TADODataSet 的 Next 方法性能探讨

Discussion about the performance of TADODataSet's Next mothed

黄 河 （温州职业技术学院 计算机系 浙江温州 325035）

摘要:本文探讨了 Delphi 的 ADOExpress 组件中存在一个性能问题:TADODataSet 组件的 Next 方法的时间性能会随着游标的向后移动而变差。通过分析 TADODataSet 组件的源代码,找到的导致这个性能问题的原因。并且,提出的两种解决此问题的方法。

关键词:ADO VCL TADODataSet 性能

1 ADOExpress 组件

如何存取数据一直是软件技术开发的重心之一,因为大部分的应用程序都需要存取各种不同的数据并根据这些数据进行运算。随着数据类型不断地复杂和多样化,应用程序程序员必须花费更多的时间和成本撰写存取数据的程序代码。再加上新操作系统平台的出现,应用系统的需求增加以及数据库系统持续进步,都使数据存取的工作更加困难,并超过了程序员能够负担的范围,因此使用一种统一的标准让程序员存取数据便成为一个非常重要的需求。这便是 ODBC、BDE/I DAPI 和 ADO 等标准数据存取技术出现的重要原因之一。

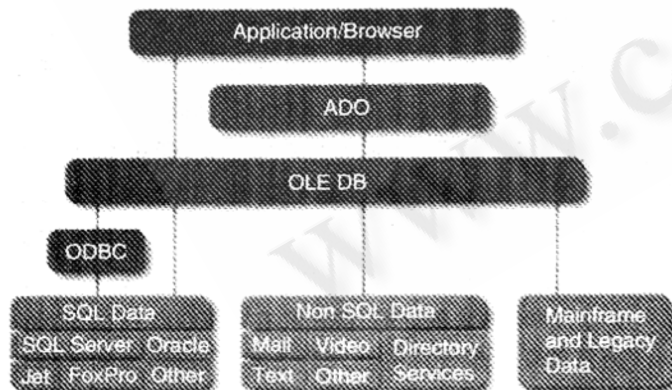

图 1

ADO 是 Microsoft 存取通用数据源的标准引擎。ADO 主要是让应用程序或 Web 应用程序存取各种不同的数据源。ADO 封装了 OLE－DB 复杂的接口,以极为简单的 COM 接口存取数据。图 1 是 ADO 的架构图。从图 1 我们可以看到,ADO 能够藉由 OLE－DB 存取传统的关系数据库,或 Flat－File 类型的数据库;也可以存取非传统的数据,例如文字、Email、声音、图形、影像等。更可以通过 OLE－DB,藉由 Connector 来存取大型的数据源,例如 CICS 等。但是不管应用程序要存取哪一种数据源,应用程序都只需要使用 ADO,而不需要使用各种不同的复杂 API 来存取不同的数据,这样就可以大大简化应用程序员的工作。

原生 ADO 对象的架构图如图 2,主要对象有这么几个:Connection 对象负责和数据源建立连接,因此 ADO 应用程序一开始应该使用 Connection 对象来连接数据源。Command 对象代表 ADO 应用程序向数据源下达的命令。Recordset 对象可以说是原生 ADO 对象中的灵魂对象,它提供了访问结果集的能力。

在 Delphi 中,为了和原来 VCL 的数据访问和数据感知组件相配合,Delphi 的 ADOExpress 组件封装了原生 ADO 对象。ADOExpress 几乎是以一对一的方式来封装原生 ADO 对象,例如 ADO 的 Connection 对象是以 TADOConnection 组件封装,ADO 的 Command 对象是以 TADOCommand 组件封装。但是 ADO 的 Recordset 对象在 ADOExpress 中却使用了数个不同的 VCL 组件来封装,它们分别是 TADODataSet、TADOQuery、TADOTable 和 TADOStoredProc 组件。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

图 3 是 ADOExpress 组件架构图。在图中我们可以看到 TADODataSet 继承自 TCustomADODataSet，而 TCustomADODataSet 又继承自 TDataSet，TDataSet 是 Delphi 中数据访问的核心组件，Delphi 中的数据感知组件都可以通过 TDataSet 工作。这样，通过 ADOExpress 就把 ADO 融入了 Delphi 的 VCL 体系。

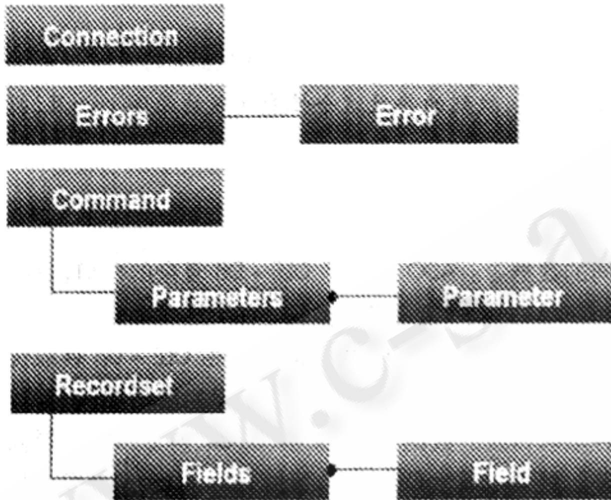

图 2

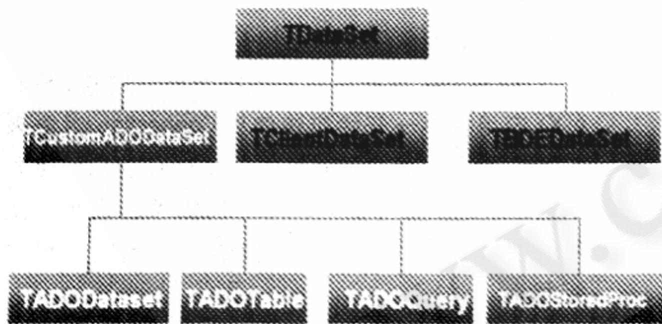

图 3

2 Next 方法的低效现象

笔者在利用 Delphi 开发一个应用程序时，碰到这么一个现象：使用 TADODataSet 打开一个结果集，然后调用其 Next 方法移动游标一行一行地遍历整个结果集，刚开始速度很快，越到结果集的后部越慢。如果结果集比较小，比如几千行的话，这种速度的变慢不易察觉，但是随着结果集的变大，这种速度的变缓会变得不可承受。为了说明这个现象，笔者作了一个实验。

实验是这样的：使用 TADODataSet 打开一个 10 万行左右的结果集，然后反复调用 Next 方法进行遍历（除了 Next 方法，没有调用任何其他方法），以 10000 次调用为单位进行计时。实验结果如表 1 所示。

表 1

计时次数	耗时(ms)	比上次增加耗时
1	1983	-
2	5948	3965
3	9954	4006
4	13500	3546
5	17255	3755
6	21080	3825
7	24876	3796
8	28711	3835
9	32537	3826
10	36332	3795

从结果中我们可以发现以下现象：

随着游标向后的移动，调用 Next 方法的耗时变得越来越大。从第 1 行到第 10000 行，遍历的时间，也就调用 10000 次 next 方法的耗时为 1.9 秒；而从第 90000 行到第 100000 行，遍历的时间膨胀到 36.3 秒。同样是 10000 次调用 next，两者时间差距达 30 多倍。而且这种增长几乎是线性的，每次移动 10000 行，都要比前一个 10000 行多耗时 4 秒左右。

可以预见，随着结果集的变大，调用 Next 方法（游标移动）的速度会越变越慢。而这种现象是要避免出现的，理想情况是，Next 调用速度不应该随着游标的向后移动而变慢，也就是说，在第 1 行上调用 Next 方法的耗时，应该和在第 10000 行调用 Next 方法的耗时是一样的。只有这样，应用程序的效率才能得到保证。

3 探寻低效的根源

为什么会出现这种现象呢？到底是 ADO 原生组件的问题，还是 Delphi 封装的问题呢？带着这些疑问，笔者对 TADODataSet 的源代码进行分析。分析的源代

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

码主要在两个单元:ADODB、DB。出于篇幅考虑,笔者仅列出特别需要说明的代码。

前面提到过,TADODataSet 继承自 TCustomADODataSet,而 TCustomADODataSet 又继承自 TDataSet。我们的分析也基本上在这三个组件中进行。

TADODataSet 没有覆盖 Next 方法,而 TCustomADODataSet 也没有覆盖 Next 方法。所以观察 TDataSet 的 Next 方法,其内容很简单,仅仅是调用了 MoveBy(1)。我们继续跟踪 MoveBy 方法。MoveBy 方法比较长,其中关键的调用是:GetNextRecord。而在 GetNextRecord 中关键的一句是 GetRecord(GetBuffer(FRecordCount),GetMode,True);其中 GetMode 是一个变量,已经在方法的开头设置为 gmNext;而 GetRecord 方法是一个虚方法,是 TDataSet 留给其子类去实现的。那么,问题关键转移 TCustomADODataSet 是如何实现 GetRecord 方法的。

在 TCustomADODataSet 的 GetRecord 方法中,除去一些维护 Filter 和 State 的一些代码,它把真正的移动动作交给了另一个方法 InternalGetRecord。移动结果集游标的最为核心,真正工作的代码正是隐藏在这里。

在 InternalGetRecord 方法的基本工作原理如下:首先它判断移动的方向,若是要求是向后移动,便调用 TCustomADODataSet 内含的原生 ADO 的 RecordSet 的 MoveNext 方法。若是向前,便调用 TCustomADODataSet 内含的原生 ADO 的 RecordSet 的 MovePrevious 方法。如果调用成功,便设置书签状态和行位置。

笔者分析,实际上真正的移动动作,在调用好 MoveNext 方法(或 MovePrevious)已经完成。而问题应该出现在其后的那些代码中。这部分的代码摘抄入下:

```
if Result = grOK then
  begin
    with PRecInfo(Buffer)^ do
   begin
    RecordStatus := Recordset.Status;
    if (BookmarkSize > 0) and ((adRecDeleted and RecordStatus) = 0) then
    begin
     BookmarkFlag := bfCurrent;
     Bookmark := Recordset.Bookmark;
     if ControlsDisabled then
      RecordNumber := -2 else
      RecordNumber := Recordset.AbsolutePosition;
    end else
     BookmarkFlag := bfNA;
   end;
   Finalize(PVariantList(Buffer + SizeOf(TRecInfo))^, Fields.Count);
   GetCalcFields(Buffer);
  end;
```

为了证明这种猜测,笔者将这段代码注释掉(也就是不做那些多余的动作),重新编译程序,程序运行的结果如下:

计时次数	耗时(ms)
1	40
2	40
3	30
4	40
5	40
6	30
7	30
8	40
9	40
10	40

从结果看,Next 方法的耗时大大缩减了,并且也不会随着游标的移动而变慢!果然,效率低下的原因正是出现这段代码中。

到底这段代码中的哪一行语句是罪魁祸首呢?笔者经过简单的实验,发现影响最大的是 RecordNumber := Recordset.AbsolutePosition;若是注释掉这一语句,next 方法的调用时间是基本固定的,不会随着游标的移动而变慢。正是这句导致了 Next 方法的低效。这一语句的要去取原生 RecordSet 方法的 AbsolutePosition 属性,而获取这个属性,在原生 ADO 对象中也是比较慢的会随着游标的向后移动变慢的操作。

问题的结论是这样的,Delphi 的 ADOExpress 的实现者,为了维护结果集当前游标的位置,在 InternalGetRecord 中不恰当的取用了原生 ADO 对象的 AbsolutePosition 属性导致 Next 方法性能会随着游标的移动而变差。

4 结论

问题的根源找到了,但是修改是却要考虑全面。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

如果我们简单的注释掉那一语句(RecordNumber := Recordset. AbsolutePosition;),会导致 TADODataSet 的组件中使用到 RecordNumber 数据成员的代码工作不正常,所以我们应该找到所有使用 RecordNumber 数据成员的地方,经过搜寻,发现只有在 GetRecNo 方法和 DoRecordsetDelete 方法中使用到 RecordNumber。解决的办法也很简单,把 GetRecNo 方法和 DoRecordSet-Daelete 方法中对 RecordNumber 的取用,改为直接取用 RecordSet. . AbsolutePosition。

对 TADODataSet 的源代码做了以上修改后,重新编译,再运行上面的实验,结果如下:

计时次数	耗时(ms)
1	50
2	60
3	50
4	50
5	50
6	60
7	60
8	60
9	60
10	50

这样修改后,虽然耗时比完全注释掉代码段 I 要增加一点,但是耗时不再随着游标的移动而变大。

还有另一种修改的方法:如果在代码中,仅仅是读取数据,那么向下移动游标时,可以选择不调用 TADO-DataSet. Next 方法;而是直接调用 TADODataSet 内含的原生 ADO Recordset 对象的 moveNext 方法。这样做的好处是不用修改 VCL 的源代码,风险要小的多。

参考文献

1 DELPHI 7.0 英文版,帮助文档,[CP/DK]。

2 (美)Steve Teixeira /Xavier Pacheco《Delphi5 开发人员指南》,机械工业出版社,2001。

3 (美) Lischner《Delphi 技术手册》[M],中国电力出版社,2001。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn