# 基于ADO. NET 的数据库访问及其性能优化

华国栋, 刘文予
(华中科技大学 电子与信息工程系, 湖北 武汉 430074)

**摘　要**: 在构建和开发基于数据库的应用时,如何提高应用程序对数据库的访问效率和消除性能瓶颈是一个关键问题。通过剖析 ADO. NET 访问数据库的机制,分析了在开发基于 ADO. NET 的数据库应用时影响数据库访问性能的一些主要原因,并提出了改善性能的方法和思路。

**关键词**: ADO. NET; 数据库; 性能优化; SQL; C#
**中图法分类号**: TP311. 13　　**文献标识码**: A　　**文章编号**: 1001-3695( 2004) 06-0215-04

# Performance Optimizing in Database Accessing Based on ADO. NET

HUA Guo-dong, LIU Wen-yu
( Dept. of Electronics & Information, Huazhong University of Science & Technology, Wuhan Hubei 430074, China)

Abstract: How to enhance the database accessing efficiency and eliminate performance bottlenecks is always a key problem when we construct and develop applications based on database. This article analyses the mechanism which use ADO. NET to access the database and main reasons which affect the performance in database accessing, and proposes some methods to improve the performance.

Key words: ADO. NET; Database; Performance Optimizing; SQL; C#

## 1　引言

ADO. NET 是 Microsoft . NET 应用程序的数据访问模型。它由 ADO 技术发展而成,在某种程度上, ADO. NET 代表了最新版本的 ADO 技术。由于 ADO. NET 建构于. NET 框架之内,它的建立和管理都是基于 CLR( Common Language Runtime, 通用语言运行时) ,所以直接或间接地得益于. NET 框架在内存管理、类型转换、对象池等方面的技术改善和优化。ADO. NET 引入了一些重大变化和革新,它对 XML 提供全面的支持,提供了新的非连接数据缓冲模型,使其在构建结构松散的、非链接的 Web 应用程序上有着得天独厚的优势[1,2]。而基于数据库的 Web 应用程序一个核心功能就是为用户提供可靠和高效的数据库访问,对数据库访问性能的优劣是评价一个 Web 应用程序好坏的关键标准。一般而言,影响数据库访问性能的主要因素有三个: 数据库服务器的硬件性能、网络性能以及数据库访问程序的设计。目前,前两项的性能普遍得到提高和增强,所以数据库访问的程序设计将直接影响到应用程序的最终性能。

## 2　ADO. NET 的访问机制分析

### 2. 1　ADO. NET 概述

ADO. NET( 图 1) 有以下两个核心组件:

( 1) ADO. NET DataSet。它是 ADO. NET 的非连接结构的核心。DataSet 的设计目的是为了实现独立于任何数据源的数据访问,在本地内存中实现一个数据缓存。它的数据源并不仅仅局限于数据库,也可以从 XML 文件或自定义的本地数据文件中获得数据。DataSet 包含一个或多个 DataTable 对象的集合,这些对象由数据行和数据列以及主键、外键、约束和有关 DataTable 对象中数据的关系信息组成。

( 2) . NET 数据供应器( Data Provider) 。它是一组包括 Connection, Command, DataReader 和 DataAdapter 对象在内的组件。它实现了对数据库的连接、操作和快速、只读的访问。Connection 对象提供与数据源的连接; Command 对象用来执行数据库命令; DataReader 从数据源中获得高性能的数据流; DataAdapter 对象是连接数据源和 DataSet 对象的桥梁。DataAdapter 使用 Command 对象在数据源中执行 SQL 命令和调用存储过程,以便将数据加载到 DataSet 中,并保持 DataSet 中数据的更改与数据源中的数据一致。System. Data 包含了独立于供应器的类型,如 DataSet 及 DataTable。

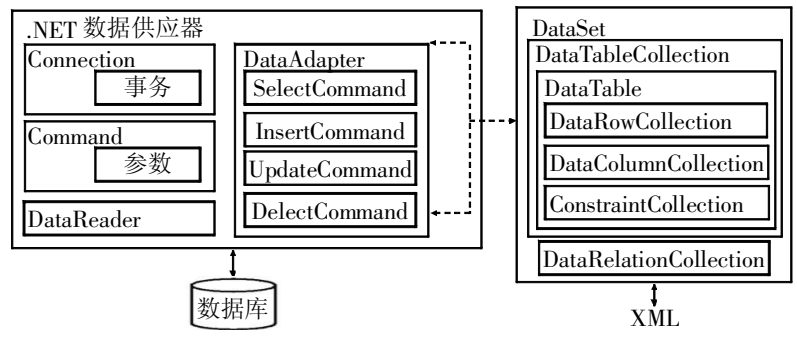


图 1　ADO. NET 结构示意图

### 2. 2　ADO. NET 的两种访问数据库的模式

对于不同的应用需要, ADO. NET 设计了两种访问数据库的模式: 使用 DataReader 对象进行基于连接的访问和使用 DataAdapter 对象进行的非连接的访问。由于这两种模式的设计目的不同,所以在应用程序开发中应该根据实际需要选择合

收稿日期: 2003-03-28; 修返日期: 2003-06-30

适的数据库访问模式以发挥出 ADO. NET 的最佳性能,这对于提升应用程序性能有着非常重要的意义。下面对两种访问模式进行分析和比较:

(1) DataSet 对象。它能够提供读取数据的本地缓存,而且由于 DataSet 对象对 XML 良好的支持,使它特别适合于在应用的各层之间或通过 Web Service 以 XML 方式进行传递,另外它还能满足当所需数据来自于多个数据源(如多个数据库或文件),而又必须建立相互之间关联关系的需要。DataSet 对象同时也提供对获得数据的批量更新功能。但构建 DataSet 对象时需要较大的额外开销,使它在数据访问效率上较 DataReader 对象稍逊。值得注意的是,由于每个 DataSet 对象都会占据一定量的内存,如果设计不当,会造成 DataSet 对象的大量生成,从而耗尽内存资源,严重降低性能。

(2) DataReader 对象。它通过为数据存取提供一个连续、只读的指针(Cursor)能够提供高效的、仅向前的流式结果输出,所以使用它能进行高速的数据读取,尤其是在对二进制大对象(BLOB)的数据读取上这种优势尤为明显,但与 DataSet 相比则缺少了许多灵活性。

由于一般的 Web 应用只是需要从数据库中高效地读取数据,然后经过简单处理或者直接将其显示到用户页面或进行 XML 输出。在这种情况下,应该优先使用 DataReader 对象,而对于需要进行 XML 交换、动态关联和交互的应用则应该选择 DataSet 对象。

## 3 使用ADO.NET 访问数据库的性能优化

### 3.1 选择合适的.NET 数据供应器

ADO.NET 的数据库访问基础是.NET 数据供应器(Data Provider)。其中最常用的是 OLE DB 和 SQL Server 两种,前者主要用于连接支持 OLE DB 接口的数据库,如 Oracle, SQL Server 6.5, Access 等,而后者则是为连接 SQL Server 7.0 以上数据库定制。由于它使用 SQL Server 本身的 TDS(Tabular Data Stream)数据交换协议与数据库交互,不必通过 OLE DB 层的协议转换,因而效率很高,较之 OLE DB 数据供应器,它的性能可高出 30% ~40%。对于以 Microsoft SQL Server7.0 以上版本为后端数据库的应用程序,应该首选该数据供应器以获得最佳性能。

### 3.2 数据库连接池使用与合理配置

数据库连接池是目前改善程序数据访问性能的重要手段,它的基本原理是预先建立若干数据库连接对象放入特定内存结构(称为池)中,每次收到用户请求时便从池中取出一个连接对象交给用户使用,使用结束后并不真的关闭数据库连接,而是将它重新返回池中,如此就可以循环使用这些连接对象,避免反复建立和关闭连接的系统开销。由于数据库连接是极其重要和有限的资源,而反复地建立和释放连接造成了大量的开销和性能降低,产生性能瓶颈,采用数据库连接池技术可以有效地消除这个瓶颈,极大地改善数据库访问性能。

在 ADO.NET 中的连接池由.NET 数据供应器提供,各种供应器实现连接池的确切机制不尽相同。本文以 SQL Server.NET 数据供应器为例介绍 ADO.NET 连接池的使用和配置。对于 SQL Server .NET 数据供应器,建立连接时是自动使用数据库连接池的,但为了获得最佳性能,有必要对连接池的参数进行定制,通过在数据库连接串中指定相关属性对连接池实现设定,如连接串:

```
string connStr =
"server = (local) \\AppDB; Integrated Security = SSPI; Database = ApplicationMax Pool Size = 50; Min Pool Size = 5 ";
```

设定连接池的初始连接数为 5 最大连接数为 50。

相关属性的具体功能如下所示:

- Pooling 赋值 True 或 False 指定是否使用连接池,缺省值为 True(使用连接池);
- Min Pool Size 赋值不小于 0 的整数指定池内维持连接对象的最小数量,缺省值为 0;
- Max Pool Size 赋值不小于 0 的整数指定池内维持连接对象的最大数量,缺省值为 100;
- Enlist 赋值 True 或 False 指定当前连接是否自动加入到当前调用线程的事务中,缺省值为 True;
- Connection Reset 赋值 True 或 False 指定当连接从池中移除时是否重置,缺省值为 True;
- Connection Lifetime 赋值不小于 0 的整数以指定连接的生命期(秒),每次连接返回池中时,都会将当前时间与建立时间相比较,如果差值大于所设生命期,则关闭连接对象;指定 0 则表示生命期无限,缺省值为 0。

其中的最大连接数限定了各进程所能争抢的连接总数。当用户请求的连接总数超过这个值时,就会进入排队等待状态,直到有连接空出。这样实际上也限制了能并发访问的用户数量,此值设置不当,会造成性能瓶颈。实际开发中可以通过 Windows 平台的性能监视器或数据库自身的性能监视工具观查不同设置下的访问性能以选择最佳值。另外需要格外注意的是,使用连接池时,ADO.NET 要求每次连接的连接串必须完全相同,否则将导致当前的连接池无效,而建立新的连接池,从而造成不必要的开销。如下面两个连接串:

```
string connStr = "server = (DBServer)
\\AppDB; Integrated Security = SSPI; Database = Application";
string connStr = "server = (DBServer)
\\AppDB; Integrated Security = SSPI ; Database = Application";
```

后者仅比前者多出一个空格,却仍然会造成连接池失效而建立新的连接池。为避免这种情况,可考虑将连接串写入配置文件中,使用时读取。

### 3.3 SQL 命令的优化

与数据库交互的基本语言是 SQL,数据库每次解析和执行 SQL 语句需要进行很多步骤。以 SQL Server 为例,当数据库收到一条查询语句时,语法分析器扫描 SQL 语句并将其分成逻辑单元(如关键词、表达式、运算符和标志符)生成查询树,最后查询优化器分析所有可以访问源表的方法,从中选择一组返回结果最快且消耗资源较少的步骤。查询树随即进行更新以准确记录这个步骤,然后交由数据库引擎开始执行,然后将查询结果返回用户[3,4]。可见数据库引擎每次执行 SQL 命令都会有很大的开销,如果提交的 SQL 质量不高甚至有逻辑错误就会造成无谓的开销和时间浪费。为避免这种情况,在使用 SQL 命令时应注意以下原则:

(1) 在编写 SQL 查询之前了解表的索引结构。有效的利用索引能够避免不必要的全表扫描,缩短查询时间。应该避

免在 WHERE 子句中使用 "IS NULL", "< >", "! =", "! >", "! <", "NOT", "NOT EXISTS", "NOT IN", "NOT LIKE 等命令, 它们通常会导致全表扫描而导致索引无效。

(2) 在编写 SQL 命令时应该尽量缩小查询范围并限制返回的列数。避免使用不带 WHERE 子句的查询, WHERE 子句能够有效地缩小查询范围, 提高查询效率, 而不带 WHERE 子句的查询往往会导致查询时间过长, 系统开销增大, 导致性能降低。除非你能确知返回结果的数量很少, 否则一定要用 WHERE 子句对查询范围加以限定。避免使用 select * from 查询语句, 这会返回指定表的所有列, 而实际真正需要的往往只是少数的几列, 如果表的列数很多或增加, 势必会造成返回结果数据量太大, 增加系统开销和网络负载。

(3) 尽量不要在查询中使用数据库服务器的排序功能 (Order by), 这样会降低查询的性能, 可以将查询结果读至本地数据集(DataSet) 中进行排序操作, 那样通常会更快。

(4) 善用数据库的存储过程。与 SQL 命令不同, 它不需要每次都重新分析和优化, 而是一次性编译成执行计划缓存在数据库中, 以后可以直接调用, 避免了重复的解析过程, 节省了时间。特别是当查询任务由一系列 SQL 命令组成时, 应该考虑将其编写成存储过程, 这样能够避免数据在网络上来回传送, 降低网络流量。例如, 在笔者开发的系统中需要经常地新增和更新用户数据库, 在新增用户时, 必须保证用户名的唯一性, 一般的操作是先进行查询判断有无重复, 然后再进行插入, 如果将这个步骤编写成存储过程, 则每次只需要执行一次命令。

(5) 在 WHERE 子句中, 任何对列的操作(函数、计算等) 都将导致索引失效, 这些操作应该尽可能地移至等号右边, 如:

WHERE SUBSTRING(id, 1, 1) = 'A' 应写成 WHERE id LIKE 'A%'

WHERE result* 10 > 3 应写成 where result > 30

对 SQL 命令进行优化的基本原则是尽量减少类型转换和计算, 充分利用表索引, 减少全表扫描的次数。SQL 命令的优化是一个十分复杂的工作。以上列出的优化方法主要是在应用开发层面的。在实际开发中对于一些使用频率很高的 SQL 命令应该根据具体的情况, 提出不同的几种方案, 试验选出实际性能最好的 SQL 语句。

### 3.4　程序设计中的优化

(1) 由于在整个数据库访问过程中, 数据库连接对象都是非常有限和宝贵的资源, 所以在使用数据库连接时, 不论是否采用了连接池, 都应该尽可能晚地打开连接, 尽可能早地关闭连接。关闭连接时一定要关闭其间建立的所有临时对象, 并结束所有用户定义的事务, 这样才能保证连接的正常关闭, 避免系统资源浪费。当使用 DataReader 对象时, 应该仅在执行 SQL 命令前打开连接, 读取结果后即关闭连接, 否则数据库连接会一直被占用, 不能用于其他目的, 应该尽早调用 DataReader 对象的 Close( ) 方法关闭连接。如以下代码片断(C#) 从数据库中读取 10 条记录进行一定的处理:

```
dbConn. Open( ) ; //打开连接
SqlDataReader dbReader = SQLCmd. ExecuteReader( ) ;
//执行 SQL 命令, 返回结果
string[ ] testStr = new string[ 10] ;
int i = 0;
while (dbReader. Read( ) ) //读取记录
{
testStr[ i ++ ] = dbReader. GetString( 0) ) ;
}
dbReader. Close( ) ;
dbConn. Close( ) ; //关闭数据库连接
for( i = 0; i < testStr. Length; i ++ )
{
testStr[ i] = process( testStr[ i] ) ;
//调用方法 process( ) 对结果进行处理
}
```

其中的 Process( ) 方法如果在 While 循环中调用会造成数据库连接更长时间的占用, 所以对于返回结果较多的情况, 应该先将数据读入本地缓存, 在关闭数据库连接后再进行结果处理。

虽然 DataSet 对象是一个非连接的本地数据缓存, 在它的生命期内并不会一直占用数据库连接, 它通过 DataAdapter 对象读取数据, 调用 Fill( ) 方法时自动打开和关闭数据库连接。但 Fill 方法并不会明确地关闭数据库连接, 它只会关闭由自己打开的连接, 如果数据库连接是在调用 Fill 方法之前打开的, 则会保持连接的开放而不是将它关闭, 所以在这种情况下, 应该在程序中显式地关闭数据库连接, 避免资源浪费。

(2) 使用 DataReader 对象读取查询结果时, 应该优先通过列的序号而不是列的名称来读取, 这样能够避免列名到列序号的转换额外开销。在明确知道查询列的数据类型的情况下, 应该使用类型化的存取器方法(如 GetInt32, GetString), 避免读取列数据所需的类型转换以提高性能。不同方法的读取性能比较如下所示:

```
// 按列名称读取, 速度慢
string value = dbReader[ "value"] . ToString( ) ;
// 按列序号读取, 速度快
string value = dbReader[ 0] . ToString( ) ;
// 按列序号并指定读取数据类型, 速度最快
string value = dbReader. GetInt32( 0) . ToString( ) ;
```

(3) 使用 DataReader 对象访问数据时, 如果碰到查询时间过长或返回数据太多的情况, 这时需要终止查询以避免将不必要的数据从服务器发送到客户端。为了快速有效地结束查询, 应该在调用 DataReader 对象的 Close 方法之前调用 Command 对象的 Cancel 方法, 这是因为 DataReader 对象的 Close 方法将填写输出参数的值、返回值和 RecordsAffected 属性, 因此增加了关闭查询的时间, 所以在这之前应该先调用 Command 对象的 Cancel 方法以确保查询结果被抛弃, 而不会被发送到客户端。

(4) Command 对象有四种执行 SQL 命令的方法: ① ExecuteReader 主要用来读取查询结果数据; ② ExecuteNonQuery 主要用来执行 Insert, Delete, Update 等命令和存储过程; ③ ExecuteScalar 则主要用来执行返回单个值的命令; ④ ExecuteXmlReader 用来从 SQLServer 数据库中获得 XML 格式的数据输出。

由于 ExecuteReader 方法会创建 DataReader 对象, 需要较大的系统开销, 所以当从数据库中检索单个值, 如执行 count(*), MAX( ), AVG( ), SUM( ) 等命令时应该优先使用 ExecuteScalar 方法。以下代码(C#) 查询用户总数:

```
SqlCommand sqlCmd =
new SqlCommand( "select count( * ) from users", dbConn) ;
dbConn. Open( ) ; //打开连接
SqlDataReader dbReader = sqlCmd. ExecuteReader( ) ;
//执行 SQL 命令, 返回结果
int count = sqlCmd. ExecuteScalar( ) ; //获得单一值
```

```
dbConn. Close( ) ; //关闭连接
```

如果检索结果超过一个值但数量很少,可以使用 ExecuteNonQuery 方法,然后通过 Command 对象的输出参数获得结果,如以下代码( C#) 利用输出参数获得对应列的最大值和平均值:

```
SqlCommand sqlCmd =
new SqlCommand( "select @ pmA = AVG( price) , @ pmB = MAX( price) from Products", dbConn) ;
SqlParameter paramA =
sqlCmd. Parameters. Add( "@ pmA", SqlDbType. Money) ;
paramA. Direction = ParameterDirection. Output;
//指定参数为输出类型
SqlParameter paramB =
sqlCmd. Parameters. Add( "@ pmB", SqlDbType. Money) ;
paramB. Direction = ParameterDirection. Output;
//指定参数为输出类型
dbConn. Open( ) ;
sqlCmd. ExecuteNonQuery( ) ;
dbConn. Close( ) ;
Console. WriteLine( ( decimal) paramA. Value) ;
//显示或处理返回结果
Console. WriteLine( ( decimal) paramB. Value) ;
```

以上两种方法避免了 DataReader 对象的创建,以较小的开销获得了所需数据。

( 5) 如果需要反复地执行一条 SQL 语句( 如插入和更新操作) ,为提高效率可以利用 Command 对象的 Prepare( ) 方法创建命令的一个准备版本以避免对 SQL 语句的重复分析,加快执行速度,以下代码( C#) 以 Prepare 方法预处理 SQL 插入命令,然后填写不同参数反复执行。这种方法在执行大量的数据插入和更新命令时,效率很高。

```
dbConn. Open( ) ;
SqlCommand sqlCmd = new SqlCommand( null, dbConn) ;
sqlCmd. CommandText = "insert into user( username, password)
values ( @ username, @ password) ";
sqlCmd. Parameters. Add ( "@ username", userinfo[ 0] [ 0] ) ;
sqlCmd. Parameters. Add ( "@ password", userinfo[ 0] [ 1] ) ;
sqlCmd. Prepare( ) ;
sqlCmd. ExecuteNonQuery( ) ;
for( int i = 1; i < users. Length; i + + )
{
sqlCmd. Parameters[ 0] . Value = userinfo[ i] [ 0] ;
sqlCmd. Parameters[ 1] . Value = userinfo[ i] [ 1] ;
sqlCmd. ExecuteNonQuery( ) ;
}
dbConn. Close( ) ;
```

( 6) 在创建数据库连接时,在连接串中使用数据库服务器的 IP 地址,而不是服务器名称,这样可以避免域名解析,加快建立速度,如以下代码( C#) :

```
string connStr =
"server = ( DBServer) \\AppDB;
IntegratedSecurity = SSPI; Database = Application";
string connStr =
"server = ( 192. 168. 21. 12) \\AppDB;
IntegratedSecurity = SSPI; Database = Application";
```

避免了将服务器名称 DBServer 映射成 IP 地址 192. 168. 0. 3 的解析过程。

综观以上几种方法,在程序设计中优化和改善数据库访问性能的基本原则是:尽量减少资源的开销,尽量缩短系统资源特别是数据库连接的占用时间,尽量避免不必要的数据映射和转换。

## 4 结束语

ADO. NET 是当前开发基于数据库的应用系统中的重要技术,如何将这种技术成功地应用到自己的应用系统中,发挥它最大的优势,减少系统瓶颈,提高应用性能,是一个复杂而艰巨的任务,需要对系统的各个环节都有深入的了解和把握。本文在初步分析 ADO. NET 数据库访问机制的前提下,分析了一些系统瓶颈产生的原因,提出了一些改善性能的思路和方法,在笔者实际开发基于. NET 框架的 Web 应用系统中得到了应用并取得了良好的效果。

**参考文献:**


[ 1] 王毅. ASP. NET 1. 0 高级编程[ M] . 北京: 清华大学出版社, 2002.

[ 2] 康博. C# 高级编程[ M] . 北京: 清华大学出版社, 2002.

[ 3] 邱仲潘. SQL Server 2000 实用全书[ M] . 北京: 电子工业出版社, 2002.

[ 4] 胡江奕. 基于 SQL Server 的数据库应用系统性能的优化[ J] . 计算机工程与应用, 2001, 37( 2) : 95-97.

[ 5] icrosoft. NET Data Access Architecture Guide[ DB/OL] . http: //msdn. microsoft. com/library/default. asp? url = /library/en-us/dnbda/html/daag. asp, 2001-10.


**作者简介:**


华国栋( 1975-) , 男, 湖北武汉人, 在读硕士研究生, 主要研究方向是通信与信息系统; 刘文予( 1963-) , 男, 湖南株洲人, 教授, 博士生导师, 博士, 主要研究方向为计算机图形学、计算机视觉、多媒体信息处理。


( 上接第 149 页)

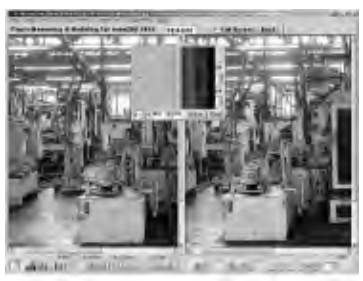

图 7 测量软件用立体像对

图 8 数字化三维重建

## 7 实验结果

基于此方案开发的双目立体测量系统, 结合 AutoCAD 2000, 具备了更高的使用效率, 并已投入使用。它不仅在零件的三维重建上发挥了巨大的功能, 在诸如工厂数字化的大型工程中, 改进后的双目测量软件, 其优越性也充分地显现出来。

**参考文献:**


[ 1] 贾云得. 机器视觉[ M] . 北京: 科学出版社, 2000. 160-161.

[ 2] 王新华, 李宇光. 图像分析与测量技术[ M] . 武汉: 武汉测绘科技大学出版社, 1998.

[ 3] 陈杉, 王宁, 郭建峰. 用 ObjectARX 开发 AutoCAD 2000 应用程序[ M] . 北京: 人民邮电出版社, 2000.

[ 4] 王福军, 张志民, 张师伟. AutoCAD 2000 环境下 C/Vsiual C + + 应用程序开发教程[ M] . 北京: 北京希望电子出版社, 2000.


**作者简介:**


宋丽华( 1977-) , 女, 硕士研究生, 主要研究方向为图像检测与机器视觉; 仲思东( 1962-) , 男, 副院长, 教授, 主要研究方向为机器视觉及精密检测技术。