优化 Visual Basic.NET 应用程序的性能

于 华　都一兵　(山东财政学院计算机信息工程系 250014)

摘要:Visual Basic .NET 的一个主要目标是比以前版本执行得更快。但是性能依赖于应用程序如何设计。程序的优化大体上可分为两种:速度优化和大小优化。本文从这两方面介绍了对 VB.NET 应用程序的优化方法。

关键词:Visual Basic.net　性能　优化

1　速度的优化

在用户评价应用程序是否满意的诸多因素中,速度是决定性的因素。所以优化应用程序,首先应加快其速度。

程序的速度可分为三种:真实速度(执行代码的实际时间)、显示速度(屏幕显示的时间)和感觉速度(程序运行时的感觉速度)。

1.1　优化真实速度

为了加快应用程序的真实速度,可从以下角度进行考虑。

(1) 避免使用 Object 类型。作为 Object 声明的引用类型变量可以指向任何类型的数据。但是这种灵活性也会降低性能,因为 Object 变量通常是迟绑定的。如果引用类型变量声明为特定类(例如 Form),它就是早绑定的,这样就允许 Visual Basic 编译器在编译时就作一定的优化,例如类型检查和成员查看表。当在运行时访问早绑定对象变量的成员时,编译器已经完成了大量的管理工作了。如果变量声明为 Object 类型或者没有明确的数据类型它就是迟绑定。当代码访问类似变量的成员时,通用语言运行时被迫在运行时执行类型检查和成员查看表。早绑定比迟绑定的对象的性能好得多,同时也更容易阅读和维护,并减少了运行时错误的数量。因此在设计时就应该使用具体类的类型声明对象变量。在不必要的时候要尽量避免 Object 类型的使用。注意,如果 Option Explicit 设置为 Off,没有明确数据类型的变量声明就是 Object 类型。

(2) 选用最佳的数据类型。最高效的数据类型使用运行时平台的数据宽度。在当前的平台上,对于计算机和操作系统来说数据宽度是 32 位的。因此 Visual Basic .NET 中目前效率最高的数据类型是整数(Integer),接着是长整型(Long)、短整型(Short)和字节型(Byte)。如果需要小数值,最好的选择是双精度型(Double),因为当前平台的浮点处理器使用双精度执行所有操作。按效率排序接着的是单精度(Single)和十进制(Decimal)。

(3) 尽量减少装箱(Boxing)和取出(Unboxing)。当把值类型处理为引用类型时,通用语言运行时必须处理装箱(Boxing)。例如声明一个整型变量并且把它指定给一个 Object 变量或者把它作为 Object 参数传递给某个过程。在这种情况下,通用语言运行时必须把变量装箱并转换成 Object 类型,它复制该变量,把副本嵌入新的分配的对象中,并且存储它的类型信息。如果随后把包装的变量指定给值类型的变量,通用语言运行时必须取出该变量,把数据从堆实例中复制到值类型变量。而且包装的变量必须在堆中管理,无论它是否被取出过。

包装和取出明显降低了性能。如果应用程序频繁地把值类型变量作为对象处理,最好最初使用引用类型声明。另一种选择是只包装变量一次,在使用过程中一直保留它,再次需要该变量时取出它。

(4) 将常用的属性值、函数返回值缓存在变量中。变量的执行速度比同类型的属性的处理速度快许多。因为变量的访问是简单地从内存中存取,而属性的访问需要调用该属性上的 Get 或 Set 方法,而它们除了存取值外还要作其他的操作。因此如果经常用到某一属性的值(如在循环体中),应该先将该属性值赋给某一变量,以后用该变量代替该属性,这样就能够提高代码的速度。同样的技术可用于处理函数的返回值,用变量缓存函数的返回值,也会大幅度的提高速度。

(5) 正确地选择集合和数组。当有一系列处理方法相似的相关联的对象时,可以把它们放入对象数组中,或者把这些对象作为某个集合的成员。从速度的观点看,选择何种方式取决于对象的访问方式。下面几点可以帮助你在两者之间作出选择:

- 通用语言运行时能为数组优化代码,但是如果每个都访问集合的话就需要一个或者更多的调用。因此,当数组支持所有必要的操作时,通常更好。
- 对于索引的访问,数组从来不慢,并且通常比集合

快。

· 对于键控制的访问，应该使用集合。数组不支持使用键字段的访问，因此，必须编写代码搜索整个数组元素查找键。

· 对于插入和删除，集合通常更好。数组没有直接支持添加和删除元素。如果你在数组的末尾插入或删除，必须使用 ReDim 语句，而它会降低性能。为了随处插入或删除，必须使用 ArrayList 对象代替标准的数组。与此对比，在集合中插入和删除都是直接的操作，并且不管元素涉及的位置如何它们的速度都相同。

(6) 内嵌过程与过程调用。采用过程调用使代码更具模块化的风格，更容易阅读，同时也使应用程序的其他位置可以调用该过程。但过程调用总是增加额外的操作和处理时间。如果循环体中多次调用某一过程，就可以直接把该过程写到循环体中去，以消除过程调用时的额外负担。

但另一方面，如果应用程序的其他部分需要访问该代码，重复的代码一方面增加了应用程序的大小，另一方面将使维护更加困难，容易出现更新同步错误的问题。因此，在使用内嵌过程时要仔细权衡利弊。

(7) 尽量用 ByVal 传递过程参数，而不用 ByRef。当使用 ByRef 关键字给过程传递一个参数时，无论该变量是值类型还是引用类型，Visual Basic 只复制下层变量的指针。当使用 ByVal 传递参数时，将复制下层变量的内容。对于引用类型，内容只包括对象本身的指针；对于值类型，包含所有的变量数据。当你传递适当大小的值类型(Integer 或 Double)时，因为编译器优化了调用代码(例如在寄存器中保持了参数)，所以 ByVal 效率更高。在没有强制原因要使用 ByRef 时，应该使用 ByVal 传递参数。

(8) 尽量少用点操作。当从 Visual Basic 中引用其他应用程序的对象时，可使用点语法"."对对象的集合、属性和方法等进行应用。然而每一个"点"都需要 Visual Basic 产生多次调用，因此为了写出最有效的应用程序，引用对象时应尽可能少使用多余的点操作。

· 使用缺省方法和属性。当对象使用缺省方法和属性时，可以省略该方法和属性，也就减少了一次"点"的使用。

· 使用 With...End With。使用 Visual Basic 提供的 With...End With 语句，不仅可以大大减少"点"的使用，同时还使代码简洁易读。

(9) 使用线程 。如果应用程序要用很长时间等待它的某个操作完成，应考虑使用异步处理，使用 System.Threading 命名空间中 Thread 类的方法。异步处理和多线程需要额外的代码，但是它们能显著地提高性能。但是要注意，多线程也有开销并且使用必须小心。使用期限短的线程不仅效率低下，而且上下文(context)转换明显花费了很多执行时间。因此应该尽量减少线程的数量，并尽量少进行切换。

1.2 优化显示速度

在 Windows Form 应用程序中，加快图形和其他控件的显示速度对整个应用程序的速度的提高起非常重要的作用。可以考虑下面的方法：

(1) 避免不必要的控件重画。重画的代价是昂贵的，一种有用的途径是在设置控件的属性时隐藏它。

例如，假设在窗体的 Resize 事件中调整整个列表框的大小：

```
Sub Form_Resize()
  Dim I as Integer,sHeight as Integer
  SHeight=Scaleheight/4
  For I=1 to 3
    lstDisplay(i).Move 0,I * sHeight,ScaleWidth,sHeight
  Next
End Sub
```

上面的代码产生四次独立的重画，每个列表框一次。可以把所有的列表框放在一个图片框中，并在移动或调整列表框大小之前隐藏图片框，就会减少重画的次数。当再次使图片框可见时，所有的列表框一次画出，修改后的代码如下：

```
Sub Form_Resize()
  Dim I as Integer,sHeight as Integer
  PicContainer.Visible=False
  PicContainer.Move 0,0,ScaleWidth,ScaleHeight
  SHeight=Scaleheight/4
  For I=1 to 3
    lstDisplay(i).Move 0,I * sHeight,ScaleWidth,sHeight
  Next
  PicContainer.Visible=True
End Sub
```

(2) 当需要重画控件或者任何显示对象时，尽量重画对象的新的暴露区域。这会减少用户的等待时间。

(3) 在 Visual Basic .NET 中没有与早期版本中的 Image 相等的控件，必须使用 PictureBox 控件显示大多数图形，而且也不支持 AutoRedraw 功能。

1.3 优化感觉速度

通常，应用程序的感觉速度和代码的实际执行速度并无多大关系。对用户来说，启动快、绘画快并提供不间断的反

馈信息的应用程序显得速度快;而在完成任务时似乎"悬挂"起来的应用程序显得速度慢。下面的技术可以提高应用程序可感觉到的显示速度。

(1) 用进展指示器和等待光标。如果程序中存在不可避免的长时间延迟,则必须给用户以提示,以帮助用户确信应用程序仍然在运行。可使用 System.Windows.Form 名字空间中 ProgressBar 类的 ProgressBar 控件,或者设置窗体的 MousePointer 属性把鼠标指针变为沙漏形。

(2) 隐藏窗体。在预先载入窗体或者控件时,把它们隐藏。这样减少了描绘的必要。

(3) 预加载数据。在应用程序需要重要数据前预先载入数据,这包括窗体和控件以及其他的数据项。尽管载入这些数据需要相同的时间,但是减少了用户需要看到它们时的等待时间。

(4) 使用线程。当应用程序在等待用户输入时,在后台使用线程和计时器作一些小的事务。这样在用户请求数据时可以帮助准备数据。

(5) 加快应用程序的启动速度。用户对应用程序印象的好坏往往取决于程序的启动速度,因此应尽可能地加快程序的启动速度。下面的方法可提高启动速度:

- 简化启动窗体以减少载入和初始化时间。
- 在启动窗体的 load 事件的第一行调用 me.show。
- 避免立即载入不必要的模块。

2 大小优化

缩小应用程序的大小不仅可以节约系统资源(减少内外存的占用),同时较小的执行文件运行更快,因此应对应用程序的大小进行优化。

(1) 减少同时载入的窗体的数量。每一个加载的窗体,无论可视与否,都要占用一定数量的内存。只在需要窗体时才载入,除非希望优化可以感觉到的显示速度。当使用完窗体时,卸载它并把变量设置为 Nothing,以释放所有空间。

(2) 减少控件数目。窗体应尽量少用控件,而且应尽量使用最简单的控件。控件越简单占用的 Windows 资源越少,例如使用标签代替文本框。

(3) 组织模块。尽量把相关的过程放在同一模块中。这样减少了载入的模块数量。

(4) 使用简单的数据类型。简单数据类型不仅运行速度快,其占用的内存资源也少。

(5) 变量用完后,把它设置为 Nothing,以释放其占用的资源。

(6) 如果已经使用完了数组中的一些元素,但是需要保持另外的元素,应使用 ReDim 释放不必要的内存。

(7) 对象不再被使用时,及时用 Dispose 方法清除以释放它所占用的内存空间。微软.NET 通用语言运行库(CLR)提供了垃圾回收功能(Garbage collection),当对一个对象的所有引用都结束时,Garbage collection 会自动将其销毁,同时释放它所占用的内存空间。但是,开发人员无从知道垃圾回收器将于何时运行并销毁所创建的对象,因此,应用程序应跟踪对象并判断哪些对象不再被使用,通过发送对象不再被使用的信号,来提高这一过程。这就需要调用一个对象的 Dispose 方法。

.Net 框架中很多类都包括 Dispose 方法,例如:SqlConnection ,SqlCommand ,qlDataReader ,Timer ,EventLog,Font 等。例如下面的 VB.NET 源代码范例演示了当处理一个 SQL Server 连接时的这些对象的使用过程。

```
Dim conn As SqlConnection
Dim comm As SqlCommand
Dim dr As SqlDataReader
conn = New SqlConnection()
comm = New SqlCommand()
conn.ConnectionString = "data source = test;
user id = test;password = test"
comm.CommandText = "SELECT * FROM test
comm.CommandType = CommandType.Text
comm.Connection = conn
conn.Open()
'Do something with the database
conn.Close()
conn.Dispose()
comm..Dispose()
dr.Close()
```

以上 ,笔者介绍了 Visual Basic.Net 应用程序的优化技巧,期望能对广大编程爱好者有所启迪。