

基于 ASP.NET 缓存与分页策略优化 Web 数据查询性能

董一华

(杭州师范学院信息工程学院, 浙江 杭州 310012)

摘要: 利用 ASP.NET 的缓存功能, 结合性能良好的存储过程, 一次性地缓存数据库数页数据, 以供用户分页查询, 是优化 Web 数据查询性能的比较理想的方式。文章详细介绍了优化模型的实现, 并进行了性能分析。

关键词: ASP.NET; Web 数据查询; 存储过程; 缓存; 分页

0 引言

随着 Web 应用的深入, 如何提高 Web 数据查询性能, 并使用户在合理的时间内搜索到自己需要的信息, 已经成为 Web 发展的关键问题之一。

数据库分页查询, 是解决这一问题提高 Web 数据查询效率的一个重要手段。但许多基于 ASP.NET 的网站在处理分页时(如图 1 所示), 往往只重视数据库服务器本身的分页过程, 而没有合理利用数据库服务器返回到 Web 服务器的分页记录集。因为对于多页查询结果, 用户往往更倾向于浏览邻近页面的内容, 很少会跳过许多页面去查看数据。

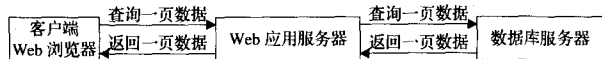


图 1 传统分页模型

因此, 将 ASP.NET 缓存技术与数据库存储过程相结合, 从数据库一次性地提取适量页面的数据, 将之缓存于 Web 应用服务器, 使用户浏览相邻页面数据时, 只需提取 Web 应用服务器缓存的数据, 而无需一次一次地连接数据库服务器, 减少了数据库查询的次数, 从而优化 Web 数据库查询性能。

1 基于 ASP.NET 的分页优化模型

如图 2 所示, 用户通过 Web 浏览器向 Web 应用服务器请求一页数据, 应用服务器从数据库中提取一个虚拟页, ASP.NET 再将虚拟页细分成实际页, 将之缓存, 并根据用户请求, 将缓存的一个实际页绑定到 ASP.NET 列表控件(如 DataGrid, DataList)或 PagedDataSource 类, 最终向用户返回所请求的一页数据。为了尽可能减少服务器间的数据流量, 减少数据库服务器的压力, 可以预先设置虚拟页的大小 P_v 和实际页的大小 P_t , Web 应用服务器则根据内存情况以及利用存储过程得到的查询记录总数, 实时调整 P_v 的大小, 使 $P_v \geq P_t$, 如果数据不多, 则可以使 $P_v = P_t$, 即不使用虚拟页。

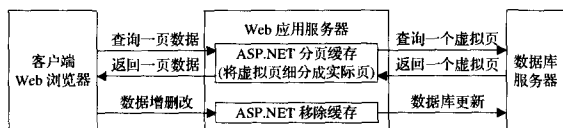


图 2 基于 ASP.NET 的分页优化模型

由于使用了 ASP.NET 缓存技术, 对于已经缓存在 Web 应用服务器上的数据, 就无需再次访问数据库服务器, 用户可以

直接从 Web 应用服务器获得分页信息, 大大节省了访问时间; 对于没有缓存或缓存失效的数据, 就需要向数据库服务器读取分页信息; 一旦数据记录被更新, ASP.NET 会根据缓存失效机制移除缓存, 以便保持数据的一致性。

2 ASP.NET 的缓存技术

ASP.NET 1.1 主要使用两种缓存技术: 输出缓存和应用程序数据缓存。

2.1 输出缓存(Output Cache)

主要是缓存 ASP.NET 整页(.aspx)或页中的用户控件(.ascx), 缓存的格式是页面生成后的 HTML 代码。可以利用 @OutputCache 指令, 或一组 API(HttpCachePolicy, HttpResponse.Cache, Page.Response, PartialCachingAttribute 等)进行缓存。

以 @OutputCache 指令为例, 将 @OutputCache 指令加入到 ASP.NET 页(.aspx)或用户控件(.ascx), 能实现整页或页中部分代码的输出缓存。具体用法如下:

```
<%@ OutputCache
    Duration="#ofseconds" [缓存过期时间]
    Location="Any|Client|Downstream|Server|None|ServerAndClient"
    Shared="True|False" [用户控件的输出缓存是否可以由多个页
        共享(只用于用户控件)]
    VaryByParam="parametername" [缓存随参数值变化]
    ..... %>
```

其中, Location 的属性值决定了缓存位置, 但只适用于.aspx。而 VaryByParam 属性, 主要与 URL 中的查询字符串配合使用, 比如 http://server/abc.aspx?page=1, 设置 VaryByParam="page", 此时, 就可以根据用户查看过的页码缓存相应的页面。这样就减轻了数据库服务器的压力, 同时, 由于输出缓存是已经生成的 HTML, 无需再通过 Web 应用服务器生成 HTML, 这样也减轻了 Web 应用服务器的压力, 优化了 Web 数据查询的性能。

2.2 应用程序数据缓存(Cache 类)

利用 Cache 类, 可以缓存原始的数据对象(如 DataTable, DataSet), 为此, 可以使用简单的 Cache[key]=DataObject, 将数据对象添加到 Cache 集中, 也可以通过比较复杂的方法 Cache.Add()或 Cache.Insert(), 设置缓存的依赖项、过期策略、优先级等属性。可以通过 Cache.Remove()使缓存显式地移除。

由于 Cache 的范围是全局性的, 所以当 A 用户已经缓存了 N 页数据时, B 用户就可以直接使用这 N 页数据, 而不需要再

次访问数据库。

2.3 两种缓存技术对比

缓存技术	缓存内容	缓存位置	范围	可控性
输出缓存	整页或部分页的 HTML 代码	客户端 / 服务器端	全局性 / 特定用户	较弱 (与后台程序交互比较困难)
应用程序缓存	数据对象 (如 DataTable)	服务器端	全局性	较强 (Cache 对象可以非常容易地与后台代码交互数据)

3 存储过程分页

为了实现分页,必须定义一些分页变量。存储过程代码如下所示^[2]:

```
CREATE PROCEDURE 存储过程名
@CurrentPage INT,      -- 当前页
@PageSize INT,         -- 虚拟页大小
@TotalRecords INT OUTPUT, -- 总记录数(返回参数)
.....
AS
BEGIN
    DECLARE @startRow INT, -- 页内开始行
    @endRow INT, -- 页内结束行
    /* 初始化变量 */
    /* 此处插入分页方案 */
END
```

以下是几种典型的存储过程分页方案的伪代码^[1-5]。

3.1 临时表

```
CREATE TABLE #Temp
( ID int IDENTITY PRIMARY KEY, PK 数据类型 - PK 是
  Table 的主键字段,以下同..... )
INSERT INTO #Temp SELECT ...
FROM Table ORDER BY SortColumn
SELECT ... FROM #Temp ORDER BY ID WHERE ID
BETWEEN @startRow AND @endRow
DROP TABLE #Temp
```

3.2 子查询

3.2.1 子查询(SELECT TOP + NOT IN)

```
SELECT TOP @PageSize ... FROM Table
WHERE ( PK NOT IN
( SELECT TOP (@CurrentPage-1)*@PageSize PK
  FROM Table ORDER BY @SortCols)
) ORDER BY SortColumn
```

3.2.2 子查询(SELECT TOP + MAX())

如果 SortColumn 只有一个字段,且类型为数字或字符时,可使用以下子查询:

```
SELECT TOP @PageSize ... FROM Table
WHERE (SortColumn >=(SELECT MAX(SortColumn)
  FROM ( SELECT TOP (@CurrentPage-1)*@PageSize
    SortColumn
  FROM Table ORDER BY SortColumn ) AS Temp)
) ORDER BY SortColumn
```

3.2.3 子查询(SELECT TOP+ 表变量)

```
DECLARE @Temp TABLE
```

```
( PK 数据类型 NOT NULL PRIMARY KEY )
```

```
INSERT INTO @Temp
```

```
SELECT TOP @PageSize ...
```

```
FROM( SELECT TOP @CurrentPage*@PageSize
/* 或者 TOP (@startRow + @PageSize) */
... FROM Table ORDER BY SortColumn
```

```
) AS Temp
```

```
ORDER BY SortColumn
```

```
SELECT PK FROM @Temp ORDER BY SortColumn
```

3.2.4 子查询(SELECT TOP+TOP)

```
SELECT TOP @PageSize ...
FROM(SELECT TOP(@PageSize+@startRow) ...
FROM Table ORDER BY SortColumn) AS Temp
ORDER BY SortColumn
```

3.3 RowCount

```
DECLARE @Sort 数据类型
```

```
-- 设置存放 SortColumn 字段值的变量
```

```
SET ROWCOUNT @startRow -- 限定查询的行数
```

```
SELECT @Sort = SortColumn FROM Table ORDER BY
SortColumn
```

```
SET ROWCOUNT @PageSize
```

```
-- 限定查询从 @StartRow 行到(@StartRow + @PageSize)行
```

```
SELECT ... FROM Table WHERE SortColumn >= @Sort
ORDER BY SortColumn
```

```
SET ROWCOUNT 0
```

3.4 游标

```
DECLARE @PK 数据类型
```

```
DECLARE @Temp TABLE ( PK 数据类型 NOT NULL
PRIMARY KEY)
```

```
DECLARE PagingCursor CURSOR DYNAMIC READ_ONLY
FOR -- 定义游标
```

```
SELECT @CurDataCols FROM Table ORDER BY SortColumn
OPEN PagingCursor
```

```
FETCH RELATIVE @StartRow FROM PagingCursor INTO
@PK
```

```
WHILE @PageSize > 0 AND @@FETCH_STATUS = 0
BEGIN
```

```
INSERT @Temp ( PK ) VALUES( @PK )
```

```
FETCH NEXT FROM PagingCursor INTO @PK
```

```
SET @PageSize = @PageSize - 1
```

```
END
```

```
CLOSE PagingCursor
```

```
DEALLOCATE PagingCursor
```

```
SELECT @PK FROM @Temp ORDER BY SortColumn
```

3.5 各种方案性能测试及取舍

以上方案适用不同的场合。Jasmin Muharemovic、李洪根等已经对有关方案作过测试,其中,RowCount 分页方案性能相对较佳^[1-5]。

4 ASP.NET 分页

利用 ASP.NET 列表控件(如 DataGrid)或 PagedDataSource

分页类实现 ASP.NET 分页。

以 DataGrid 为例,它可实现两种分页功能:默认分页和自定义分页。由于用户只需要获得一页数据,而默认分页要求取出全部数据,效率不高,因此必须将 AllowCustomPaging 属性设为 True,进行自定义分页,同时将 VirtualItemCount 属性设为从存储过程返回的总记录数,将 PageSize 属性设为实际页的大小,然后将一个实际页数据绑定到 DataGrid。

5 基于 ASP.NET 的分页优化模型的性能分析

设两个模型(1 为传统模型,2 为优化模型)的每次连接数据库时间均为 T_c ,存储过程执行时间分别为 T_{p1} 和 T_{p2} ,数据集从数据库服务器返回 Web 服务器的延时分别为 T_{w1} 和 T_{w2} ,优化模型将虚拟页细分为实际页的时间可以忽略不计;客户端向 Web 应用服务器请求的延时为 T_s ,Web 应用服务器向客户端返回数据的延时为 T_r ;Web 应用服务器内部处理客户端提交的时间和处理返回结果的时间分别为 T_a 和 T_b ;虚拟页大小 P_v 和实际页大小 P_r 。

假设有 X 条记录,用户需要每次返回 P_r 条作为一页内容,如果用户共查询了 N 次,则

传统模型需要用时 $T_1=(T_s+T_a+T_c+T_{p1}+T_{w1}+T_b+T_r) \times N$ 。

而对于优化模型的查询,设虚拟页大小 $P_v=\delta \times P_r$ (δ 为正整数 P_v 且是 N 的整数倍);直接从虚拟页(从缓存而不经过程序)获取数据的概率是 ρ ,则优化模型查询用时 $T_2=T_{2a}+T_{2b}$,其中:

(1) 经过数据库的查询:

$$T_{2a}=(T_s+T_a+T_c+T_{p2}+T_{w2}+T_b+T_r) \times N \times (1-\rho)。$$

(2) 直接取自缓存,不经过程序的查询: $T_{2b}=(T_s+t_r) \times N \times \rho$ 。

(由于数据已经缓存,故 T_a 和 T_b 可忽略不计。)

虚拟页一般不会超过实际页大小的 10 倍,即 $0 < \delta < 10$,故有:

$$T_w=(T_{w1} \approx T_{w2}), T_p=(T_{p1} \approx T_{p2})。$$

两模型的用时差为: $T=T_1-T_2=(T_a+T_c+T_p+T_w+T_b) \times N \times \rho$ 。

由此可见,与传统分页模型相比,使用缓存的分页优化模型在性能上有明显的优势。

6 结束语

利用 ASP.NET 的分页缓存功能和存储过程,能使 Web 数据查询性能得到优化。然而,由于缓存占用了 Web 应用服务器的内存,当 ρ 非常小且 P_v 比较大时,Web 应用服务器的内存会很紧张。当然,正如前面所述,可以通过优化缓存的一些属性,如依赖项、过期策略、优先级等,对缓存进行有序的管理。

参考文献:

- [1] Jasmin Muharemovic, Paging of Large Resultsets in ASP.NET [DB/OL], <http://www.codeproject.com/aspnet/PagingLarge.asp>
- [2] 李洪根,SQL Server 存储过程的分页方案 [DB/OL], <http://blog.csdn.net/lihonggen0/archive/2004/09/14/103511.aspx>
- [3] Alon Catz, Paging and Sorting on SQL Server and Oracle using Sql.Net [DB/OL],<http://www.codeproject.com/cs/database/PagingSqlNet.asp>
- [4] Andrew Rosca, Server-Side Paging with SQL Server [DB/OL], <http://www.winnetmag.com/Article/ArticleID/40505/40505.htm>
- [5] Andrey Shchurov, Using RowCount for sorting when paging large result sets [DB/OL], http://www.codeproject.com/aspnet/ben_paging.asp



2007

计算机 时代

刊号: CN 33-1094/TP ISSN 1006-8228

《计算机时代》月刊,1983年创刊,是《中国期刊网》、《中国学术期刊(光盘版)》和《中国核心期刊(遴选)数据库》收录期刊。主要栏目有:技术广角、网络天地、信息安全、应用实践、经验技巧、考试园地、计算机教育、企业信息化等。丰富的内容使您能够自由地遨游在信息的海洋里,了解最新技术,掌握最新技术,享受精彩纷呈的数字生活。

欢迎投稿 欢迎订阅 欢迎刊登广告

邮发代号: 32-81 月价: 5元 全年定价: 60元

地址: 杭州市环城西路33号省计算所大楼
E-mail: Computer_era@21cn.com

邮编: 310006
电话: (0571)87054111