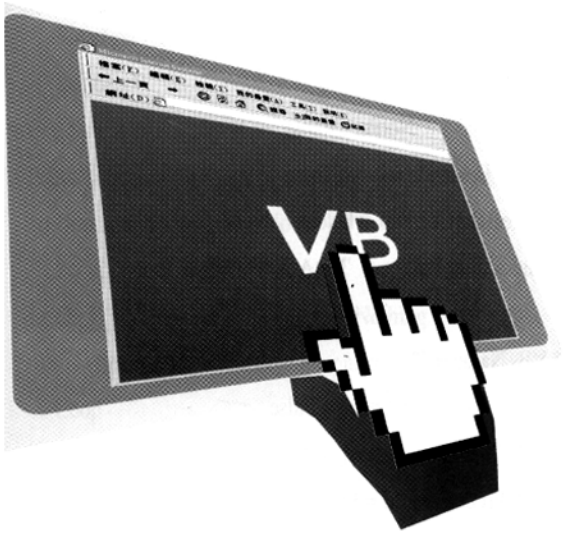

VB 中指针功能变通实现及应用

裴钟哲　刘小明 （北京工业大学交通工程研究所）
申礼宏 （南京军区司令部）
丁健 （南京解放军理工大学）
张万军 （总装工程兵驻蚌埠地区军代室）

Alternative Realization and Application of Points at VB

1 引言

在某些应用情况下，链式存储结构是存储数据方法的唯一选择，例如在对于包含10000个顶点的网络（图）的数据存储中，用"邻接矩阵"法仅邻接矩阵就至少需要100M的存储空间，这显然是不现实的，所以必须采用基于链式存储结构的"邻接表"存储方法。但在VB中，由于不存在指针类型变量，使得开发人员无法在VB环境下实现链式存储结构，极大地限制了VB在软件工程中的应用。因此，探索在VB中变通实现指针及扩展其在数据结构建立中的应用具有显著的实用意义。

VB从5.0版本开始支持定义对象类型变量，并提供了"Set"语句来将对象引用赋给对象类型变量。通过使用"Set"语句，可创建预定义类或对象的新实例、对新实例进行引用、必要时断绝变量与对象实例间的关联，从而完成类似C和C++中指针类型变量对变量内存地址引用的功能，使得在VB中实现链式存储结构成为可能。我们在开发语言为VB的工程中，运用本文所述方法，在交通网络（图）的存储表示中，实现了基于链式存储结构的"邻接表"存储方式。工程运行实践表明本文所述方法切实可行、简单高效。

2 VB中对象变量及Set语句

2.1 对象变量

对象变量可定义为已知类或Object类型。Object类型变量存储为32位（4个字节）的地址形式，类型为Object的对象变量。可以引用任何类型对象实例。

对象变量的声明方法和声明其它变量一样，要用Dim、ReDim、Static、Private和Public。仅有的不同在于可选的New关键字和Class参数。语法是：

{Dim | ReDim | Static | Private | Public} variable As [New] class

对象变量与其它变量的区别在于，除了存储值以外，对象变量还可引用对象。

VB中用Set语句将对象赋予对象变量。

2.2 Set语句语法

Set语句的功能是将对象引用赋给变量或属性，其语法为：

Set objectvar = {[New] objectexpression | Nothing}

Set语句的语法包含下面部分：

Set语句将对象赋予对象变量的形式为：

Set variable = object

3 VB中的指针功能的变通实现方法

我们通过VB的对象变量，可以类似实现C和C++中指针变量的功能。首先声明变量：

表1

部分	描述
Objectvar	必需的。变量或属性的名称，遵循标准变量命名约定。
New	可选的。通常在声明时使用New，以便可以隐式创建对象。如果New与Set一起使用，则将创建该类的一个新实例。如果objectvar包含了一个对象引用，则在赋新值时释放该引用。不能使用New关键字来创建任何内部数据类型的新实例，也不能创建从属对象。
Objectexpression	必需的。由对象名，所声明的相同对象类型的其它变量，或者返回相同对象类型的函数或方法所组成的表达式。
Nothing	可选的。断绝objectvar与任何指定对象的关联。若没有其它变量指向objectvar原来所引用的对象，将其赋为Nothing会释放该对象所关联的所有系统及内存资源。

Dim variable As class

然后把对象赋予变量：

Set variable = object ' object是已存在类

上述语句执行结果是variable对象变量引用了object对象实例，对象变量内存储了对象实例的地址。可以设置多个对象变量引用同一个对象实例。

另外，Set语句的New（申请内存，创建类的新实例）和Nothing（释放对象所关联的所有系统及内存资源）关键字使得开辟申请内存、释放内存成为可能。

因此，我们完全可以把对象变量当作指针变量来使用。所不同的是，在C和C++中指针变量指向用"Type"定义的节点结构，而在VB中对象变量指向定义为类的节点结构，这种类内部还包含面向特定数据结构（如链表）的各类实现方法。

4 VB中指针应用实例——链式存储结构创建

4.1 链式存储结构及其在VB中的创建步骤

链式存储结构中节点由数据域V(i)和指针域Next(i)组成，其中V(i)是地址为I的节点中所存放的数据元素值，可按工程实际需要设置多个元素，Next(i)为V(i)的后件的存储地址。

VB中链式存储结构建立步骤是：设计节点类、定义节点类对象、调用类的方法创建链表。

4.2 建立图拓扑数据的"邻接表"存储结构

采用邻接表类型存储图的拓扑数据结构，在业界早已达成共识。邻接表的基础是链式动态存储。

邻接表是表示图内顶点之间相邻关系的N（N是图中顶点个数）个单向链表，N个单向链表形成顶点表。一个实际的网络图及其邻接表的示意图如下：

在VB中建立图的邻接表关键代码为：

(1) 节点类"My_Node"定义

节点类"My_Node"的定义放在"ajd_table_node.cls"文件内，此文件在新建工程时以类模块形式加入，代码如图1：

```
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1          'True
    Persistable = 0        'NotPersistable
    DataBindingBehavior = 0  'vbNone
    DataSourceBehavior  = 0  'vbNone
    MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = ""
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
    '上面的是VB中添加类模块时共有的设置
Public Vertex_no As Integer, weight_data As Single, pnext As My_Node
    '定义3个公有成员，含义分别为：图内某顶点关于边的邻接顶点，边的长度等权值，后件地址
Private length As Integer, pfirst As My_Node
    '定义2个私有成员，含义分别为：邻接表长度，邻接表表头
Property Get first_node() As My_Node
    '取得链表表头
    Set first_node = pfirst
End Property
Property Get length_of_my_node() As My_Node
    '取得链表表长度
    length_of_my_node = length
End Property
Public Sub makeempty()
    '建立空链表表
    Dim new_node As My_Node
    Set new_node = New My_Node      ' 定义一个新节点
    new_node.Vertex_no = 0          ' 邻接顶点为0
    new_node.weight_data = 0        ' 权值（边的长度）为0
    Set new_node.pnext = Nothing    ' 后件地址为无
```

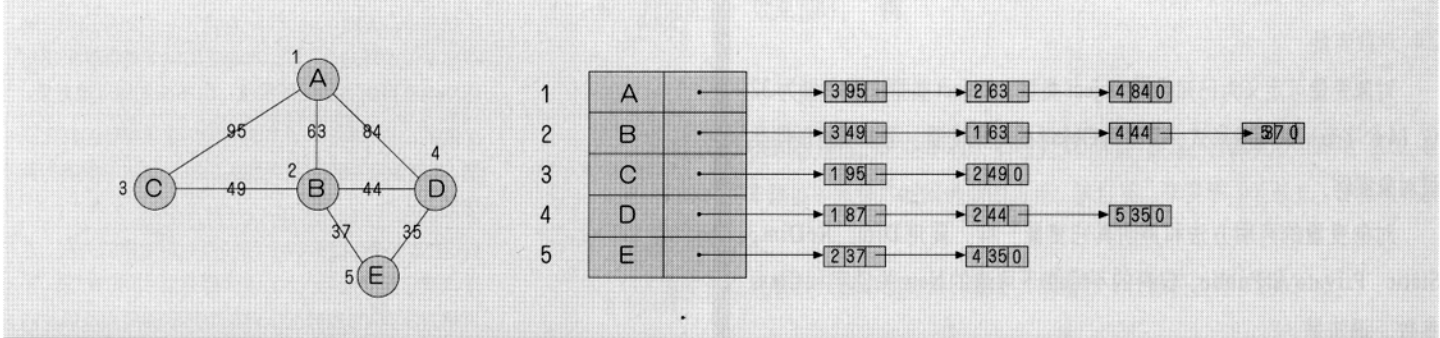

图1

```
    Set pfirst = new_node                ' 表头指针指向无
    length = 0                           ' 表长度为 0
End Sub
Public Sub addnode(ByVal V_no As Integer, w_data As Single) '对表加顶点
    Dim add_node As My_Node
    Set add_node = pfirst
    Do While Not add_node.pnext Is Nothing
        Set add_node = add_node.pnext ' 移动指针至链表尾节点
    Loop
    Set add_node.pnext = New My_Node
    add_node.Vertex_no = V_no
    add_node.weight_data = w_data
    Set add_node = add_node.pnext ' 在链表尾加入新节点
    Set add_node.pnext = Nothing ' 新节点后件地址设置为无，标示链表结束
    length = length + 1                  ' 链表长度在原来基础上加 1
End Sub
Public Function searchnode(ByVal V_no As Integer) As My_Node '对表寻找顶点
    Dim search_node As My_Node
    Set search_node = pfirst
    Do While search_node.Vertex_no <> V_no And Not search_node.pnext Is Nothing
        Set search_node = search_node.pnext ' 按编号检索节点
    Loop
    If Not search_node.pnext Is Nothing Then ' 检索到指定编号节点，返回节点
        Set searchnode = search_node
    Else
        Set searchnode = Nothing ' 如检索至尾节点，仍没找到匹配，则返回空
    End If
End Function
```

(2) 顶点表类型及顶点表数组定义

```
' 全局定义
Type vertex_adj_table_node    ' 用户自定义数据类型，顶点表类型
    Vertex_no_by_turn As Integer
    Next_node As My_Node
End Type
Global adj_table() As vertex_adj_table_node    ' 顶点表数组定义
```

(3) 邻接表建立

```
' 放在事件过程内
    real_vertx_num = 5    ' 针对示意图中的图顶点数为 5
    ReDim adj_table(real_vertx_num)    '按图的实际顶点数重新定义顶点表数组
    For i = 1 To real_vertx_num
        adj_table(i).Vertex_no_by_turn = i    ' 每个顶点的编号
        Set adj_table(i).Next_node = New My_Node ' 每个顶点建立一个链表
        adj_table(i).Next_node.makeempty
        For j = 1 To edge_number
            '用网络拓扑关系即 L_V()数组内容对链表填充，L_V()数组定义见下文
            If L_V(j).qd_n = i Then
                adj_table(i).Next_node.addnode L_V(j).zd_n, L_V(j).xdcd
            End If
            If L_V(j).zd_n = i Then    ' 考虑无向图情况
                adj_table(i).Next_node.addnode L_V(j).qd_n, L_V(j).xdcd
            End If
        Next
    Next
    其中 L-V()在先前被定义为 Line-to-vertex-type 类型变量
    Type Line-to-vertex-type    '边与顶点关系类型
        Line-n As Integer          '边的序号
        qd-n As Integer            '边的起点序号
        zd-n As Integer            '边的终点序号
        xdcd As Single             '边的长度
    End Type
    L-V() As Line-to-vertex-type
```

L-V()数组内容即网络拓扑关系的填充本文从略。

4.3 链表的使用

链表的使用主要是链表的检索，本文以前面建立的邻接表数据对话框显示输出为例，给出代码：

```
    ' 放在事件过程内
    Dim pppp As My-Node
    For i = 1 To real-vertx-num
        Set pppp = adj-table(i).Next-node.first-node
        Do While Not pppp.pnext Is Nothing
            ' 对话框显示链表各节点的内容
            MsgBox "邻接顶点序号：" + Str(pppp.Vertex-no) + "；距离：" + Str(pppp.weight-data)
            Set pppp = pppp.pnext
        Loop
    Next
```