计算机高级编程技术

第三章　面向对象的程序设计

第3章　面向对象的程序设计

- 对象（也被称为类）是包括数据和代码的实体，Delphi的对象通过全面支持继承、封装和多态性，提供了面向对象编程的强大功能。
  - 3.1 面向对象的基本概念
  - 3.2 类与对象
  - 3.3 类的方法
  - 3.4 类的封装与继承
  - 3.5 异常处理

3.1 面向对象的基本概念

- 面向对象是一种方法，一种思想，同时又是一种技术。
- 面向对象技术以基本对象模型为单位，能层次清晰地表示企业全局对象模型。
- 面向对象方法则从根本上对问题域中的对象及其关系进行详尽的分析，并在此基础上完成需求功能。

3.1 面向对象的基本概念

- 对象有以下几个共同特点：
  - 某类对象是对现实世界具有共同特性的某类事物的抽象。
  - 对象蕴含着许多信息，可以用一组属性来表征。
  - 对象内部含有数据和对数据的操作。
  - 对象之间是相互关联和相互作用的。

3.1 面向对象的基本概念

- 面向对象的特点主要概括为抽象性、继承性、封装性和多态性。
  - 多态性：是指不同类型的对象可以对相同的激励做出适当的不同相应的能力。
  - 抽象性：指对现实世界中某一类实体或事件进行抽象，从中提取共同信息，找出共同规律，反过来又把它们集中在一个集合中，定义为所设计目标系统中的对象。

3.1 面向对象的基本概念

- 继承性：新的对象类由继承原有对象类的某些特性或全部特性而产生出来，继承性简化了对新的对象类的设计。

- 封装性：是指对象的使用者通过预先定义的接口关联到某一对象的服务和数据时，无需知道这些服务是如何实现的。

3.2 类与对象

- 类与对象是Object Pascal程序设计中的主要概念，类是用户定义的数据类型。而对象是类的实例，它是由类定义的数据类型的变量。

- 可以把具有相似特征的事物归为一类。也就是把具有相同属性的对象看成一个类(class)。在面向对象的程序分析和设计技术中，“类”就是对具有相同属性和相同操作的一组相似对象的定义。从另一个角度来看，对象就是类的一个实例。

3.2.1 类的定义

- 类的定义形式如下：

```
type
  className=class(ancestorClass)
    memberList
end;
```

3.2.1 类的定义

上面的className为类的名称。

ancestorClass为父类的名称。

memberList为成员列表。

在Delphi中，如果不指明父类，则默认的父类为TObject类，也就是直接从TObject类派生出一个新类。TObject类是在System单元中定义的。

3.2.2 构造函数和析构函数

- 作为类类型的实例的对象（Object）是一个动态分配的内存区，它具有和类类型相同的数据结构。
- 构造函数和析构函数是一个类对象的特殊方法，它们控制了对象的建立和删除。
- 构造函数和析构函数是类定义中两个非常重要的函数，它们完成的功能正好相反，它们的定义也比较特殊。

3.2.2 构造函数和析构函数

- 1. 构造函数
  - 对象的创建和初始化工作是由类的构造函数来完成的。
  - 在定义构造函数的时候，使用保留字constructor。
  - 构造函数必须使用默认的函数调用约定方式，也就是使用register指令字方式。
  - 如果在创建并初始化对象时，调用构造函数发生错误，则会自动调用析构函数来删除这个没有完成的对象。

3.2.2 构造函数和析构函数

- 2. 析构函数
  - 析构函数的作用是将对象删除并释放相应的内存资源，此外还可以在这之前保存一些数据信息并关闭文件或数据库等，或者对一些设备进行复位并关机。
  - 在定义析构函数的时候，使用保留字destructor。如果在定义类的时候没有定义析构函数，则系统会自动为该类生成一个默认的析构函数。

3.3 类的方法

- 在调用构造函数的时候，用户使用的是类，而不是具体的对象。类的方法是在类中定义的且包装在类中的子程序。

- 方法的实现与一般的过程和函数的声明类似，只不过需要在实现的首部加上类的名称。如下例是关于方法的实现方法。

3.3 类的方法

例3-1：类方法的使用。

```
program Project1;
{$APPTYPE CONSOLE}
type
TStudent=class                          // 学生类
Name:string;                  // 学生姓名
class procedure AddOne;       // 学生数增加一个
destructor Destroy;override;  // 学生数减少一个
end;
var
```

3.3 类的方法

```
StudentNum:Integer;          // 表示当前的学生数
S1,S2:TStudent;              // 声明学生类的变量
{TStudent}
class procedure TStudent.AddOne;
begin
StudentNum:=StudentNum+1;
end;
destructor TStudent.Destroy;
begin
StudentNum:=StudentNum-1;
```

3.3 类的方法

```
inherited Destroy;
end;
begin
S1:=TStudent.Create;
S1.AddOne;// 调用类方法改变变量StudentNum
S1.Name:='王晓慧';
Writeln('学生数为：',StudentNum);
S2:=TStudent.Create;
S2.AddOne;
S2.Name:='周星星';
```

3.3 类的方法

```
Writeln('学生数为：',StudentNum);
S1.Free;
S2.Free;
Writeln('学生数为：',StudentNum);
Readln;
end.
```

运行结果如下：

学生数为：1

学生数为：2

学生数为：0

3.4 类的封装与继承

- 在对象Pascal语言中，类是一个构造类型，由属性和方法构成。其中属性又包括类的内部属性和外部属性；方法则是该类或其实例可以操作的过程和函数。
- 一个类中包含了一系列数据成员和方法，在一个好的面向对象的程序设计当中，数据应该被封装，仅在类中使用。类的封装就是把数据和代码组合在同一结构中，这样就可以对类中的数据起到保护作用。

3.4 类的封装与继承

- 每个保留字的具体含义如下
- 1. private
  - 具有private属性的成员称为私有成员，不能被类所在单元以外的程序访问。但是，如果在同一个单元文件中定义了两个类，那么，在一个类的成员中就可以对另一个类中的私有变量进行访问，或者调用另一个类中的私有方法。

3.4 类的封装与继承

- 2. protected
  - 具有protected属性的成员称为保护成员，可以被该类的所有派生类访问，并且成为派生类的私有成员。
- 3. public
  - 具有public属性的成员称为公有成员，可以被该类以外的类访问。如果两个类不在同一个单元文件中，则要在uses语句中包括被访问的类所在的单元名称。

3.4 类的封装与继承

- 4. Published
  - 具有Published属性的成员为发行类型成员，它的访问权限与公有成员相同，只是在设计期间亦可被访问。通常发行类型成员用在组件类的声明中，这样，就可以在对象编辑器中访问组件的发行类型的成员。

3.4 类的封装与继承

- 5. automated
  - 具有automated属性的成员称为自动类型成员，它的访问权限基本同公有成员。
- 除了在类封装的时候可以限制成员的访问权限外，单元文件中也可以限制对变量、对象、函数和过程等的访问权限。
- 类的继承是面向对象的程序设计允许用户定义的从一个已经存在的类定义一个新类的技术

3.5 异常处理

- 当一个错误或其他一些事件中止了程序的正常运行，系统就会抛出一个异常。通过Delphi的异常处理机制，可以捕获这个异常并进行处理。
- 可以利用类的继承性将一组异常组合成一个系列。
  - 3.5.1 raise语句
  - 3.5.2 try…except语句
  - 3.5.3 try…finally语句

3.5.1 raise语句

- 使用raise语句调用一个异常类的构造函数，并抛出一个异常。
- 通常，raise语句的形式如下：

  raise object at address

  - 其中object和at address是可选项
  - address通常是一个指向过程或函数的指针。一个抛出的异常在处理过后自动地被删除，一般不去主动地删除一个异常对象。

3.5.2 try…except语句

- 在try…except语句中可以进行抛出异常和处理异常的工作。
- try…except语句的一般形式如下：

```
try
  Statements1;
except
  on Exception1 do HandleStatements1;
  on Exception2 do HandleStatements2;
  …
```

3.5.2 try…except语句

```
on ExceptionN do HandleStatementsN;
else
  Statements2;
end
```

3.5.2 try…except语句

- 其中Statements1为程序中的普通代码，可以为复合语句。在Statements1中可能会抛出异常：通过raise语句可以抛出异常，运行时错误也可以抛出异常。
- 当异常产生后，程序就转到except部分。
- 如果异常列表中没有当前产生的异常对象，则执行else部分。

3.5.2 try…except语句

- 在例程中，假设整型变量ResultNum的范围在0~100之间。如果小于0，则抛出一个EResultToSmall错误；如果大于100，则抛出一个EResultTooBig错误。

```
program Project1;
{$APPTYPE CONSOLE}
uses SysUtils;
type
EResultTooBig=class(Exception) // 异常类1
 Num:Integer;
```

3.5.2 try…except语句

```
constructor Create(N:Integer);
// 异常类1的构造函数
  end;
  EResultTooSmall=class(Exception)  // 异常类2
    Num:Integer;
  end;
  constructor EResultTooBig.Create(N:Integer);
begin
 Num:=N;
            end;
```

3.5.2 try…except语句

```
var
        ResultNum:Integer;
        begin
        ResultNum:=-2;
try                                    // 尝试运行
   if ResultNum>=100 then      // 如果数据太大，抛出异常
     raise EResultTooBig.Create(ResultNum)
   else if ResultNum<=-1 then  // 如果数据太小，抛出异常
     raise EResultTooSmall.CreateFmt('数据不可以小于0。',[ ]);
```

3.5.2 try…except语句

```
except
    on EResultTooBig do
      Writeln('错误：返回值不可以大于100。');
    on EResultTooSmall do
      Writeln('错误：返回值不可以小于0。');
  else
    Writeln('错误：其它运行时错误。'); // 其他异常
  end;
  Readln;
end.
```

3.5.3 try…finally语句

- 对于有些操作，在异常处理部分要进行，在正常情况下也要进行。例如，在正常情况下，使用完文件之后关闭文件；如果在对文件操作的过程中出现了异常，也需要关闭已经打开的文件。这时，就可以把关闭文件的过程放在try…finally语句的finally部分，不管try部分的操作是否正常，都要进行finally部分的操作。

3.5.3 try…finally语句

- 通常try…finally语句的形式如下：

```
try
  statementList1
finally
  statementList2
end
```

3.5.3 try…finally语句

- 可以看到，try...finally语句的用法与try...except语句的用法很相似。statementList1可以为简单语句，也可以为复合语句。如果在statementList1中抛出了异常，程序立即转到finally部分；如果在statementList1中执行了Exit、Break或Continue过程而导致程序的控制离开statementList1部分时，程序也会跳转到finally部分；如果在try部分正常执行完毕，接着执行的还是finally部分。