SQL Server 存储过程

调试指南

SQL Server 存储过程调试指南

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中。用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。有时人们将存储过程称为“数据库中埋头苦干的老黄牛”，它是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。但是无论编写还是调试存储过程都是一项复杂的工作，因此在本次技术手册中，我们将对 SQL Server 存储过程的调试进行详细的介绍，包括了基础的调试方法和在调试过程中出现的 T-SQL 性能问题和解决方法。

SQL Server 存储过程调试基础

本部分介绍了在 SQL Server 中使用异常处理调试存储过程的方法，并对怎样调试 T-SQL 存储过程进行了详细介绍，通过对基础的学习，相信您会对存储过程的基础有个更加深刻的了解。

- SQL Server 中使用异常处理调试存储过程（一）
- SQL Server 中使用异常处理调试存储过程（二）
- 怎样调试 T-SQL 存储过程（一）
- 怎样调试 T-SQL 存储过程（二）
- 怎样调试 T-SQL 存储过程（三）

T-SQL 性能问题和解决方法

当应用程序用户开始遇到性能问题时，一般他们会联系数据库管理员并询问是否数据库存在问题。导致严重性能问题的往往是编写不当的 Transact-SQL （T-SQL）代码。因此，你必须找出确定性能糟糕的查询并对它们进行优化。

- SQL Server 中使用游标进行行处理
- 存储过程与嵌套查询优化
- SQL Server 中的视图与 UDF 性能问题
- SQL Server 不必要的记录锁
- 解决 SQL Server 触发器滥用问题
- 如何诊断和修复 T-SQL 问题

SQL Server 存储过程的修改与变更

在对 SQL Server 存储过程进行修改和变更时，往往会遇到各种各样的错误和问题，这可能是由于与现有数据库函数冲突造成的。本部分介绍了如何对存储过程进行升级与批量修改，而不造成错误的方法。

- 批量编辑 SQL Server 存储过程
- 在 SQL Server 2005 中升级存储过程

SQL Server 中使用异常处理调试存储过程（上）

异常处理被普遍认为是 T-SQL 脚本编程中的最弱的方面。幸运的是，这一点在 SQL Server 2005 中得到了改变，因为 SQL Server 2005 支持结构化异常处理。本文首先关注新特性“TRY……CATCH”的基本构成，然后在 SQL Server 2000 和 SQL Server 2005 中对照着看一些 T-SQL 的例子，这些例子中使用事务代码故意制造了一些违反约束限制的情况。将来的文章会继续探讨这一主题。

在 SQL Server 之前的版本中，你需要在执行 INSERT，UPDATE，DELETE 之后立即检查全局变量“@@error”来处理异常，如果“@@error”变量不为零的话（表示有错误），就接着执行一些纠正动作。开发人员常常重复这种与业务逻辑无关的代码，这会导致重复代码块，而且需要与 GOTO 语句和 RETURN 语句结合使用。

结构化异常处理为控制具有许多动态运行时特性的复杂程序提供了一种强有力的处理机制。目前，这种机制经实践证明是良好的，许多流行的编程语言（比如：微软的 Visual Basic.Net 和 Visual C#）都支持这种异常处理机制。接下来你会在例子中看到，采用了这种健壮的方法以后，会使你的代码可读性和可维护性更好。TRY 块包含了可能潜在失败的事务性代码，而 CATCH 块包含了 TRY 块中出现错误时执行的代码。如果 TRY 块中出现了任何错误，执行流程被调转到 CATCH 块，错误可以被处理，而出错函数可以被用来提供详细的错误信息。TRY……CATCH 基本语法如下：

```
BEGIN TRY
RAISERROR （'Houston, we have a problem', 16,1）
END TRY
BEGIN CATCH
SELECT ERROR_NUMBER（） as ERROR_NUMBER,
ERROR_SEVERITY（） as ERROR_SEVERITY,
ERROR_STATE（） as ERROR_STATE,
ERROR_MESSAGE（） as ERROR_MESSAGE
END CATCH
```

注意上面脚本中函数的用法，我们可以用它们代替局部变量和（或者）全局变量。这些函数只应该被用在 CATCH 块中，函数功能说明如下：

ERROR_NUMBER（） 返回错误数量。

ERROR_SEVERITY（） 返回错误严重等级。

ERROR_STATE（） 返回错误状态号。

ERROR_PROCEDURE（） 返回出错位置存储过程或者触发器的名称。

ERROR_LINE（） 返回程序中引起错误的行号。

ERROR_MESSAGE（） 返回错误信息的完整文本。错误内容包括可替换参数的值，比如：长度，对象名称或者时间。

我会先用 SQL Server 2000 演示一个简单例子，然后演示一个 SQL Server 2005 异常处理的例子。

下面是一个简单的存储过程示例，先用 SQL Server 2000 编写，然后改用 SQL Server 2005 实现。两者都从简单的表开始，我们在对这些表执行插入操作时会违反约束限制。下面是表结构：

```
create table dbo.Titles
 (TitleID int Primary Key identity,
TitleName nvarchar (128)  NOT NULL,
Price money NULL constraint CHK_Price check  (Price > 0) )
create table dbo.Authors
 (Authors_ID int primary key identity,
au_fname nvarchar (32)  NULL,
au_lname nvarchar (64)  NULL,
TitleID int constraint FK_TitleID foreign key
references Titles (TitleID) ,
CommissionRating int constraint CHK_ValidateCommissionRating
Check  (CommissionRating between 0 and 100) )
create table dbo.Application_Error_Log
 (tablename sysname,
```

```
    userName sysname,
    errorNumber int,
    errorSeverity int,
    errorState int,
    errorMessage varchar（4000）)
```

（作者：Serdar Yegulalp 译者：冯昀晖 来源：TT 中国）

SQL Server 中使用异常处理调试存储过程（下）

点击这里获取存储过程 P_Insert_New_BookTitle_2K 的源代码。你可以看到，这个存储过程包含了非结构化的错误处理代码，这是我们在 SQL Server 2005 之前使用的方式。

我们已经先看到了存储过程 P_Insert_New_BookTitle_2K 中使用的代码。你顶多能说：“至少我们有异常处理。”下面的语句执行这个 SQL Server 2000 下的存储过程。

```
exec P_Insert_New_BookTitle_2K 'Red Storm Rising',16.99,
'Tom','Clancy', 200
```

在用指定的参数执行存储过程时，对 Authors 表的插入失败了，因为佣金费率值无效。我们的约束检查发现了该无效值，我们可以看到如下错误信息：

Msg 547, Level 16, State 0, Procedure P_Insert_New_BookTitle, Line 23 The INSERT statement conflicted with the CHECK constraint "CHK_ValidateCommissionRating". The conflict occurred in database "Adventureworks2005", table "dbo.Authors", column 'CommissionRating'. The statement has been terminated.

这里的问题是我们不能阻止这些消息被送到客户端。所以判断哪里出错的重担就放到了客户端的头上。令人遗憾的是，在有些情况下，这样的结果对于一些不使用约束限制的应用程序可能足够了。

我们再来试一次，这次我们使用 TRY……CATCH 代码块。

点击这里获取存储过程 P_Insert_New_BookTitle_2K5 的源代码。在这段新改进的存储过程中，我们看到使用了 TRY……CATCH 代码块的结构化错误处理：

要注意 SQL Server 2005 异常处理代码是经过简化的，因此具有更好的可读性和可维护性。不需要剪切和粘贴异常处理代码，也不需要使用 GOTO 语句。执行该存储过程时，你可以看到如下结果：

```
exec P_Insert_New_BookTitle_2K5 'Red Storm Rising',16.99,
'Tom','Clancy', 200
```

我们用指定的参数执行存储过程，同样因为佣金费率值无效，对 Authors 表的插入失败了。错误发生时，程序执行流程跳转到了 CATCH 代码块，在 CATCH 代码块中我们回滚了事务，然后用 SQL Server 2005 自带的函数给 Application_Error_Log 表插入一行日志。

新的 TRY……CATCH 代码块无疑使编写处理错误代码更容易，它还可以在任何时候阻止错误信息发送到客户端。当然这可能需要 T-SQL 程序员的编程思维有一个转变，这是一个绝对有必要使用的特性。要记住迁移 SQL Server 2000 代码到 SQL Server 2005 时，如果程序的错误处理机制已经设计为旧的发送错误到客户端的方式，那你可能不得不修改应用程序了。从长远来看，我相信为这种潜在的问题付出努力重新设计是值得的。

（作者：Serdar Yegulalp 译者：冯昀晖 来源：TT 中国）

怎样调试 T-SQL 存储过程（一）

我会执行（或者单步执行）一个 T-SQL 存储过程示例程序，并在调试过程中执行以下操作：给输入参赋值，监视变量内容，在运行过程中跟踪存储过程的逻辑流程，估算 T-SQL 表达式的值，查看储存过程输出内容，设置断点，大体检查环境状态。（后面的技巧文章会继续探讨这些主题）。我们演示示例存储过程时，不是在 Management Studio 中调试，而是在 Visual Studio 2005 开发环境中调试。我提到这一点是因为在 SQL Server 2000 下，我们可以使用 Query Analyzer 调试存储过程。可能将来 Management Studio 也会增加对调试功能的支持。

T-SQL 示例存储过程：P_DisplayProductDetails

我们使用的存储过程示例使用函数来根据单价给每一个产品子类排序，从 AdventureWorks 数据库中查询产品明细信息。该存储过程接收产品类别名作为一个可选入参。几个输出参数会给调用它的分支返回有用的信息。最后，正如前面的文章讲到的，该存储过程使用了“结构化异常处理”。

```
Use AdventureWorks
GO
IF EXISTS （SELECT * FROM sysobjects WHERE type = 'P' AND
name = 'P_DisplayProductDetails'）
DROP Procedure P_DisplayProductDetails
GO
CREATE Procedure P_DisplayProductDetails
 （@Categoryname varchar（50） = NULL,
@MatchingRows int = NULL OUTPUT,
@ErrorString varchar（128） = NULL OUTPUT,
@ErrorNumber int = NULL OUTPUT）
as
BEGIN TRY
-- 添加一个‘%’（模糊查询），这样调用者不需要知道子类的完整名称
if @CategoryName is null
select @CategoryName = '%'
else
```

```
select @CategoryName = @CategoryName + '%'
-- 使用 rank 函数，根据子类名称按 ListPrice 字段排名
-- DENSE_RANK 函数分配相邻值
SELECT Production.Product.ProductID,
Production.Product.Name AS ProductName,
Production.ProductCategory.Name AS CategoryName,
Production.ProductSubcategory.Name AS SubcategoryName,
Production.Product.ListPrice,
DENSE_RANK () over (Partition by
Production.ProductSubcategory.Name ORDER BY
Production.Product.ListPrice DESC) as PriceRank
FROM Production.Product
INNER JOIN Production.ProductSubcategory
ON Production.Product.ProductSubcategoryID =
Production.ProductSubcategory.ProductSubcategoryID
INNER JOIN Production.ProductCategory
ON Production.ProductSubcategory.ProductCategoryID =
Production.ProductCategory.ProductCategoryID
WHERE Production.ProductCategory.Name like @CategoryName
ORDER BY Production.ProductCategory.Name
select @MatchingRows = @@ROWCOUNT
return 0
END TRY
BEGIN CATCH
--把错误信息记录日志……在调试时，我们可能会跳过这一步!
insert dbo.Application_Error_Log (UserName, errorNumber,
errorSeverity, errorState, errorMessage)
values (suser_sname (), ERROR_NUMBER (),ERROR_SEVERITY (),
ERROR_STATE (), ERROR_MESSAGE () )
SELECT @ErrorNumber = ERROR_NUMBER (),
@ErrorString = ERROR_MESSAGE ()
RAISERROR (@ErrorString, 16,1)
END CATCH
```

(作者：Serdar Yegulalp 译者：冯昀晖 来源：TT 中国)

怎样调试 T-SQL 存储过程（二）

从哪里开始调试存储过程

在进行任何调试动作之前，你必须先创建该存储过程。我们可以用 Management Studio 中的“新建查询”或者通过使用 Visual Studio 2005 图形化方式创建该存储过程。在 Visual Studio 中，可以打开一个数据库项目，然后用 Visual Studio 已安装的模板来创建存储过程。

存储过程创建好了以后（不管是用哪种方式创建的），你就可以开始在 Visual Studio 2005 中调试了。在为调试目的启动 Visual Studio 时，你不需要创建一个新项目。取而代之的是，你可以在 Server Explorer 中创建一个到 AdvertureWorks 数据库的连接，就像我在下图中做的一样。如图 1。（你必须提供服务器名以及登录验证，然后选择 AdventureWorks 数据库。）然后你可以展开树形结构，打开存储过程文件夹。接下来，右击你想调试的存储过程，然后在右键菜单中选择“单步调试存储过程”。然后你就可以启动调试了！

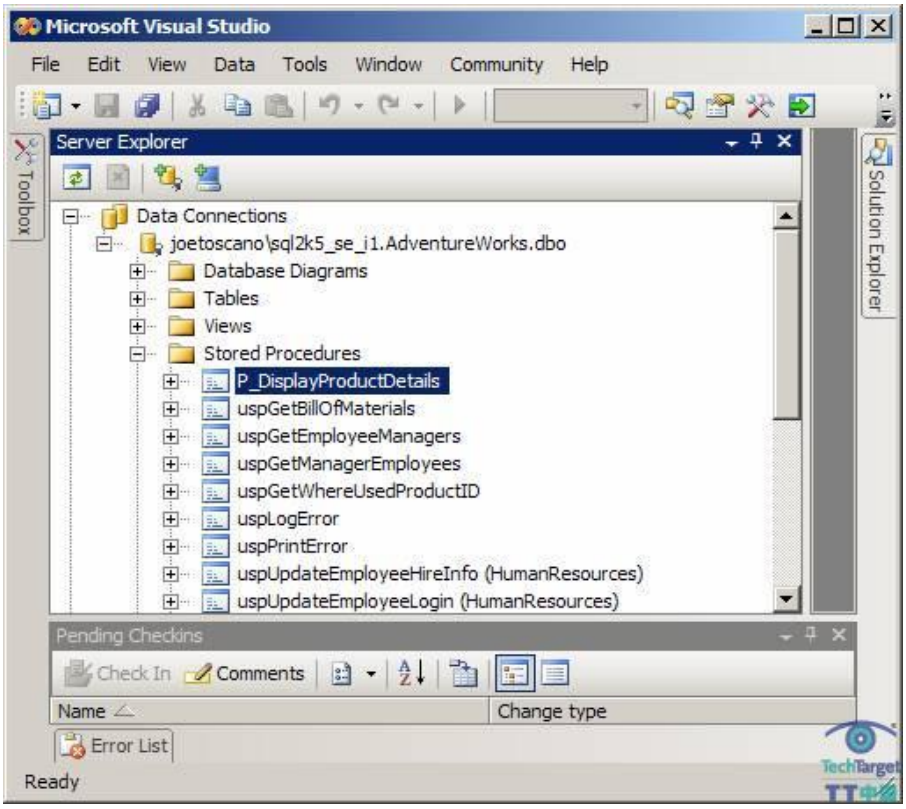

图 1：在 VS 2005 中怎样启动调试

怎样运行存储过程

在执行单步调试存储过程时，你实际上是在告诉 Visual Studio 一行一行地运行存储过程。既然我们的存储过程示例不接受入参，你会看到一个 Local Window（局部变量窗口），在这个窗口中你可以滚动浏览存储过程中的局部变量和参数。如图 2，在表头 Direction 列下面，你可以发现 Visual Studio 给你识别出了输出参数，Value 列是该表中唯一你可以修改的列。在这个例子中，我输入 Bike 作为 CategoryName 的一个值。

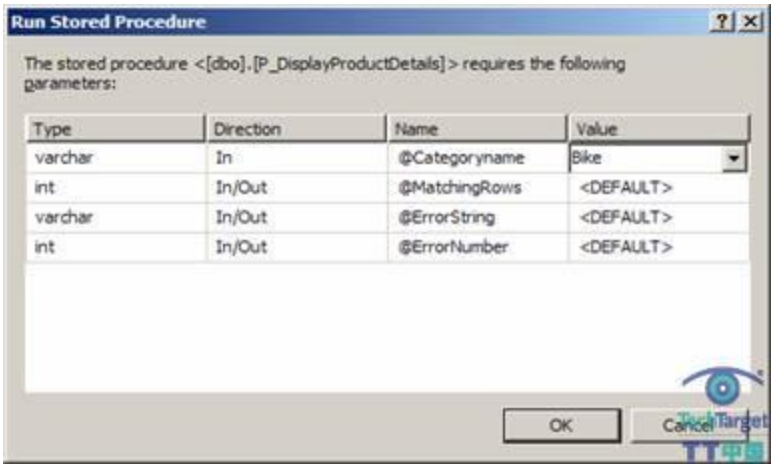

图 2：输出参数

Visual Studio 提供了许多窗口来查看运行环境的状态。代码中即将运行的行被标注了一个黄色的箭头。如下图 3 所示。

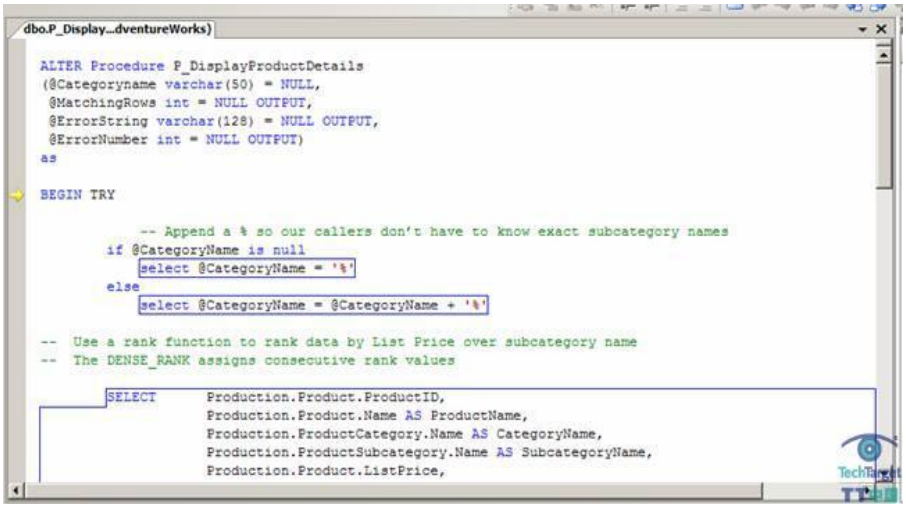

图 3：待执行代码

大部分时候，你会在存储过程中使用单步执行或者跳过执行命令。下面的命令都可以应用到 T-SQL 单行：

单步执行 Step Into（F11）:用来单步执行代码。（黄色箭头会一行一行向下移动。）

跳过执行 Step Over（F10）：如果有的代码行可能修改数据或者调用其他存储过程，但是你在调试过程中不关心该部分内容，那么就可以使用跳过执行。例如：你可能想跳过执行审计的代码。

跳出执行 Step Out（SHIFT-F11）：执行存储过程的剩余部分，中间不停顿。

运行到光标位置 Run to Cursor（CTRL-F10）：把光标定位到代码中的一点，然后摁 CTRL-F10 执行当前光标位置前的全部代码。

继续执行 Continue（F5）：重新开始执行，运行到完成或者运行到下一个断点（可以很快跳过多个断点）。

（作者：Serdar Yegulalp 译者：冯昀晖 来源：TT 中国）

怎样调试 T-SQL 存储过程（三）

Visual Studio 调试窗口

Visual Studio 给我们提供了许多信息丰富的调试窗口。我会把我们使用的示例代码过一下，演示几个这样的窗口。我们从 Autos 窗口开始。

Autos 窗口显示当前语句中使用的变量。当黄色箭头指向“select @CategoryName = @CategoryName + '%'”这一行代码时，注意看“@CategoryName”的值实际上是该语句执行前的值。在本例中，我们给这个参数值后面追加了一个百分号，如图 4.

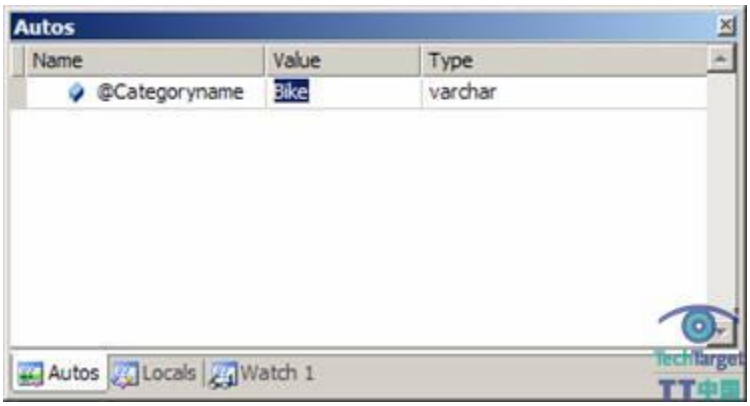

图 4：Autos 窗口

Locals 窗口显示当前局部变量和参数，你可以在代码执行过程中在这个窗口里交互式地修改这些变量的值。该窗口会把改变的值着色显示。如图 5，你可以看到我把“Bikes”改成了“Clothing”，为了便于在 Visual Studio 中查看，我把它标为红色。

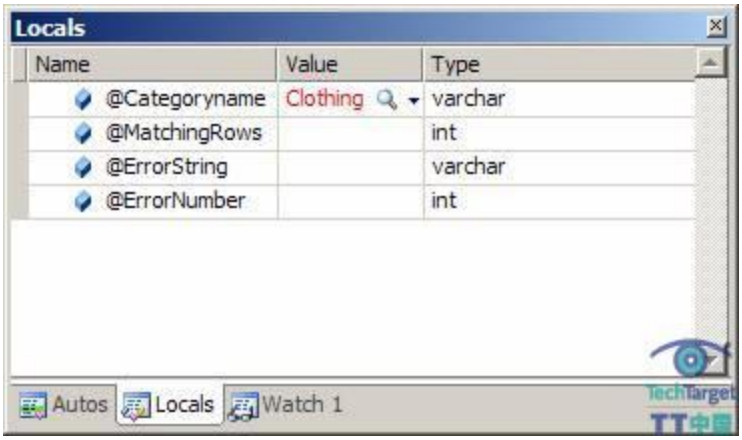

图 5：本地变量窗口

监视窗口支持你输入或者从代码中拖拽 T-SQL 表达式进来，在该窗口中估算该表达式代码的实际值。如果你想检查条件表达式（像 IF，WHILE 或者 CASE）所包含表达式的值，这一点可能很有用。实际上，你可以同时使用多达四个监视窗口。

输出窗口显示查询语句或者输出语句返回的结果集。在我们的例子中，我们返回了 35 个匹配“Clothing”的行数据（见图 6）。

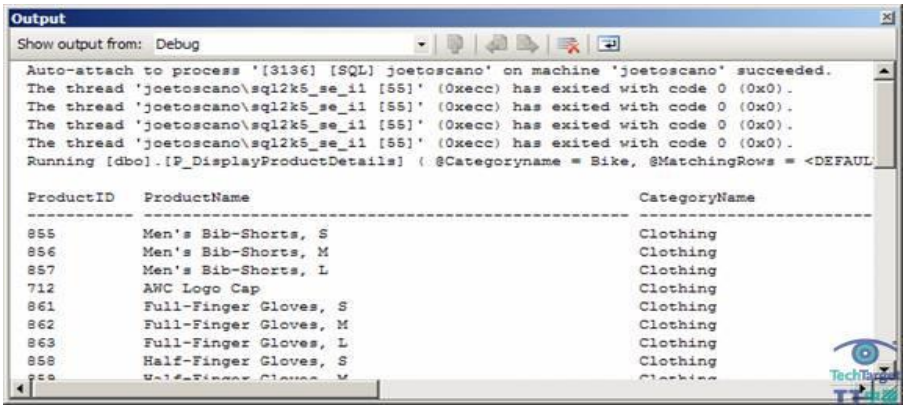

图 6：输出窗口

悬浮值窗口不是一个真正的窗口，但是它是一个特别有用的特性，非常值得一提。如果你在调试模式下，把光标悬浮在代码行上，你会看见与该行有关变量的值。这一点看起来与 Windows 应用程序的功能类似：在 Windows 应用程序中，你把光标悬浮到工具栏图标时，不需要点击就可以看到帮助信息。

断点窗口显示了当前所有断点，而且允许你添加新断点。断点是用户定义的代码位置和（或者）暂停执行的条件，允许调试人员停下来思考，查看等等。你可以通过点击代码窗口左侧边缘部分添加断点。要注意在图 7 中显示的断点窗口实际上被分成了两个窗口。靠下的窗口提供了断点信息，告诉我们在第 20 行有一个断点。图 7 中靠上的窗口包含一个代码片段。在上面的窗口中我们可以看到在第 20 行有一个红色的泡泡，它就是 Visual Studio 的断点标识。

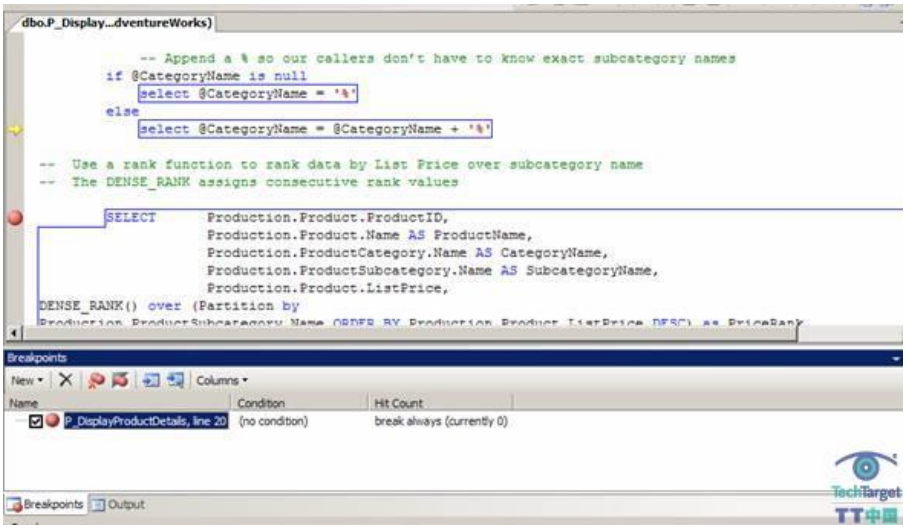

图 7：断点窗口

结论

Visual Studio 2005 为你的 T-SQL 存储过程提供了非常容易使用的图形化调试工具。你可以利用它对你的代码进行单元测试。因为开发人员典型的单元测试代码缺少所有支持的存储过程或者函数时，Step Over 选项真的迟早会有用的。而且既然你可以很容易改变入参或者局部变量的值，你就可以强行改变程序的执行路径分支，这样就可以执行代码的特定部分。

（作者：Serdar Yegulalp 译者：冯昀晖 来源：TT 中国）

SQL Server 中使用游标进行行处理

当应用程序用户开始遇到性能问题时，一般他们会联系数据库管理员并询问是否数据库存在问题。有趣的是，大多数用户都会查看内存、CPU 和磁盘使用率，而不是造成严重性能问题的特定的代码模块;很不幸，导致严重性能问题的往往是编写不当的 Transact-SQL （T-SQL）代码;而且，通过在服务器上增加更多的内存来解决这些问题，而不对问题的根源进行修复，将直接导致更大的问题。因此，你必须找出确定性能糟糕的查询并对它们进行优化。

T-SQL 是一个强大的程序语言，它可以读取和修改数据、改变 SQL Server 设置、创建和修改数据库对象、修改 Registry 设置等等。然而，必须谨记的是，没有任何一个 T-SQL 功能可以适应所有环境和所有应用的。比如，如果你的应用需要在不同网络共享中复制大量的文件，那么 T-SQL 就不是你的最佳选择了。类似的，使用 SQL Server 的内置电子邮件功能来发送多兆字节附件的电子邮件并不是一个好主意。相反，如果你需要在数据库中检索数据行或者修改所有满足特定条件的行，那么 T-SQL 则是你的最佳选择。由于 T-SQL 支持大量的功能，因此我无法全面的阐述可能造成性能问题的所有具体情况。相反，我将介绍某些情况下一种编写代码的方法将可以改善其它方面的性能。

使用游标进行行处理

游标可能是最常见的引起性能问题的原因，虽然它们对于某些任务而言是一个强大且方便的方法。比如，假设你有一个复杂的存储过程，其中你必须反复调用存储在临时表中的每一行数据。你可以使用光标从临时表中检索参数值，同时对每一个检索的值都调用一次存储过程。基本上，游标允许你处理数据集中的每一行数据，同时对数据记录进行相同的逻辑处理，每次处理一行数据。然而，T-SQL 是一个基于集合的语言;这种语言是针对记录集进行数据读写操作优化的，而不是针对记录行。从前端特定语言（比如 VBScript、ASP 或者 ColdFusion）开发转变过来的 T-SQL 初学者，他们往往在并不需要循环时在 T-SQL 中错误地使用循环。比如，下面的代码将酒店数据库的所有 California 作者的“姓”标记为“changed”：

```
DECLARE @last_name VARCHAR（50），
```

```
@au_id CHAR (11)
DECLARE last_name_cursor CURSOR FOR
SELECT au_id, au_lname FROM authors
WHERE state = 'ca'
OPEN last_name_cursor
FETCH NEXT FROM last_name_cursor INTO @au_id, @last_name
WHILE @@FETCH_STATUS = 0
BEGIN
UPDATE authors
SET au_lname = @last_name + ' changed'
WHERE au_id = @au_id
FETCH NEXT FROM last_name_cursor INTO @au_id, @last_name
END
CLOSE last_name_cursor
DEALLOCATE last_name_cursor
```

这段代码是可以运行的，因为只有少量 California 作者，因此它很快就完成运行。现在让我们来看看我们是否可用简单的更新语句来重写相同的逻辑：

```
UPDATE authors
SET au_lname = @last_name + ' changed'
WHERE state = 'ca'
```

应该怎么做呢？事实上，两段代码都完成了相同的任务，但是第二段代码不仅更加简单，而且它在大数据集中运行的速度大约比第一段代码快 100 倍。我曾经在过去的几个项目将游标使用方法改变为基于数据集的方法，从而将查询的持续时间从几个小时减少为不到一分钟。

（作者：Serdar Yegulalp 译者：曾少宁 来源：TT 中国）

存储过程与嵌套查询优化

使用嵌套查询是初学者所犯的典型错误。有些语言提供直接从前端应用发送到数据库的合并查询。如果你的应用用户数较少，那么你可以使用这个方法。但是，一旦你的用户是成十成百的增长，那么你一定会遇到性能问题通知（以及甚至是用户的抱怨）。

该怎么处理呢？SQL Server 具备一个非常智能的嵌套查询优化器，它可以在运行时创建查询执行规划。优化器可以缓存最近使用执行规划并重用它们。这样做将导致减少服务器的负载从而获得更好的性能。然而，优化器重用存储过程的查询规划比嵌套查询更容易。你的存储过程至少可以保证与嵌套查询一样的执行速度——99%的情况下存储过程比嵌套查询性能好。另外的 1%是极其罕见的个例，即当参数值频繁变化时，重用查询规划实际上是个糟糕的方法。那么你该如何处理呢?通常的做法是，总是使用存储过程替代嵌套查询。

（作者：Serdar Yegulalp 译者：曾少宁 来源：TT 中国）

SQL Server 中的视图与 UDF 性能问题

SQL Server 视图为 T-SQL 程序提供多种好处：它们可以用于掩盖实际的表结构，限制来自某些用户的敏感数据等等。然而，视图并不总是最好的

某些从 Oracle 转到 SQL Server 的程序员很喜欢使用视图;较老版本的 Oracle 并不允许从存储过程返回数据集，因此必须使用视图。这并不是 SQL Serve 不存在此 r 的问题，。但是，凭心而论，并不是只有 Oracle 程序员才犯这样的错误。

或许，最常见的误解是使用大量 JOIN 操作的视图比使用相同 JOIN 操作的存储过程快。事实上并不是这样的。视图对于拥有相同数目 JOIN 操作的查询并没有任何优势。事实上，视图有一个最大限制——它们不能接收参数。因此，如果你有一个连接 10 个表的视图并且这些表中的其中一个有上百万的行，那么你的视图将返回至少 1 百万的行。在存储过程中，将这样的视图连接到很少的几个表上就将成为灾难。

你该怎么做呢?在这样的情况下，一个较好的选择是用户自定义方法（UDF），它可接收参数并允许你限制返回行的数目。另外一个选择是在一个单元存储过程中连接所有表并使用参数限制输出。

自从 UDF 出现在 SQL Server 2000 中时，它就受到了 T-SQL 程序员的欢迎。UDF 允许你创建非常类似于内置方法执行的常规程序，但是它们按照你的具体业务需要来执行数据处理。比如，你可以写一个方法来计算两个时期之间的营业天数。

有三种类型的 UDF：

标量：返回唯一值

嵌套表：返回行集（或者表）

多语句：返回行集（或者表）

如果在一个搜索很大的表的查询中调用标题 UDF，那么它可能造成一个巨大的性能问题。问题是这样的，在调用 UDF 的表中，UDF 为每一行执行一次。比如，如果 Authors 表中有一百万行数据，那么下面的查询将调用 my_udf 一百万次：

```
SELECT dbo.my_udf（au_lname），
phone,
address
FROM authors
```

因此，要小心使用标量 UDF。即使你的标量 UDF 中的逻辑很简单，但在一个大的表中执行时，它也可以急剧地减慢你的查询。

你该怎么处理呢?要仔细考虑是否你真的需要标量 UDF。例如 dbo.my_udf：

```
CREATE FUNCTION dbo.my_udf （@string VARCHAR（50））
RETURNS VARCHAR（100）
AS
BEGIN
DECLARE @return_value VARCHAR（100）
SELECT @return_value = 'Mr. or Mrs. ' + @string
RETURN @return_value
END
```

执行上面调用标量 UDF 的 SELECT 查询，将导致 23 个读取（因为酒店数据库在 Authors 表中恰巧有 23 个记录）。另外一方面，下面的查询将在一个读取中返回相同的结果：

```
SELECT 'Mr. or Mrs. ' + au_lname, phone, address FROM authors
```

（作者：Serdar Yegulalp 译者：曾少宁 来源：TT 中国）

SQL Server 不必要的记录锁

如果你的应用执行缓慢，那么你可能在锁住一些不必要的记录。这对于在同一服务器上同时整合报表和交易活动的应用往往会出现。

默认情况下，SQL Server 会在执行 SELECT 语句期间锁住记录。其他的用户必须一直等到你的 SELECT 完成才可以修改数据。

报表很少需要最新的数据。例如，一个作业记录应用一般只显示之前几周或者数月的数据。这样，你可以使用锁定提示（NOLOCK）或者一个较少限制的事务孤立级别而不影响报表质量。

你应该怎样做呢?在你的报表查询引用的表中添加 NOLOCK 提示。例如，下面的查询在不锁定任何记录的情况下查询一个包含所有标题和作者的列表：

```
SELECT au_lname + ', ' + au_fname, title, price
FROM authors a (NOLOCK)
INNER JOIN titleauthor b (NOLOCK) ON a.au_id = b.au_id
INNER JOIN titles c (NOLOCK) ON b.title_id = c.title_id
```

(作者：Serdar Yegulalp 译者：曾少宁 来源：TT 中国)

解决 SQL Server 触发器滥用问题

触发器滥用

触发器是在添加、修改数据或者将数据迁移到一个指定表时执行某些动作的方法。SQL Server 2005 也同样提供数据库级的触发器，但是我在此只说明表级触发器。由于触发器会启动一个隐藏事务，因此触发器会在服务器上强加额外的开支。一旦执行触发器，一个新的隐藏事务就会开始，同时在事务中的任意数据检索将锁定受影响表。

你应该怎么处理呢？要尽量少用触发器并且尽可能使触发器逻辑简单。你可以简单地以完整性引用约束来替代某些触发器;其它触发器功能可以在存储过程中实现。

返回受影响行数

默认情况下，SQL Server 返回一个友好的消息来报告每个查询所影响的行总数。这对于调试应用或者直接在 Query Analyzer 中修改数据是一个非常好的设置。然而，你的前端应用并不需要知道受影响的行数目——它只需要数据。发送这个消息可能会引入不必要的网络负荷。

你应该如何处理呢？在你的所有存储过程中使用“SET NOCOUNT ON”来减少网络传输。

条件语句执行

往往，你需要根据传递到存储过程的参数的不同值应用不同的代码逻辑。比如，如果我的参数值是 0，那么我可以从一组表中检索值，如果参数值为 1，则可以从另一组表中检索。

```
IF @parameter = 0
BEGIN
SELECT column1, column2
FROM some_tables…
END
```

```
ELSE
BEGIN
SELECT column1, column2
FROM other_tables…
END
```

这个代码迫使我的过程在每次执行时都必须重新编译，因为 SQL Server 只有在运行时才能识别两个规划中的哪一个是有用的。

你应该如何处理呢？将上面的过程分成两个独立的过程，每个对应查询不同的表集。在你的中间层代码中确定参数值，然后再调用恰当的存储过程。

（作者：Serdar Yegulalp 译者：曾少宁 来源：TT 中国）

如何诊断和修复 T-SQL 问题

有几种方法可以诊断代码相关的性能问题，但由于篇幅局限，在本章中我将不全面的涉及。但是，你可以使用 SQL Profiler 来去掉性能糟糕的一系列常规程序。同时，你必须检查查询执行规划以便确保 SQL Server 使用最佳的规划来执行你的程序。

一旦你意识到存储过程出现了问题，那么可以采取下面的步骤来解决：

1. 创建名称稍微不同的存储过程（例如以“routine1”替代“routine”）

2. 在新过程中修改代码同时确保它比现有代码效率更高

3. 将旧的过程重新命名为“routine2”

4. 将新的过程重新命名为“routine”

如何解决嵌套查询的问题呢？你可以尝试重写现有的查询，但是，很可能你将需要重新编译和部署代码。这样做是为了同时避免嵌套查询重复和将所有 T-SQL 附加到存储过程上。

如何防范未然

制定编码规范和准则。Microsoft 提供了对这方面有所帮助的资源。

在部署之前进行代码检查。代码检查必须：

确保代码完成应该完成的任务。

确保代码符合编码规范和准则。

培训新程序员关于编码技巧和以前没遇到过的技术。

在将程序部署到生产环境之前，先全面地进行程序压力测试。

(作者：Serdar Yegulalp 译者：曾少宁 来源：TT中国)

批量编辑 SQL Server 存储过程

我维护的数据库系统广泛使用存储过程。在数据库中执行的大多数的普通任务是专门由 SQL Server 2000 和 SQL Server 2005 通过存储过程来处理的。经常可以通过简单编辑 SQL Server 中的存储过程来使得数据库系统发生全局性更改。然而，有时你需要一次性修改全部的存储过程，这就像做外科手术一样，需要尽可能地完美。当然，这里假定你不使用 Visual Studio 或其它的一些 IDE 来完成这种工作,这些 IDE 自然会节省很大量的苦差。

这是我修改 SQL Server 中一组存储过程的最基本的方法：

1.生成一个 SQL 脚本，在数据库中创建所有相关的存储过程。确保此脚本也执行了任何所需的 DROP FUNCTION 命令,以使得现存的全部存储过程复本都被删除。实现这个的最简便的方法是仅需使用 SQL Server 2000 数据库企业管理器中的“Generate SQL Script”上下文菜单选项。在 SQL Server 2005 数据库中，则是“Tasks”上下文菜单选项中的“Generate Scripts”选项。

注意，SQL Server 2005 脚本向导看起来与老版本有很大差别，且如果你正运行着 SQL Server 2005 SP1 的话，会使得你在这种问题上限入一个可能引发重复操作的麻烦。在“Script Behavior”菜单下，仅有“Generate DROP statements only”与“Generate CREATE statements only”。换言之，没有选项可在同一脚本中同时产生 DROP 和 CREATE 语句，因此现有的同名存储过程可以被优先删除。许多人抱怨这个，就我所知,它已集

成在 SQL Server 2005 SP2 中。如果 SQL Server 2005 SP2 的脚本向导的复杂性让你头疼，你并不是一个人。Bill Graziano 已经制作了一个叫作 Scriptio 的第三方小工具来解决一些这样的问题。它在 SQL Server 2000 中也运行良好。

2. 备份一下这个新脚本。这很重要，不要在原文件上来操作。

3.用你最喜欢的文本编辑器打开此脚本。查找与替换的工作越复杂，你越希望用一个有正则表达式的编辑器。在大多情况下，甚至像 Notepad2 这样简单的编辑器都可以胜任

这个工作。我强烈推荐使用编辑器来编辑存储过程，它可以高亮显示 T-SQL 语法。这样，如果你犯了语法错误，这些错误会在你偿试重新导入此脚本之前被标注出来。

4. 在这个新的，编辑过的脚本中向后合并, 需注意存储过程函数中的任何错误。如果有错误发生，你通常可以使用你创建的备份脚本将存储过程恢复到它们的原始状态。结束的时候，我要说明一下关于像数据库版本之类的东西，因为它们在不同地方可能有很大区别。

(作者：Serdar Yegulalp 译者：张峰 来源：TT 中国)

在 SQL Server 2005 中升级存储过程

最近，我介绍了在 SQL Server 2005 中怎样的更改可能会破坏老的应用程序与数据库，尤其是使用过时的函数。现在，我们将讨论在您的迁移数据库中如何应用存储过程来发挥 SQL Server 2005 中的功能特性，同时又不与现有函数发生冲突。

假设您有一个前端应用程序，由不同的团队进行写入或管理，并且后台数据库没有经常升级。您仍然想更改 SQL Server 应用程序的存储过程以利用其新特性。您还想出色地漂亮的完成这些，而不导致数据库及前端系统发生故障。

例如，您可能想升级存储过程以便应用在 SQL Server 2005 的中 T-_SQL 的新函数中，诸如 try 与 catch 函数，它们能够更容易的捕获复杂的错误。通常，您把现有的存储过程复制到一个新的、并行的存储过程，对此进行升级以应用新的函数，接着通过一个前端应用程序的修改版本(如果它可用)或者查询分析器来测试它们。

当您确信正在运转的新存储过程应该并且准备好在生产环境中可以应用时，就可以通过重命名这两个存储过程或者复制代码对它们进行无缝切换或者通过重命名这两个存储过程或者通过复制代码。我个人更喜欢进行重命名，因为那样允许您对之前的代码进行备份----与数据库一起备份以防意外万一。

可以说这是一个相当标准的技术，但是它之中还有这是一个很有趣的东西皱纹，我已经看到过关于涉及到它的可选参数：

```
    CREATE PROCEDURE my_procedure
    {other parameters go here},
    @optionalparameter Boolean=FALSE
    AS
    If @optionalparameter=TRUE
    Begin
    {NEW version of stored procedure with SQL Server 2005-
specific commands goes here}
    End
    Else
```

```
Begin
{OLD version of stored procedure goes here}
End
```

另外还有一个好用的方法来对代码进行选择性测试其他有用的可选方式进行测试这些代码。当前的前端调用存储过程将不使用可选参数，而将执行原有老代码。您可以在新的前端代码中对此存储过程进行测试，之后优雅地升级到现用的存储过程。因为参数对所有现存的调用的存储过程都是可选的(即，那些没有参数)，将如从前一模一样。

如果您不被允许增加新的存储过程，但是允许对现有的进行修改，这是一个绕过这个限制很好的的好方法方式。如果仍然没有遗留代码，就可以在存储过程中对假设进行逐步取消。最后，在存储过程与前端代码中分离这个参数。

(作者：Serdar Yegulalp 译者：司学峰 来源：TT 中国)