ASP.NET 2.0 (Whidbey)

Mi várható a 2005. évi ASP.NET-ben?

IV. rész

Az ASP.NET Cache és az SQL Server 2005 együttműködése

Bevezetés

Az előző részben megnéztük az ASP.NET Cache alapvető használatát, illetve adatbázisfüggő Cache-elés felépítését SQL Server 2000 háttérrel. Ebben a részben megvizsgáljuk, milyen új Cache vezérlési lehetőségeket aknázhatunk ki az SQL Server 2005 nyújtotta új szolgáltatások felhasználásával.

Adatbázisfüggő Cache felépítés alapjai

Emlékeztetőül a probléma leírása. Adatbázisból lekérdezések eredményhalmazát jelenítjük meg weblapokon. A lekérdezések erőforrásigényesek, ezért szeretnénk minimalizálni őket. A lekérdezések adatait tárolhatjuk a futó weblapok kódjával azonos AppDomainben, az ASP.NET Cache objektumban. Azonban az adatbázisadatok véletlenszerű időzítéssel változhatnak, amely után a gyorsítótárba helyezett adatokat ki kell üríteni, illetve a következő kérésnél újra kell építeni. A fő probléma: hogyan értesüljön róla a tár, hogy a lekérdezett adatok változtak? Kinek is van ismerete elsőkézből a változásokról? Természetesen az adatbázisnak. Ha az adatbázis vissza tudna szólni a Cache-nek, hogy az adatok változtak, akkor a problémát megoldottuk.

Ehhez két követelményt kell teljesíteni:

1. Az adatbázis tudjon róla, hogy egy bizonyos lekérdezés, - amely akár WHERE feltételt, JOIN-t, stb. is tartalmaz – kimenete megváltozott egy adatmódosító művelet hatására. Azaz az adatbázisnak meg kell jegyezni a kitüntetett lekérdezéseket, és figyelni az adatmódosításokat, melyek érintenék a korábbi adatokat. Ez nem egyszerű probléma.
2. Az adatbázis képes legyen aszinkron módon értesíteni az ASP.NET Cache-t a megváltozott adatokról. Az aszinkron követelmény miatt kiesett a „triggerrel közvetlenül beszólok a Cache-nek” megoldás, mert a szinkron Cache kezelés nagyon lelassítaná az adatbázismódosításokat.

A második követelmény miatt szükséges valamiféle átmeneti tároló, ahol várakoznak a Cache értesítések feldolgozásra. Ha igazságos rendszert akarunk létrehozni, aki elsőnek jött, azt kell elsőként kiszolgálni, azaz egy várakozási sorra, egy Queue-ra lesz szükség. Ezt a sort már lehetne triggerrel táplálni, ám triggerrel még mindig nehéz lenne az 1. követelményt implementálni. Szerencsére az SQL Server 2005-ben mindkét követelményre van megoldás. A lekérdezés eredményhalmazának változását a Query Notifications nevű technológia képes figyelni, az események aszinkron továbbítását egy várakozási soron keresztül pedig a Service Broker nevű új SQL Server komponens képes elvégezni.

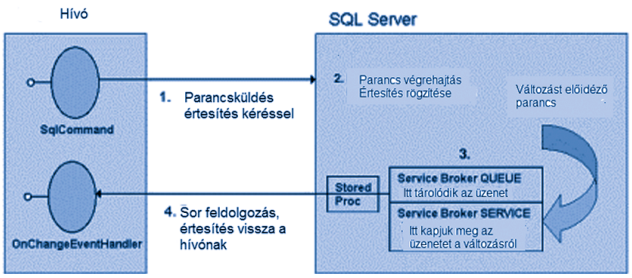
Bár a cikk az ASP.NET Cache-ről szól, nagy vonalakban érdemes megismerni ezen két szereplőt, így jobban megértjük az SqlCacheDependency működését is.

SQL Server 2005 Query Notifications

A szolgáltatás építőkövei a következők:

- SQL Server 2005 Query Engine
- SQL Server Service Broker
- sp_DispatcherProc (CLR) tárolt eljárás
- SqlNotification osztály (System.Data.Sql.SqlNotificationRequest)
- SqlDependency osztály (System.Data.SqlClient.SqlDependency)

Az ADO.NET SqlCommand kapott egy új jellemzőt a 2.0-ban, a Notification-t. Ha a parancs futtatása előtt ezt kitöltjük egy SqlNotificationRequest példánnyal, akkor ezzel jelezzük az SQL Servernek, hogy kérünk értesítést, ha a futtatott lekérdezés eredményhalmaza megváltozik. Az SQL Server ezek után „figyel” minden adatmódosítást, hátha az érinti a korábban regisztrált értesítendők egyikét. Ha változás történt, akkor a Query Engine elküld egy üzenetet a Service Broker várakozási sorába. Ezek után az értesítés vagy visszamegy közvetlenül a hívóhoz, vagy a hívó kérdezi le van-e a számára értesítés. A folyamatot szemlélteti a következő ábra:



Az SQL Server 2005 Query Notifications működése

A következő kérdések az izgalmasak:

1. Honnan tudja az SQL Server, hogy egy lekérdezés eredményhalmaza megváltozott?
2. Hogyan szól vissza a hívónak?

Az eredményhalmaz változásfigyelés valójában már benne volt az SQL Server 2000-ben is. Amikor ugyanis indexelt né-

zetet hoztunk létre, akkor az SQL Servernek tudnia kellett átvezetni a nézet alapját adó táblák változásait a letárolt nézetre. Mivel ugyanez a mechanizmus működteti az értesítéseket, ezért ugyanazok a kötöttségek vonatkoznak a felhasznált lekérdezésekre. Pl. UNIONt, DISTICTet, OUTER JOINt, allekérdezést, és még jó pár rázósabb parancsot nem lehetett használni indexelt nézetekben, és ezekre az értesítések se mennek. Miért? Mert nagyon nehéz megmondani, hogy egy tábla változás hatására változik-e egy, a fenti bűnös parancsokat használó lekérdezés eredményhalmaza. Eme háttérismeret azért nagyon fontos, mert az indexelt nézetek dokumentációja alapján máris tudhatjuk, hogy a Query Notifications fog-e működni a konkrét lekérdezésünkre. Ami „elég egyszerű”, arra menni fog.

Nézzük a második kérdést. Mi történik, ha a szerver észreveszi, hogy értesítést kell küldenie? A Query Engine a Service Broker egy SERVICE-éhez küldi az értesítést.

A Service Broker egy új elem az SQL Server 2005-ben, a 2000-nek még nem volt része. Ő tulajdonképpen egy adatbázisban megvalósított várakozási sor, amelyben SERVICE-ek létesítenek kommunikációs végpontokat. A SERVICE egy olyan végpont, amely adattárolását egy QUEUE végzi el, és egy CONTRACT (gyakran XML Schema) írja le a SERVICE-ben feldolgozható üzenetek formátumát. Azaz egy QUEUE-ban lehet több SERVICE, és minden SERVICE-hez csatolhatunk CONTRACT-okat.

A Query Notifications CONTRACT-ja (szerződése, formátumleírása) be van építve az SQL Serverbe, és a következő URL azonosítja:

```
http://schemas.microsoft.com/SQL/Notifications/
ƒPostQueryNotification
```

Ez csak egy azonosító, nem tölt be semmit az SQL Server a fenti URL-ről. Kifinomult esetben létrehozhatunk saját SERVICE-t, saját QUEUE-ban is, de az alap értesítések kedvéért már léteznek ezek az msdb adatbázisban. Miután az értesítések elkezdenek szaporodni az alapértelmezett QUEUE-ban, ki lehet őket olvasni (pollozással). De azt ígértem, hogy az SQL Server képes visszaszólni a hívónak! És tényleg. A már említett sp_DispatcherProc hozzá van rendelve az alapértelmezett Query Notification SERVICE-hez, az dolgozza fel, ha értesítések esnek be. Ő egy .NET-ben írt tárolt eljárás, amely felolvassa a várakozási sorba érkezett értesítéseket, és saját protokoll segítségével visszaszól az ügyfélnek. A saját protokoll lehet TCP és HTTP, de ez nem azonos az SQL parancsok végrehajtására használt, TDS csomagokat szállító csatornával, különösen, hogy ez a kiszolgáló felől nyílik meg az ügyfél irányába. Mivel ez egy független csatorna, az értesítések kedvéért nem kell nyitva tartani a normál adatbázis-kapcsolatot, az értesítések „hátulról”, független szálakról érkeznek be a hívóba. Amikor az ügyfél megkapta az értesítést, az sp_DispatcherProc törli a QUEUE-ból az értesítési kérést, és indítja a következő feldolgozást.

Az SQL Server 2005 Query Notifications programozása – SqlDependency

Habár belülről igen összetett az értesítő architektúra, programozni meglepően egyszerű. Az SqlDependency osztály egységbe zár minden részletet, ami az értesítések kezeléséhez szükséges. Csak annyi a teendőnk, hogy hozzárendeljünk egy SqlDependency példányt a futtatandó SqlCommandhoz, és rákapcsolódjuk az OnChanged eseményére. Ügyelni kell arra, hogy a parancs olyan legyen, amiről az SQL Server 2005 képes értesítést küldeni, ezt kitárgyaltuk az előző fejezetben. A példánk egyszerű SELECT lesz, de ebben is kéttagú táblaneveket kell használni, hogy a tulajdonos (SQL Server 2005-ben schema) egyértelmű legyen. E nélkül nem kapnánk értesítést.

Lássunk hát egy működő vázat:

```
using System;
using System.Data;
using System.Data.SqlClient;

class App {
  static void Main() {
    string connString = "Data Source=.;Initial
↳ Catalog=AdventureWorks;Integrated
↳ Security=true;";
    using (SqlConnection conn =
      new SqlConnection(connString))
    // 2-tagú táblanevek kellenek,
    //mint az indexelt nézetekben
    using (SqlCommand cmd =
      new SqlCommand(
      "SELECT Description, StartDate FROM
↳ Sales.SpecialOffer", conn)) {
      try {
        //Az értesítést kezelő objektum
        // hozzárendelése a parancshoz
        SqlDependency depend =
          new SqlDependency(cmd);
        //Az eseménykezelő regisztrációja
        depend.OnChanged +=
          new  OnChangedEventHandler(OnChanged);

        conn.Open();
        SqlDataReader rdr = cmd.ExecuteReader();
        //Eredményhalmaz feldolgozása
        while (rdr.Read())
          Console.WriteLine(rdr[0]);
        rdr.Close();

        //Várunk az értesítésekre
        Console.WriteLine("ENTER");
        Console.ReadLine();
      }
      catch (Exception e) {
        Console.WriteLine(e.Message);
      }
    }
  }

  static void OnChanged(object caller,
    SqlNotificationEventArgs e) {
    Console.WriteLine("A lekérdezés
↳ eredményhalmaza megváltozott.");
    Console.WriteLine("Source " + e.Source);
    Console.WriteLine("Type " + e.Type);
    Console.WriteLine("Info " + e.Info);
  }
}
```

Az SqlDependency konstruktorban lehet átviteli csatornát választani, hitelesítést kérni, stb. Ezeket most nem tárgyalom részletesen, főleg, mert ezek még változhatnak a végleges változatig.

Kapcsolat az ASP.NET Cache-sel

Ha már ilyen szépen kidolgozták az adatváltozásokról szóló értesítéseket, nem volt túl nehéz alkalmazniuk azt ASP.NET környezetre is. Az ASP.NET Cache eleve rendelkezik függőségkezeléssel, így külső esemény vagy változás hatására képes kiütni valamilyen tárolt tartalmat. Ami új a 2,0-ban, hogy kapunk adatbázis-függőségi kezelést is, az SqlCacheDependency személyében.

Ez az osztály kétféleképpen tud működni: az előző részben bemutatott SQL 7 és 2000-re épített pollozós módon, ezt láttuk, hasznos, de nem túl izgalmas.

SQL Server 2005 esetén már sokkal izgibb a játék, hisz a Query Notification nem bután a tábla változásaira reagál, hanem szelektíven egy adott lekérdezésre hangoltan működik. Nézzünk egy példát az SqlCacheDependency kézi használatára, az előző részben látott mintára építve:

```
public partial class SqlCacheDep :
  System.Web.UI.Page
{
  protected void Page_Load(object sender,
    EventArgs e)
  {
    DataSet cachedData = (DataSet)Cache["d"];
    if (cachedData == null)
    {
      SqlCacheDependency dependency;
      cachedData =
        LoadFromDatabase(out dependency);
      dg.DataSource = cachedData;
      Cache.Insert("d", cachedData, dependency);
    }
    else
    {
      dg.DataSource = cachedData;
    }
    dg.DataBind();
  }

  private DataSet LoadFromDatabase(
    out SqlCacheDependency dependency)
  {
    using (SqlConnection conn =
      new SqlConnection("..."))
    {
      SqlDataAdapter adapter =
        new SqlDataAdapter(
        "SELECT Description, StartDate FROM
↳ Sales.SpecialOffer", conn);
      dependency = new
        SqlCacheDependency(adapter.SelectCommand);
      DataSet ds = new DataSet();
      adapter.Fill(ds, "SpecialOffer");
      return ds;
    }
  }
}
```

A lap azonban hibával száll el:

```
System.Data.SqlClient.SqlException: User "NT
AUTHORITY\NETWORK SERVICE" does not have
permission to request query notification
subscriptions on database "AdventureWorks".
A severe error occurred on the current command.
The results, if any, should be discarded.
```

Ez aztán az irgum-burgum! Viszont logikusan jött ez a hiba, mert az értesítések adatbázis erőforrásokat kötnek le, ezért védeni kell őket. Adjunk hát jogot a weblapunkat futtató felhasználónak:

```
GRANT SUBSCRIBE QUERY NOTIFICATIONS TO
[NT AUTHORITY\NETWORK SERVICE]
```

Erre kapunk egy másik hibát:

```
Cannot find the object
"QueryNotificationErrorsQueue" because it does not
exist or you do not have permissions
```

A QUEUE-hoz RECEIVE jogra van szükségünk, hogy ki tudjuk venni belőle az értesítést:

```
GRANT RECEIVE
ON dbo.[QueryNotificationErrorsQueue]
TO [NT AUTHORITY\NETWORK SERVICE]
```

Persze sokkal egyszerűbb lett volna, ha beraktam volna a webfelhasználómat a sysadmin szerepkörbe, és akkor minden simán menne. Amint láttuk, némi pluszmunkával ugyan, de gyenge felhasználóval, minimális jogosultságokkal is lehet sikereket elérni, ráadásul a rendszerünk sokkal védettebb lesz a támadásokkal szemben.

Az OutputCache együttműködése az SQL Server 2005-tel

Az OutputCache-t nagyon szeretjük, mert egyszerű használni, és az egész lap html kimenetét letárolja, ezért általában sokkal hatékonyabb, mint az előbbi DataSet alapú gyorsítás. Ha szeretnénk az OutputCache-t függővé tenni az SQL 2005 Query Notifications-től, akkor ezt a fejlécet kell berakni a lap elejére:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="OutputCacheDep.aspx.cs"
Inherits="OutputCacheDemo" %>
<%@OutputCache SqlDependency="CommandNotification"
Duration="86400" VaryByParam="none"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
...
  <body>
    <form id="form1" runat="server">
        <asp:Label id="time" runat="server"/><p/>
        <asp:DataGrid id="dg"
         AutoGenerateColumns="true"
         runat="server"/>
    </form>
  </body>
</html>
```

Az SqlDependency="CommandNotification" jelöli ki, hogy nem pollozásra, hanem valódi értesítésre vágyunk.

A lapot mozgató kód rendkívül egyszerű, simán kiolvassuk az adatokat DataAdapter vagy DataReader segítségével:

```
public partial class OutputCacheDemo:
  System.Web.UI.Page
{
  protected void Page_Load(...)
  {
    time.Text = DateTime.Now.ToString();
    dg.DataSource = LoadFromDatabase();
    dg.DataBind();
  }

  private DataSet LoadFromDatabase()
  {
    using (SqlConnection conn =
      new SqlConnection("..."))
    {
      SqlDataAdapter adapter =
        new SqlDataAdapter(
        "SELECT Description, StartDate FROM
↳ Sales.SpecialOffer", conn);
      DataSet ds = new DataSet();
      adapter.Fill(ds, "SpecialOffer");
      return ds;
    }
  }
}
```

Az ASP.NET automatikusan hozzákapcsolja az SqlCacheDependency-t a végrehajtott parancshoz, és az majd mindent elintéz a háttérben!
A lap teszteléséhez egyszer le kell futtatni az aspx oldalt, utána egy napig nem fog változni a lap tetején látható idő, hacsak meg nem változtatjuk az adatokat, pl. így:

```
UPDATE Sales.SpecialOffer
SET Description = Description + '1'
WHERE SpecialOfferID = 1
```

Játsszunk kicsit az értesítő architektúrával!

```
UPDATE Sales.SpecialOffer
SET Description = Description + '1'
WHERE 0 = 1
```

Triviális trükk lett volna, természetesen nem kapunk értesítést.

```
UPDATE Sales.SpecialOffer
SET Description = Description
WHERE SpecialOfferID = 1
```

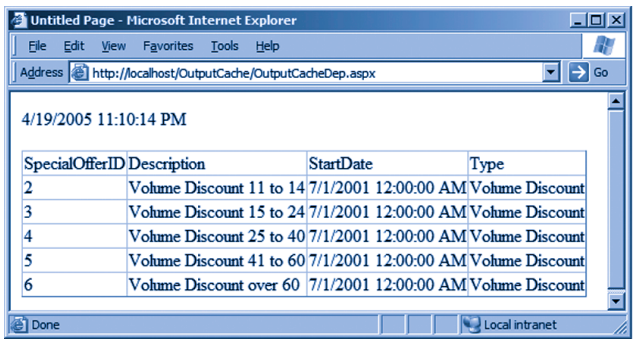
Ennek bedől, és üríti az OutputCache-t. Tanulság: ne tessék kuka, valójában semmit nem csináló UPDATE-eket írni. Végül is a triggerek is lefutnak az ilyen parancsra, a notification is buzgón jelez. Mert ő ilyen. Inkább szól kisebb megmozdulásokra is, semmint a Cache tartalma valami elavult vacakot tartalmazzon.

```
ALTER TABLE Sales.SpecialOffer
ADD UjOszlop INT NULL
```

Ez ugyan nem módosít adatokat a táblában, de azért ez elég nagy volumenű módosítás, hogy kapjunk róla hírt.
Kísérletezzünk az értesítések szelektivitásával is! Írjuk át a lekérdezést így:

```
SELECT SpecialOfferID, Description, StartDate,
[Type] FROM Sales.SpecialOffer
WHERE [Type] = N'Volume Discount'
```

Kimenet:



Lőjünk mellé egy UPDATE-tel:

```
UPDATE Sales.SpecialOffer
SET Description = Description
WHERE SpecialOfferID = 1
```

Mivel ez nem érinti az előbb leválogatott sorokat, a Cache háborítatlan marad! Ez nagyon ügyes, ezt már nem tudta az SQL Server 2000-re épülő megoldás.

Soczó Zsolt
zsolt.soczo@netacademia.net
A szerző a NetAcademia vezető fejlesztőoktatója
ASP.NET MVP, MCSE, MCSD, MCDBA, MCT

A cikkben szereplő URL-ek:
[1] netacademia.net/tudastar/articlepage.aspx?upid= 5876