许多基于Windows的系统都使用了SQL Server作为后台组件。待测程序(AUT：Application Under Test)或者待测系统(SUT:System Under Test)经常通过[存储](http://storage.it168.com/)过程来访问数据库。对于这些应用场景，可以把SQL存储过程想象成应用程序的辅助函数。有两种基本的方法可以用来编写针对SQL存储过程的轻量级的自动化测试。

　　第一种方法是在原始的SQL环境中编写自动化测试代码，也就是说测试套件代码(harness code)用于T-SQL语言来编写的，并且在类似于查询分析器(QueryAnalyzer)或者Management Studio这样的程序里被执行。

　　第二种方法是在.NET环境下编写自动测试代码，测试套件代码是用C#或者其他.NET语言来编写的，并且在诸如命令行程序等常规的Windows环境下被执行。

　　此次介绍如何使用原始的SQL环境来测试SQL存储过程。

**问题**

　　如何创建SQL数据库以及表结构以用来保存测试用例的输入数据和测试结果。

**设计**

　　编写T-SQL脚本，创建一个数据库然后创建一些表用来保存测试用例的输入数据和测试结果。如果想通过SQL认证的方式连接到上一步创建的数据库，则要创建一个专用的SQL登录账号，然后在查询分析器里或者通过osql.exe运行T-SQL脚本。

**方案**

　　下面的脚本创建了一个叫dbTestCasesAndResults的数据库，它包括一个用于存储测试用例数据的表和一个胜于存储测试结果的表，以及一个专用的SQL登录账号，这个账号便于程序通过Windows认证或者SQL认证的方式连接数据库。下面来看代码。

　 　第一步设定当前的数据库为SQLServer的主数据库(master database)。当创建一个新的用户数据库的时候，这一步是必不可少的。如果要创建的数据库已经存在，系统就会给一个错误，为了防止这种错误，可以在 试图删除新数据库的旧版本之前通过查询sysdatabases表来检查旧版的数据库是否存在。然后可以通过数据库创建用于测试用例存储的数据库。用于创 建数据库的语句有许多可选的参数，但是对于轻量级的自动测试来说，接受这些参数的默认值通常就可以满足需要了。

--makeDbTestCasesAndResults.sql  
  
use master  
  
go  
  
if exists(select \* from sysdatebases where name = 'dbTestCasesAndResults')  
  
      drop database dbTestCasesAndResults  
  
go  
  
if exists(select \* from sysxlogins where name = 'testLogin')  
  
      exec sp\_droplogin 'testLogin'  
  
go  
  
create database dbTestCasesAndResults  
  
go  
  
use dbTestCasesAndResults  
  
go  
  
create table tblTestCases  
  
(  
  
      caseID char(**4**) primary key,  
  
      input char(**3**) not null,-- an empID  
  
      expected int not null  
  
)  
  
go  
  
--下面插入的是用于usp\_StatusCode的测试用例数据  
  
--也可以通过BCP，DTS，或者C#程序从文本文件读入  
  
insert into tblTestCases values('0001','e11',**77**)  
  
insert into tblTestCases values('0002','e22',**77**) ——should be **66**  
  
insert into tblTestCases values('0003','e33',**99**)  
  
insert into tblTestCases values('0004','e44',**88**)  
  
go  
  
create table tblResults  
  
(  
  
      caseID char(**4**) not null,  
  
      result char(**4**) null,  
  
      whenRen datetime not null  
  
)  
  
go  
  
exec sp\_addlogin 'testLogin','secret'  
  
go  
  
exec sp\_grantdbaccess 'testLogin'  
  
go  
  
grant select,insert,delete on tblTestCases to testLogin  
  
go  
  
grant select,insert on tblResult to testLogin  
  
go

**注解**

　　SQL数据库支持两种不现的[安全](http://safe.it168.com/)模 式：使用Windows认证可以通过Windows账号ID和密码连接数据库，使用混合模式认证可以通过SQL登录ID 和SQL密码来连接数据库。如果想要通过SQL认证来连接数据库，应该使用系统存储过程sp\_addlogin()创建一个SQL登录账号以及相应的密 码。如果要删除一个SQL登录账号，可以先查询sysxlogin()表检查相应的账号是否存在，然后调用sp\_droplogin()来删除这个账号。 创建好一个SQL登录账号以后，应该赋予登录账号连接数据库的许可。然后需要针对数据库里的表，赋予登录账号与SQL语句相关的权限许可，比 如：SELECT,INSERT,DELETE以及UPDATE。

　　SQL登录账号很容易与SQL用户搞混。SQL登录账号是[服务器](http://server.it168.com/)范 围的对象，它用来控制针对装有SQL Server的机器的连接许可。而SQL用户是数据库范围的对象，它用来控制数据库以及它所包含的表，存储过程和其他一些对象的权限许可。当为一个SQL 登录账号分配权限的时候，会自动创建一个同名的SQL用户。所以最终会有一个SQL登录账号和一个SQL用户，两个名字相同并且相互关联。尽管也可以上让 不同名的账号和用户相互关联，但是这太容易让人迷惑了，所以最好还是使用名字相同的默认机制。

　　为了测试存储过程而测试用例存储结构的时 候，必须决定在什么时候以及如何往表中插入胜于测试用例的那些数据。最简单的办法是在创建表的时候直接加入用于测试用例的数据。使用本解决方案所演示的 INSERT语句可以很快的完成这件事情。但是，在测试的过程中几乎肯定要在很多地方碰上需要添加或者移除测试用例数据的情况，所以更灵活的方法是在后面 使用BCP(Bulk Copy Program),DTS(Data Transformation Services)或者一个辅助的C#程序来插入数据。如果想要插入以及删除测试用例数据，那么就应当针对存储测试用例数据的那张表为SQL登录账号赋予 INSERT和DELETE的权限许可。

　　这个用于创建测试用例和存储测试结果的脚本可以通过几种方法来运行，一种方法是在查询分析器程序里打开并通过Excute命令。第二种方法是使用OSQL.EXE程序来执行这个SQL脚本

**问题**

　　如何使用BCP从一个文本文件把测试用例数据导入到SQL表

**设计**

　　创建一个BCP格式的文件用于把想导入的文本文件信息映射到目标SQL表，然后把上述格式的文件作为参数传给命令行工具bcp.exe。

**方案**

　　假设SQL表定义如下：

create table tblTestCases  
  
　　(  
  
　　caseID char(**4**) primary key,  
  
　　input char(**3**) not null,  
  
　　expected int not null  
  
　　)

　　还有一个包含测试用例数据的文本文件叫作newData.dat

　　0020,e13,66

　　0021,e14,77

　　0022,e15,88

　　0023,e16,99

　　0024,e17,66

　　创建一个叫newData.fmt的BCP格式的文件，这个文件采用制表符(tab)作为分隔符。

　　8.0

　　3

　　1 SQLCHAR 0 4 "," 1 caseID SQL\_Latin1\_General\_CP1\_CI\_AS

　　2 SQLCHAR 0 3 "," 2 input SQL\_Latin1\_General\_CP1\_CI\_AS

　　3 SQLCHAR 0 2 "\r\n" 3 excepted SQL\_Latin1\_General\_CP1\_CI\_AS

　　用于导入测试用例数据的命令如下：

　　C:\>bcp.exe dbTestCasesAndResults..tblTestCases in newData.dat -fnewData.fmt -S. -UtestLogin -Psecret

　 　此命令的意思是针对数据库dbTestCasesAndResults里的表tblTestCases运行BCP程序，把newData.dat里的数 据按照newData.fmt所定义的格式映射导入到上述表中。这些命令是针对本地的SQLServer所执行的，连连数据库的时候使用叫作 testLogin的SQL登录账号，SQL密码是secret。

　　使用这种技术，关键是要理解bcp.exe程序所使用的格式文件的结 构。这个文件的第1行只有一个单独的值用来表示sql server的版本号。SQL Server 7.0的版本号是7.0，SQL Server 2000的版本号是8.0，SQL Server2005的版本号是9.0。格式文件的第二行是一个整数值，它表示格式文件中映射实际开始的行号。第3行以后的的内容都是映射信息。每个映射 行(mapping line)有8个列。前5个列代表与输入数据(本例中指文件文件)有关信息。后3个列代表要导入的目标信息(本例中指SQL表)。第一列其实就是从1开始 的系列数字。这些值总是1、2、3等，依此类推。(这些数字以及其他一些BCP格式文件里的信息看上去是没有必要的，但是在其他一些情况下确实会用到它 们。)映射行里的第二列是要导入的数据类型。当从文本文件导入数据的时候，不管这个值代表的是什么，它的类型总是SQLCHAR。第三例是前缀长度 (prefix length)。这是当进行从SQL到SQL数据拷贝时，BCP用于优化的一个相当复杂的参数。幸运的是，当把文本数据导入SQL的时候，前缀总长度值总 是0.第四列表示输入字段字符的最大长度。第五列表示字段分隔会，在此用逗号来分割所有的字段。比如说，如果输入数据文件的字段是用制表符来分割，应该在 映射文件中指定为\t。第六列到第八列指代的是目标SQL表，而不是输入文件。第六列和第七列分别指SQL表里相应列的顺序和名称。映射行第八列用来指定 要使用的SQL排序规则。

**注解**

　　使用BCP工具可以高效地以自动化的方式把测试用例数据从文本文件引入 到SQL测试用例表中。特别注意，测试用例的数据文件在数据块的最后一行之后一定不能再有换行符。如果有换行符，就会被BCP解释成一个新的空行。而用于 映射的格式文件在最后一行之后一定要有一个换行符。如果没有这个换行符，BCP就不会读入映射文件的最后一行。

　　即使文本文件的数据格式与SQL表的结果不完全吻合，也可以通过BCP工具从文本文件导入数据。换句话说，即使文本文件的数据与相应的SQL列顺序不一致或者文本文件的数据有多作的字段，仍然可以使用BCP工具来导入数据，例如：

　　0020,66,useless,e13

　　0021,77,no-need,e14

　　0022,88,go-away,e15

　　0023,99,drop-it,e16

　 　这个文件有些额外的信息并不想将其导入，而且各个字段(caseID,expected value,unneeded data,input value)的顺序也和SQL列(caseID,input,expected)的顺序不一致。针对这个文本文件的BCP文件的格式如下：

　　8.0

　　4

　　1 SQLCHAR 0 4 "," 1 caseID SQL\_Latin1\_General\_CP1\_CI\_AS

　　2 SQLCHAR 0 2 "," 3 excepted SQL\_Latin1\_General\_CP1\_CI\_AS

　　3 SQLCHAR 0 7 "," 0 junk SQL\_Latin1\_General\_CP1\_CI\_AS

　　4 SQLCHAR 0 3 "\r\n" 2 input SQL\_Latin1\_General\_CP1\_CI\_AS

　　映射文件的第6列通过设置该列的值来指定文本文件中相应数据的插入顺序，如果该列的值设为0，则在插入数据的时候忽略这个列。

　　因为bcp.exe是一个命令行程序，所以可以手动运行它也可以把想要执行的命令放到一个简单的BAT文件，然后在自己的程序里调用这个BAT文件。如果想在SQL环境下使用BCP，可以通过 BULK INSERT 命令来完成。

　　bulk insert dbTestCasesAndResults..tblTestCases

　　from 'C:\somewhere\newData.dat'

　　with (formatfile = 'C:\somewhere\newData.fmt')

**问题**

　　如何创建一个T-SQL测试套件用于测试SQL[存储](http://storage.it168.com/)过程。

**设计**

　 　首无，通过插入大量测试平台数据准备好一个包含待测存储过程的底层数据库。接下来，使用一个SQL游标(cursor)遍历这个测试用例数据表。针对每 个测试用例，调用待测存储过程并且取得它的返回值，把实际返回值与期望值进行比较，从而判定测试结果是通过与否，然后显示或保存测试结果。

**方案**

　——testAuto.sql  
  
　　——为dbEmployees填充数据  
  
　　truncate table dbEmployees.dbo.tblEmployees  
  
　　insert into dbEmployees.dbo.tblEmployees values('e11','Adams','15/10/2009')  
  
　　insert into dbEmployees.dbo.tblEmployees values('e22','Baker','15/10/2009')  
  
　　insert into dbEmployees.dbo.tblEmployees values('e33','Young','15/10/2009')  
  
　　insert into dbEmployees.dbo.tblEmployees values('e44','Zetta','15/10/2009')  
  
　　——此处插入更多数据  
  
　　declare tCursor cursor fast\_forward  
  
　　for select caseID,input,expected  
  
　　from dbTestCasesAndResults.dbo.tblTestCases  
  
　　order by caseID  
  
　　declare @caseID char(**4**),@input char(**3**),@expected int  
  
　　declare @actual int,@whenRun datetime  
  
　　declare @resultLine varchar(**50**)  
  
　　set @whenRun = getdate()  
  
　　open tCursor  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expected  
  
　　while **@@fetch\_status** = **0**  
  
　　begin  
  
　　exec @actual = dbEmployees.dbo.usp\_StatusCode @input  
  
　　if (@actual = @expected)  
  
　　begin  
  
　　set @resultLine = @caseID + ': Pass'  
  
　　print @resultLine  
  
　　end  
  
　　else  
  
　　begin  
  
　　set @resultLine = @caseID + ': FAIL'  
  
　　print @resultLine  
  
　　end  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expected  
  
　　end  
  
　　close tCursor  
  
　　deallocate tCursor  
  
　　——end script

　 　如果待测存储过程依赖于外部数据(大多数情况下都是这样的)，则必须把大量的测试平台数据填充到底层的数据库表。在SQL测试环境下，通常有两个数据 库：开发数据库供开发者在编写代码时使用，测试数据库供测试者在测试时使用。因为对存储过程进行测试经常会改变包含存储过程的这个数据库(因为存储过程经 常对数据进行插入、更新、删除)，因此不希望在开发数据襄轴上运行测试程序。因此，应该复制一份开发数据库的拷贝，并且使用这个拷贝来做测试。这样，开发 数据库的表里面保存的就是供开发者使用的数据。这些数据对于在开发待测系统(SUT)的过程中进行基本的验证测试(verification testing)是非常有必要的。但是，这些数据对于进行全面而严格的测试来说，通常是不够丰富的。

　　有好几种方法可以遍历一个存储测试 用例数据的数据库表，使用SQL游标是其他简单有效的一种。SQL游标被设计用来处理单个数据而不是像其他SQL操作(比如SELECT和INSERT) 那样处理行集(rowset)。首先应该声明一个指向保存测试用例数据的SQL表的游标：

declare tCursor cursor fast\_forward  
  
　　for select caseID,input,expected  
  
　　from dbTestCasesAndResults.dbo.tblTestCases  
  
　　order by caseID

　 　注意：与其他SQL变量不同，游标变量的名字前面并没有@字符。可供声明的游标有好几种。FAST\_FORWARD最适合用来读取测试用例数据。其他游 标类型包括FORWARD\_ONLY,READ\_ONLY,以及OPTIMISTIC。FAST\_FORWARD实际上就是FORWARD\_ONLY加上 READ\_ONLY的别名。

在使用游标之前，必须先打开游标。然后，如果想要遍历整个数据库表，则必须通过fetch next语句预读取数据库表的第一行：

open tCursor  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expected

　 　对第一行进行预读取是为了对下面的循环进行控制，使用变量@@fetch\_status来控制用于读取的这个循环，这个变量表示最近一次fetch操作 的状态。如果fetch操作成功，则@@fetch\_status值为0 。如果值为-1或-2则意味fetch失败。因此可以像下面这样每次一行地遍历整个数据库表：

　while **@@fetch\_status** = **0**  
  
　　begin  
  
　　——运行测试用例  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expected  
  
　　end

　　在主循环内部，需要调用待测[存储](http://storage.it168.com/)过程，并且把测试用例输入数据传给它。接下来取回返回值并打印pass或fail消息：

exec @actual = dbEmployees.dbo.usp\_StatusCode @input  
  
　　if (@actual = @expected)  
  
　　begin  
  
　　set @resultLine = @caseID + ': Pass'  
  
　　print @resultLine  
  
　　end  
  
　　else  
  
　　begin  
  
　　set @resultLine = @caseID + ': FAIL'  
  
　　print @resultLine  
  
　　end

　　使用完一个SQL游标之后，必须要关闭这个游标并且调用deallocate命令把它作为一个资源进行释放：

　close tCursor  
  
　　deallocate tCursor

　　如果忘了对游标进行deallocate操作，下一次执行测试套件脚本需要声明游标的时候，脚本就会报错。

**注解**

　　除了打印pass/fail信息之外，还可以把测试结果插入到SQL数据表里：

　——declare @actual int,@whenRun datetime  
  
　　——set @whenRun = getdate()  
  
　　while **@@fetch\_status** = **0**  
  
　　begin  
  
　　exec @actual = dbEmployees.dbo.usp\_StatusCode @input  
  
　　if (@actual = @expected)  
  
　　insert into dbTestCasesAndResults.dbo.tblResults values (@caseID,'Pass',@whenRun)  
  
　　else  
  
　　insert into dbTestCasesAndResults.dbo.tblResults values (@caseID,'FAIL',@whenRun)  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expected  
  
　　end

　　如果不采用通过待测[存储](http://storage.it168.com/)过程以及硬编码的语句来生成底层数据库，也可以使用BULK INSERT语句：

　——为dbEmployees填充数据  
  
　　truncate table dbEmployees.dbo.tblEmployees  
  
　　bulk insert dbEmployees.dbo.tblEmployees  
  
　　from 'C:\somewhere\richTestbedData.dat'  
  
　　with (formatfile = 'C:\somewhere\richTestbedDate.fmt')

　　使用这种方法的好处是，可以让测试套件模块化程度更高并且更为灵活，但是它也有不好的一面：它在这个原本已经有很多对象的测试套件系统里又引入了一个文件，从而增加了复杂性。

**问题**

　　如何让T-SQL测试套件把测试用例结果直接写入文本文件

**设计**

　　使用ActiveX技术实例化一个FileSystemObject对象，然后通过OpenTextFile()和WriteLine()方法直接把测试结果写入文件。

**方案**

　　请declare @fsoHandle int,@fileID int

　　exec sp\_OACreate 'Scripting.FileSystemObject',@fsoHandle out

　　exec sp\_OAMethod @fsohandle,'OpenTextFile',@fileID out,

　　'C:\pathToResults\Results.txt',8,1

　　——主测试循环

　　if(@result = @expected)

　　exec sp\_OAMethod @fileID,'WriteLine',null,'Pass'

　　else

　　exec sp\_OAMethod @fileID,'WriteLine',null,'FAIL'

　　——主测试循环结束

　　exec sp\_OADestroy @fileID

　　exec sp\_OADestroy @fsoHandle

　　我们需要有一个文件句柄和一个文件ID，它们的类型都是Int，SQL Server有一个作sp\_OACreate()的[存储](http://storage.it168.com/)过 程可以实例化ActiveX对象。sp\_OACreate()接受一个字符串作为传入的参数，这个字符串就是要创建的ActiveX对象的名字，并且以 out参数的形式返回一个已创建对象的引用，这个引用的类型是int。对于 Scripting.FileSystemObject的情况来说，返回值是关于文件句柄的一个引用。接下来，可以通过调用sp\_OAMethod()方 法打开文件。在本例中，第一个参数是sp\_OACreate()把创建的句柄，第二个参数是我们想要使用的方法的名称，第三个参数是用于保存返回的文件句 柄的变量，第四个参数指定文件的物理名称，第五个参数是可选的，它指定使用的IO模式。

　　1：以只读方式打开文件(默认)。

　　2：以写方式打开文件。

　　3：以追加方式打开文件

　　第六个参数是可选的，它是一个创建标识，用来指定当给定文件不存在的时候是否创建一个新的文件。

　　0：不创建新文件(默认)。

　　1：创建一个新文件。

　　第八个害数也是可选的，它是一个格式标识，用来指定字符的编码方式。

　　0：以ASCII方式打开文件(默认)。

　　1：以Unicode方式打开文件。

　　2：以系统默认的模式打开文件。

**注解**

　　运行T-SQL测试套件的时候，有好几种方法可以用来保存测试结果。如果把测试结果保存成文本文件，常用的技术是先把所有测试结果都保存到一个SQL表里，然后把结果转换成一个文件文件。另一种做法是，在T-SQL套件里直接把测试结果写入文本文件。

　　上述方案使用了FileSystemObject类的OpenTextFile()方法，这个方案正常工作的前提是要写入的文件已经存在。作为替代，也可以使用CreateTextFile()方法：

　　declare @fsoHandle int,@fileID int

　　exec sp\_OACreate 'Scripting.FileSystemObject',@fsoHandle out

　　exec sp\_OAMethod @fsohandle,'CreateTextFile',@fileID out,

　　'C:\pathToResults\Results.txt',0,0

　　——主测试循环

　　if(@result = @expected)

　　exec sp\_OAMethod @fileID,'WriteLine',null,'Pass'

　　else

　　exec sp\_OAMethod @fileID,'WriteLine',null,'FAIL'

　　——主测试循环结束

　　exec sp\_OADestroy @fileID

　　exec sp\_OADestroy @fsoHandle

　　CreateTextFile()方法接受一个必填的文件名参数以及两个可选的布尔值。

　 　因为SQL本身并不支持布尔数据类型，因些要使用整数类型0表示False，1表示True。第一个可选的布尔值指定覆盖标识，如果该值为True/1 则意味着覆盖已存在的同名文件。如果该值为False/0则意味着不覆盖已存在的同名文件。第二个可选的布尔参数是编码标识。其值为true/1意思是以 Unicode编码方式创建创建这个文件。其值为false/0意思是ASCII编码方式创建这个文件。这两个参数的默认值都是false/0，或者换句 话说，默认情况下都不覆盖已有文件以及使用ASCII编码方式。

　　除了从T-SQL把数据写入文本文件，还可以采用这种方法读入文本数据：

　　declare @fsoHandle int ,@fileID,int

　　declare @eof int

　　declare @line varchar(1000)

　　exec sp\_OACreate 'Scripting.FileSystemObject',@fsoHandle out

　　exec sp\_OAMethod @fsohandle,'OpenTextFile',@fileID out,

　　'C:\pathToResults\Results.txt',1,1

　　set @eof = 0

　　while @eof = 0

　　begin

　　exec sp\_OAMethod @fileID,'ReadLine',@line out

　　print @line

　　exec sp\_OAM3thod @fileID,'AtEndOfStream',@eof out

　　end

　　exec sp\_OADestroy @fileID

　　exec sp\_OADestroy @fsoHandle

**问题**

　　如何测试返回一个out参数的[存储](http://storage.it168.com/)过程。

**设计**

　　声明一个合适类型的变量，用于接受out参数，然后使用out参数调用待测存储过程。

**方案**

　　例如，设想包含待测存储的数据库有一个数据库表，这个表定义如下：

　　create table tb1Employees

　　(

　　empID char(3) primary key,

　　empLast varchar(35) not null,

　　empDOH datatime not null

　　)

　　设想待测存储过程有一个out参数，这个参数用于保存返回结果，其定义如下：

　　create procedure usp\_GetLast

　　@empID char(3),

　　@empLast varchar(35) out

　　as

　　select @empLast = empLast from tb1Employees where empID = @empID

　　return @@rowcount

　　那么用于测试这个存储过程的T-SQL代码可以是：

　　declare @input char(3)

　　declare @empLast varchar(35)

　　declare @retval int

　　declare @expectedLast varchar(35)

　　declare @expectedRet int

　　set @input = 'e22'

　　set @expectedLast = 'Baker'

　　set @expectedRet = 1

　　exec @retval = dbEmployees.dbo.usp\_GetLast @input,@empLast out

　　if(@retval = @expectedRet and @empLast = @expectedLast)

　　print 'Pass'

　　else

　　print 'FAIL'

　 　存储过程usp\_GetLast()接受一个雇员ID作为输入参数。这个存储过程取回雇员ID与输入ID相匹配的行并且把该雇员的姓氏存入一个out参 数。该存储过程返回由SELECT语句所返回的行数。可以让AUT使用这个返回值作为一种错误的检查机制：如果返回值为0，就意味着没有找到与之匹配的雇 员，但是如果返回值比1大，则说明有多于一个雇员拥有相同的ID(大多数情况下很可能出现了什么错误)。

**注解**

　　SQL存储过程有一个常用的设计模式，即存储过程可以通过out参数返回一个或多个值。当存储过程要返回的值不是Int类型的时候，这种模式是必须的，因为return关键字只接受Int类型的变量。当存储过程必须要返回多个值的时候，这种模式也是必需的。

　 　调用带out参数的存储过程就跟调用只有输入参数的存储过程一样，只不过需要在接受输出变量的参数后面要加上SQL关键字out。上述方案是根据参数的 位置进行调用的。也可以通过参数名称进行调用。但是，这种调用可能有些混乱，因为一旦使用了"@name = value"这种形式，后面的参数也必须以"@name = value"这种形式进行传递。

此部分内容创建一个轻量级T-SQL测试套件，总共有3个脚本。

　　用于创建测试平台数据和待测[存储](http://storage.it168.com/)过程的脚本：

　--======================  
  
　　--makeDbTestAndResults.sql  
  
　　use master  
  
　　go  
  
　　if exists (select \* from sysdatabases where name = 'DbTestAndResults')  
  
　　drop database makeDbTestAndResults  
  
　　go  
  
　　if exists (select \* from sysxlogins where name = 'testLogin')  
  
　　exec sp\_droplogin 'testLogin'  
  
　　go  
  
　　create database makeDbTestAndResults  
  
　　go  
  
　　use makeDbTestAndResults  
  
　　go  
  
　　create table tb1TestCases  
  
　　(  
  
　　caseID char(**4**) primary key,  
  
　　input datetime not null,  
  
　　expectedChecksum int not null  
  
　　)  
  
　　go  
  
　　--下面插入的是用于usp\_HiredAfter的测试数据，采用了对期望值求校验和的方法  
  
　　--也可以通过BCP,BTS或者C#程序从文本文件读入这些数据  
  
　　insert into tb1TestCases values('0001','10/25/2009',**1042032**)  
  
　　insert into tb1TestCases values('0002','10/25/2009',**9999999**)--deliberate error  
  
　　insert into tb1TestCases values('0003','10/25/2008',**25527856**)  
  
　　insert into tb1TestCases values('0004','10/10/2006',**1042032**)  
  
　　go  
  
　　create table tb1Results  
  
　　(  
  
　　caseID char(**4**) not null,  
  
　　result char(**4**) not null,  
  
　　whenRun datetime not null  
  
　　)  
  
　　go  
  
　　exec sp\_addlogin 'testLogin','secret'  
  
　　go  
  
　　exec sp\_grantdbaccess 'testLogin'  
  
　　go  
  
　　grant select,insert,delete,update on tb1TestCases to testLogin  
  
　　go  
  
　　grant select,insert,delete,update on tb1Results to testLogin  
  
　　go  
  
　　--结束脚本

用于创建测试例例数据和测试结果[存储](http://storage.it168.com/)的脚本：

　--==========================  
  
　　--makeDbEmployees.sql  
  
　　use master  
  
　　go  
  
　　if exists(select \* from sysdatabases where name = 'dbEmployees')  
  
　　drop database dbEmployees  
  
　　go  
  
　　if exists(select \* from sysxlogins where name = 'employeesLogin')  
  
　　exec sp\_droplogin 'employeesLogin'  
  
　　go  
  
　　create database dbEmployees  
  
　　go  
  
　　use dbEmployees  
  
　　go  
  
　　create table tb1Employees  
  
　　(  
  
　　empID char(**3**) primary key,  
  
　　empLast varchar(**35**) not null,  
  
　　empDOH datetime not null,  
  
　　)  
  
　　go  
  
　　insert into tb1Employees values('e11','Adams','10/25/2009')  
  
　　insert into tb1Employees values('e22','Baker','10/25/2009')  
  
　　go  
  
　　exec sp\_addlogin 'employeesLogin','secret'  
  
　　go  
  
　　exec sp\_grantdbaccess 'employeesLogin'  
  
　　go  
  
　　create procedure usp\_HiredAfter  
  
　　@dt datetime  
  
　　as  
  
　　select \* from tb1Employees where empDOH > @dt  
  
　　go  
  
　　grant execute on usp\_HiredAfter to employeesLogin  
  
　　go  
  
　　--end script

自动化测试套件脚本：

--=================================  
  
　　--SQLspTest.sql  
  
　　--测试dbEmployees..usp\_HiredAfter  
  
　　--读入测试用例数据并把测试结果写回  
  
　　set nocount on  
  
　　if not exists(select \* from master.dbo.sysdatabases where name='DbTestCasesAndResults')  
  
　　raiserror('Fatal error:dbTestCasesAndResults not found',**16**,**1**)  
  
　　go  
  
　　if exists(select \* from sysobjects where name='tap\_Reset')  
  
　　drop procedure tap\_Reset  
  
　　go  
  
　　create procedure tap\_Reset  
  
　　as  
  
　　truncate table dbEmployees.dbo.tb1Employees  
  
　　insert into dbEmployees.dbo.tb1Employees values('e11','Adams','10/25/2009')  
  
　　insert into dbEmployees.dbo.tb1Employees values('e22','Baker','10/12/2009')  
  
　　insert into dbEmployees.dbo.tb1Employees values('e33','Young','06/15/2001')  
  
　　insert into dbEmployees.dbo.tb1Employees values('e44','Zetta','08/15/2003')  
  
　　--其他数据在此处插入  
  
　　exec tap\_Reset  
  
　　go  
  
　　declare tCursor cursor fast\_forward  
  
　　for select caseID,input,expectedChecksum  
  
　　from DbTestCasesAndResults.dbo.tb1Testcases  
  
　　order by caseID  
  
　　declare @caseID char(**4**),@input datetime,@expectedChecksum int  
  
　　declare @whenRun datetime  
  
　　declare @resultMsg varchar(**80**)  
  
　　declare @actualChecksum int  
  
　　create table #resultRowset --for checksum technique  
  
　　(  
  
　　empID char(**3**) primary key,  
  
　　empLast varchar(**35**) not null,  
  
　　empDOH datetime not null  
  
　　)  
  
　　set @whenRun = getdate()  
  
　　print 'stored procedure under test = usp\_HiredAfter'  
  
　　print ' '  
  
　　print 'CaseID Input Expected Actual Result'  
  
　　print '==========================================='  
  
　　open tCursor  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expectedChecksum  
  
　　while **@@fetch\_status**=**0**  
  
　　begin  
  
　　exec tap\_Reset --reset test bed data  
  
　　truncate table #resultRowset --empty out the result rowset  
  
　　insert #resultRowset(empID,empLast,empDOH) --call sp under test  
  
　　exec dbEmployees.dbo.usp\_HiredAfter @input  
  
　　if(**@@rowcount**=**0**)  
  
　　set @actualChecksum=**0**  
  
　　else  
  
　　select @actualChecksum=checksum\_agg(binary\_checksum(\*)) from #resultRowset  
  
　　if (@actualChecksum=@expectedChecksum)  
  
　　begin  
  
　　set @resultMsg=@caseID + ' ' + cast(@input as varchar(**11**)) +  
  
　　' ' + cast(@expectedChecksum as varchar(**20**)) + ' ' +  
  
　　cast(@actualChecksum as varchar(**20**)) + 'Pass'  
  
　　print @resultMsg  
  
　　insert into dbTestCasesAndResults.dbo.tb1Results values(@caseID,'Pass',@whenRun)  
  
　　end  
  
　　else  
  
　　begin  
  
　　set @resultMsg=@caseID + ' ' + cast(@input as varchar(**11**)) +  
  
　　' ' + cast(@expectedChecksum as varchar(**20**)) + ' ' +  
  
　　cast(@actualChecksum as varchar(**20**)) + 'FAIL'  
  
　　print @resultMsg  
  
　　insert into dbTestCasesAndResults.dbo.tb1Results values(@caseID,'FAIL',@whenRun)  
  
　　end  
  
　　fetch next  
  
　　from tCursor  
  
　　into @caseID,@input,@expectedChecksum  
  
　　end  
  
　　close tCursor  
  
　　deallocate tCursor  
  
　　drop table #resultRowset