. 提高代码可重用性的研究 [J]. 微型电脑应用，2016，32(1)：48 - 50.

[8] 钟林辉，朱小征，宗洪雁，等. 基于本体及模式驱动的构件化软件共同变化识别研究 [J]. 计算机应用研究，2016，33(3)：773 - 778.

责任编辑：薛慧心