实现对存储过程中数据处理的动态监控

Implement Dynamic Monitor Of Data Handling in Store Procedure

邬 平 诸开梅 马继涛 吴 斌
(昆明云南省科学技术情报研究所 650051)

摘 要：应用程序往往需要使用存储过程进行数据表的大批量处理，存储过程一旦激活，前台用户就处于透明等待状态，既不了解处理进度、也不能中断处理过程，影响了应用系统的交互性。本文介绍进行动态监控存储过程中数据处理的实现方法，并给出具体的实现流程。

关键词：应用程序 异步执行 监控 存储过程

1 引言

存储过程是数据库的主要对象之一，它是服务器端需要反复使用的T-SQL语句的集合，虽然存储过程使用非过程化的T-SQL语句，但它实质上是过程化的，存储过程中可以包含程序流、逻辑控制流、接受输入输出参数、并返回单个或多个结果集或返回值；使用存储过程进行数据处理主要是基于四个方面的理由：一是处理性能好，存储过程在数据库服务器上运行，与已存储的数据记录在同一系统上，不必等待记录通过网络传递就可以进行处理，提高了数据处理效率；二是减少网络流量，一个需要数百行 T-SQL 代码的操作由一条执行过程代码的单独语句就可实现，而不需要在网络中发送数百行代码；三是可维护性好，由于存储过程具有模块化程序设计结构，在应用程序开发中可由数据库编程方面有专长的人员来开发，并独立于程序源代码而单独修改；四是简化安全管理，存储过程可作为安全机制使用，只要授予用户执行存储过程的权限，而不必对其中数据操纵语句逐一进行授权，大大简化了系统权限设置过程中繁杂的授权程序。

如果能够为用户提示存储过程执行的实时进度消息，并允许用户在非常情况下中断存储过程的执行，是改善应用程序交互性和可操作性的一项重要举措，本文将详细介绍实现这一交互过程的具体技术。

2 技术实现方法

2.1 基本概念

（1）游标。游标是数据库中对Select语句所产生结果集进行每次一行或部分行处理的机制，游标可以通过T-SQL语句在数据库内直接产生，也可以通过应用程序编程接口（API）游标函数产生，在存储过程中进行大数据处理时需要通过游标完成对数据结果集的逐行处理。

（2）事务。事务是数据库的基本工作单元。如果某一事务成功，则在该事务中进行的所有数据更改均会提交，成为数据库中的永久组成部分。如果事务遇到错误且必须取消或回滚，则所有数据更改均被清除。事务可以有效支持数据处理的原子性、一致性、隔离性及持久性，因此，要中断存储过程的执行进程，必须通过在其中定义显式事务结构（Begin Transaction/ Rollback Transaction）来撤销已执行的T-SQL语句对原始数据的改变。

（3）并发控制。当许多人试图同时修改数据库内的数据时，必须执行控制系统以使某个人所做的修改不会对他人产生负面影响，这称为并发控制。由于应用程序与数据库的交互主要是相互间数据信息的传递，因此需要采用特定的并发机制以防止读写数据过程中的冲突。

（4）异步执行模式及相关事件。应用程序通常是通过同步执行模式将存储过程执行请求提交到后台数据库服务器，在存储过程未执行完毕之前，前端应用程序处于等待状态，不能进行其他任何操作，因此要想进行中断请求、刷新屏幕提示信息等操作必须采用异步执行模式，在异步执行模式下，执行请求一旦提交，应用程序就可以进行其他处理，此后根据各种事件进行相应的处理工作，如时钟Clock事件（用于定时进行屏幕刷

新）、中断按钮Btn_Click事件（进行提交中断处理请求）、执行开始前WillExecute 事件（有关处理准备）、执行完毕事件ExecuteComplete（进行后处理）。

2.2 设计思路

（1）建立共享数据控制表，实现数据双向交换。存储过程虽然可以返回执行结果，但这一结果在执行完毕之后前端应用程序才能获得，只有寻找通过其他途径将执行进度信息返回给应用程序；同时，应用程序异步执行模式下提交存储过程执行请求后，也需要一个数据交互途径让存储过程得到数据中断请求，从而终止执行进程、并回滚已经进行的数据处理。因此，在存储过程与应用程序之间应该存在一个双向读写的数据交换区，这个数据交换区可以通过在数据库中定义一张共享的数据控制表来实现，其构成数据包括存储过程执行进度信息和中断状态信息组成，存储过程负责写入执行进度信息、由应用程序读取并刷新到交互界面上，而中断状态信息由应用程序写入、由存储过程执行过程中读入并检测，实现了数据的双向交换。

（2）设置两条记录并控制加锁粒度，解决并发控制问题。在数据交互过程中，由于存储过程在不断地向共享数据控制表写入进度信息并读取中断状态信息，应用程序也在定时（1秒一次）从共享数据控制表中读取数据，因此，必须解决对共享数据控制表的并发问题。数据库管理系统为了满足并发控制的需要，会自动对资源进行自动加锁与解锁操作，锁存在不同的级别,锁的粒度由小到大分为：行、页、键、索引、表、数据库等级别，同时在事务处理过程中，加锁后的资源要到事务处理提交或回滚之后才会解锁，此外，锁级别还会根据系统资源占用情况，动态地改变锁的级别。因此，对共享数据控制表中一条记录中的两个字段分别进行读写将受到锁的影响，必须使用两条记录并将锁粒度控制到行级，才能满足存储过程写进度信息、应用程序写中断状态信息的要求，才能解决锁冲突的问题。

（3）存储过程采用显式事务结构模式，支持用户中断请求时回滚事务处理。事务回滚的情况一般发生在出现并发冲突、出现数据唯一性、有效性、一致性校验过程中，在存储过程进行数据处理过程中需要支持用户中断请求，就必须采用显式事务结构：事务开始/回滚事务（用户中断请求发生时）/事务提交，才能在响应中断请求时，将数据恢复到初始状态。

（4）以游标方式建立数据处理结果集，实现对数据的逐行处理。在存储过程中使用T-SQL创建游标语句产生待处理的数据结果集，既可以提高数据处理效率，也可以根据数据逐行处理情况形成有效的进度提示信息，如“当前处理记录数x/总记录数N”。

（5）建立两个数据连接对象，实现对不同数据处理的请求。应用程序向后台数据库提交数据处理请求是通过数据连接对象（Connect）进行的，数据连接对象提交数据处理请求时，可以采用同步或异步执行模式进行，但是无论采用同步或异步执行处理请求，在当前连接上的数据处理未执行完毕之前，后台数据库不接受其他数据处理请求，因此，必须建立两个数据连接对象，一个连接对象以异步执行模式请求存储过程执行数据处理，另一个连接对象以同步处理模式读取共享数据库控制表中的进度信息、并在用户发出中断请求时提交更新共享数据控制表中中断状态值处理请求。

（6）启用一个时钟组件，定时读入进度数据并刷新屏幕提示信息。前端应用程序需要增加时钟组件，在异步执行开始前事件中激活，并在异步执行结束事件中关闭，时钟组件定时间隔可以根据需要进行设定，一般时钟间隔可以设定在1－3秒钟。在定时间隔事件发生时通过一个数据连接对象读回共享数据控制表中的进度信息，并刷新到屏幕上。

（7）增加一个按钮，响应用户中断请求。前端应用程序中增加一个按钮，设置其提示信息为“中断”，并在按钮单击（BTN_CLICK）事件中加入请求数据库更改共享数据控制表中断状态值处理。

2.3 交互处理流程实现框架

（1）实现框架结构。前端基于事件处理的应用程序与后台数据库管理系统的存储过程进行共享数据控制表中数据双向处理的实现框架如图1所示。

（2）共享数据控制表。共享数据控制表是后台数据库中的一张数据表，包含两条记录，分别保存前端应用程序写入的中断状态值和存储过程写入的执行进度信息。

——存储过程写进度信息字段

存储过程每处理完一条数据，向共享数据库控制表（TBO）的第一条记录中对进度信息（ZPrompt）字段进行写操作，写入时加行级锁定（With RowLock）。

——应用程序读进度信息字段

应用程序通过时钟组件定时从共享数据控制表的第一条记录进度信息（ZPrompt）字段进行读操作，读过程中不加锁（With NoLock）。

——应用程序写中断状态字段

共享数据控制表的第二条数据记录中中断状态（ZState）字段初始值为0，当用户中断请求时，应用程序请求数据库更新其值为1，更新时加行级锁定。

——存储过程读中断状态字段

每次处理新记录行时，存储过程读入中断状态值，如果为1，则回滚事务，否则正常执行数据处理，读过程中不加锁。

（3）存储过程数据处理流程。存储过程基于事务处理结构，并通过游标（Declare Cursor）进行数据结果集的逐行处理，在检测到中断状态值改变为1后，进行回滚事务处理，正常执行完数据结果集后，进行事务提交，保障数据处理的完整性。

（4）应用程序处理事件。前端应用程序

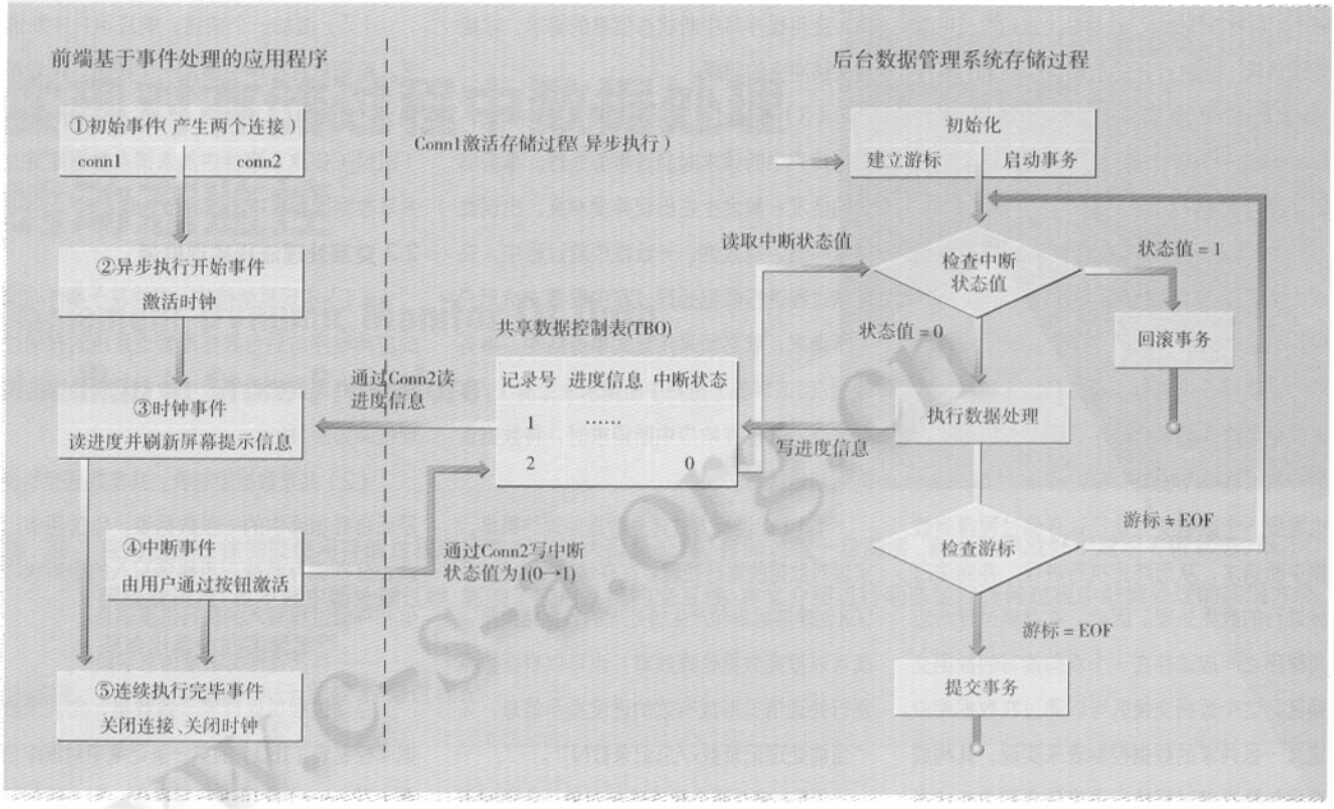


图 1　数据双向处理的实现框架图

包括五个事件，除中断事件由用户单击按钮产生外，其他事件是系统自动产生的。

——初始化事件。

由系统产生，当进度监控窗口激活时发生此事件，完成两个数据连接对象（Conn1和Conn2）的创建工作，并通过Conn1以异步执行方式（AsyncExecute）请求数据库执行存储过程。

——异步执行开始前事件。

由系统产生，连接对象1执行请求发出后产生此事件，在此事件中激活时钟组件、设定时钟事件发生间隔。

——时钟间隔事件。

由系统产生，时钟间隔一到就产生此事件，通过数据连接对象Conn2从数据库中读回进度信息，并刷新到窗口上。

——用户中断事件。由用户产生，当用户单击窗口上按钮时发生，通过数据连接对象Conn2改写中断控制状态信息。

——连接执行完毕事件

由系统产生，当数据连接对象1上的请求执行完毕后产生，此时进行释放对象、提示处理总计时间等后期操作。

3　应用实例

3.1 实例描述

（1）记帐处理。本文以会计核算系统中记帐处理作为示例，其处理流程是对凭证表中已稽核分录数据逐行进行处理，根据分录借贷会计科目记入帐本数据表中对应科目下，形成帐本科目中借贷发生额并计算新的余额。示例中显示了存储过程对698条凭证分录进行记帐的过程，正常处理结束需要执行116秒，同时显示了用户在进行到第307条分录处理时中断了存储过程的执行进度。

（2）编程环境。后台数据库管理系统采用MS SQL Server 2000、Web服务器采用IIS5.0。

应用程序开发采用DHTML应用程序进行，DHTML应用程序是Visual Basic 6.0中支持开发的一类应用程序，它是运行于IIS服务器上的一组HTML页面，最终用户使用IE4.0以上版本操作服务端的DHTML应用程序，能够自动根据服务端程序版本进行客户端透明升级。

3.2 应用程序中的事件编程

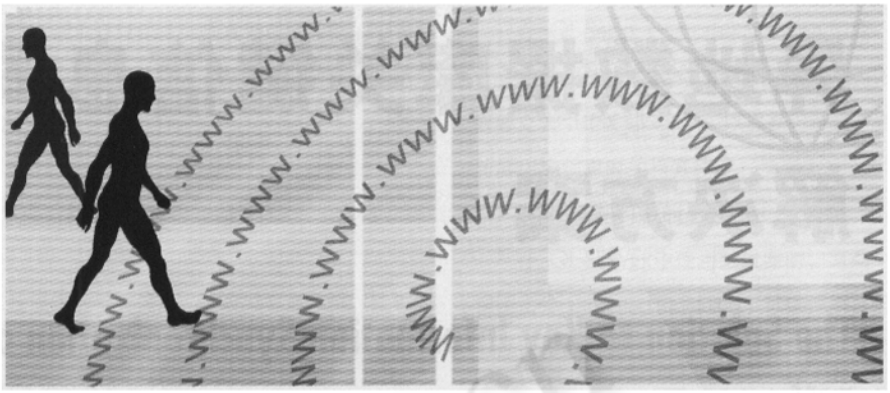
（1）窗口初始化事件

```
Private Sub BaseWindow_OnLoad
Conn1.ConnectionString="Driver={SQL Server};server=DB01;database= DBCW2003"
Conn1.Open Conn1.ConnectionString, gUser,gPassword
Conn2.ConnectionString="Driver={SQL Server};server=DB01;database= DBCW2003"
Conn2.Open Conn2.ConnectionString, gUser,gPassword
Conn1.Execute 记帐, tRecordAffected,
```

```
adAsyncExecute
    End Sub
  (2) 异步执行开始事件
  Private Sub connEvent_WillExecute
    UTimer1.Enabled = True
    UTimer1.Interval = 100
  End Sub
  (3) 时钟间隔事件
  Private Sub UTimer1_Timer
    Lable1.Caption = Conn2.Execute "Select ZPrompt From TB0 With (nolock) Where ZNo=1"
  End Sub
  (4) 用户中断事件
  Private Sub Button1_ Click
  Conn2.Execute "Update TB0 With (rowlock) Set ZState='1' Where ZNo=2"
  End Sub
  (5) 连接执行完毕事件
  Private Sub connEvent_ExecuteComplete
  UTimer1.Enabled = False
  Conn1.Close
  Conn2.Close
  End Sub
```

3.3 存储过程编程

```
  CREATE Procedure 记帐
  (@Zyf char(2))
  /*输入参数：记帐月份*/
  As
  Select @TotalRec=Count(*) from 凭证分录表 Where Zy=@Zyf
  Declare 分录游标 Scroll Cursor For Select * From 凭证表 Where Zy=@Zyf
  Begin Tran  /*启动事务*/
  Select @CurRec='1'
    Fetch First From 分录游标 INTO 字段变量集
      While  @@FETCH_STATUS=0
/*判断结果集是否为空*/
  Begin
    Select @i=ZState From TB0 with (nolock) Where ZNo=2 /*读中断状态值*/
  IF @i= '1'
   Begin
  Rollback /*回滚事务*/
  Return
  End
    Else
      Begin
  Select @PStr='当前正在处理凭证分录：'+@CurRec+'/分录总数：'+@TotalRec
    Update TB0 With (rowlock) Set ZPrompt=@PStr Where ZNo=1  /*写进度信息*/
      记帐语句.....
  /*产生发生额、计算科目余额 */
    Fetch Next From 分录游标INTO 字段变量集
      Select @CurRec=@CurRec+1
    End
  Continue
  End
  Commit Tran /*提交事务*/
  Return
```



4 结束语

对存储过程中数据处理的动态监控具有很强的实用性，能够让用户及时掌握数据处理进度，也支持用户在特定情况下中断数据处理进程，无论是客户/服务器或是浏览/服务器体系结构，不管是在Windows、Linux或Unix环境下,均可以采纳本文实现的技术和提出的架构方法，显著地提高基于后台型数据库使用存储过程进行数据处理的各类应用程序的交互性能。

参考文献

1 飞思科技产品研发中心，SQL Server 2000 数据处理技术，电子工业出版社,2001,6。

2 Dejan Sunderic,Tom Woodhead.SQL Server2000 高级编程技术，清华大学出版社,2002,2。

3 微软. SQL Server 联机丛书，2000。