Distribuované programování na platformě Microsoft .NET

Aleš Keprt 21.11.2005, revize 30.11.2006

Dotnet umožňuje realizovat distribuované aplikace třemi způsoby:

1. .NET Remoting
2. XML Web Services
3. ASP.NET

Každá z těchto věcí se hodí na jiný typ úloh.

Navíc je samozřejmě možné přímo pracovat se sítí, tj. posílat si zprávy pomocí socketů přes TCP/IP protokol, případně pomocí FTP, HTTP nebo třeba i UDP. Tyto starobylé varianty zde diskutovat nebudeme, jelikož se jedná o

i. běžnou věc nesouvisející s dotnetem
ii. blbost z hlediska praktického (proč to dělat takto složitě, když máte tamty první tři mnohem lepší možnosti?)

.NET Remoting

Distribuované aplikace v obecné formě můžete realizovat pomocí Remotingu. Uděláte s ním jakýkoliv typ aplikací, neboť Remoting je úplně obecný. Je to náhrada za systém DCOM, který se používal pro stejný typ úloh před příchodem dotnetu. V určitých rysech se Remoting a DCOM shodují, Remoting je ale novější a tedy propracovanější ➔ podstatně jednodušší k použití. Díky bohu! DCOM totiž uměl skoro totéž, ale byl pro programátory naprosto nelidsky složitý. Remoting je co do složitosti v pohodě. ☺

Co to umí?

Remoting slouží k tomu, abyste váš program spouštěli na více počítačích. Program rozdělíte do „komponent“ a ty pak můžete fyzicky umístit na různé počítače. Remoting nabízí opravdu mnoho možností, jak přesně to udělat – záleží, co pro vaši konkrétní aplikaci potřebujete.

Principiálně je to zcela triviální: Rozhodnete se, kolik komponent budete mít. Určíte, která třída vašeho programu bude patřit do které komponenty. Vytvoříte si ve Visual Studiu příslušné projekty a zkompilujete je. Umístíte je na počítače, kde mají běžet a je to. ☺

Příklad 1: Šachy s umělou inteligencí

Budeme mít tři komponenty:

i. Herní server – ten bude hlídat pravidla atp.
ii. Hráč s umělou inteligencí A
iii. Hráč s umělou inteligencí B

Aby hra byla spravedlivá, je dobré umístit každou komponentu na jiný počítač a můžete hrát.

Příklad 2: Chat

Budeme mít dvě komponenty:

i. Chatovací server
ii. Chatovací klient

Server spustíme na nějakém počítači a klienta si nahrajeme tolikrát, kolik lidí bude chatovat. Na libovolné počítače, třeba i víckrát na stejný počítač. A můžeme chatovat.

Oba příklady ukazují velmi jednoduché aplikace. Mohli byste je naprogramovat pomocí TCP/IP ručně, ale to by bylo

i. Více pracné
ii. Neobjektové, čili méně přehledné, méně flexibilní, více náchylné na chyby, atd.

Co to tedy umí?

Umí to prakticky všechno, co na síti jde udělat. Nevýhodou pak je, že musíte přesně nastavit, co tedy po Remotingu chcete, takže je to samozřejmě složitější než používat třeba XML Web services (bude popsáno níže).

- Objekty mohou být stateful i stateless
- Životnost objektů je na bázi stárnutí (zestárneš → umřeš)

Typy tříd

Rozlišujeme tři typy tříd:

1. Serializable = mohou sloužit pro přenos dat
2. Remotable = mohou být vykonávány vzdáleně
3. Ordinary = běžné třídy

Remotable třída nebo její instance mohou sloužit jako vstupní bod do programu na vzdáleném počítači (obecně v jiné aplikační doméně). Instance serializable třídy mohou sloužit jako vstupní parametry či návratové hodnoty metod volaných vzdáleně. Ordinary třídy jsou ty ostatní (běžné třídy, čili na síti k ničemu ☺).

Všechny základní typy v dotnetu jsou serializable:

Simple value types	Boolean, Byte, Char, DateTime, Decimal, Double, Enum, Guid, Int16, Int32, Int64, Sbyte, Single, TimeSpan, UInt16, UInt32, and UInt64
Core classes	Array, EventArgs, Exception, Random, String, and almost all collection classes
Data classes	DataSet, DataRow, DataRelation, DataTable, and the native SQL Server–specific types

Serializable třídy

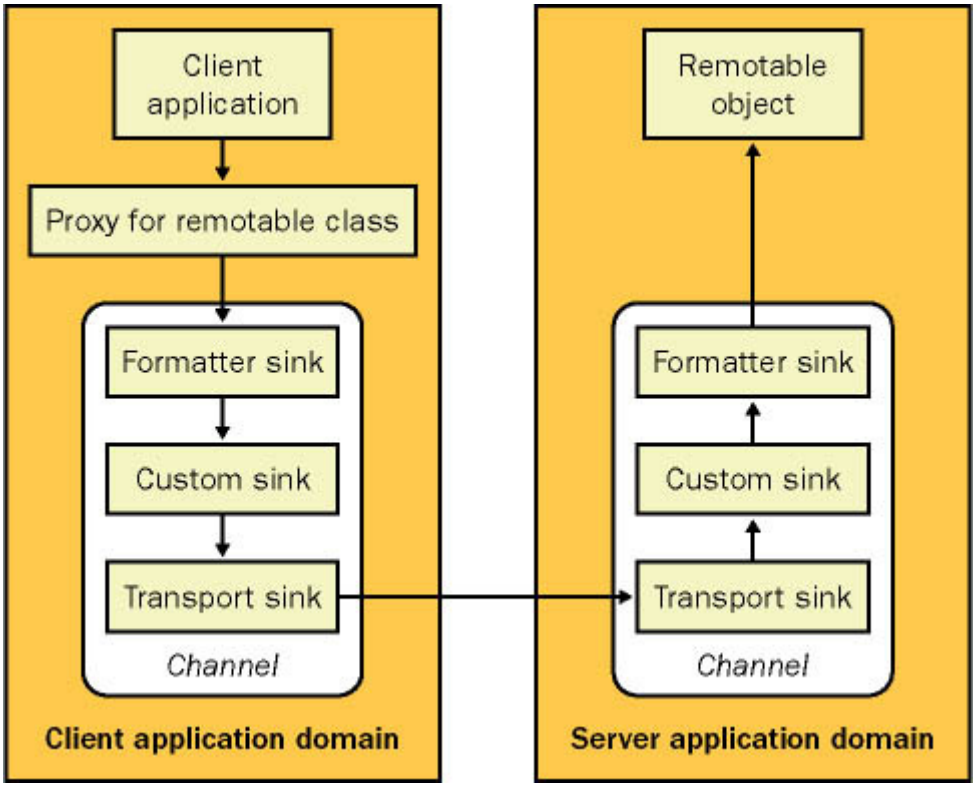
Třída má atribut [Serializable()].
Složky třídy, které nechceme serializovat, mají atribut [NonSerialized()].
U základních datových tříd stačí automatická serializace, u složitějších věcí ale musíte dávat bacha: Každá referencovaná proměnná ze serializable třídy se serializuje taky. V nejhorším případě to skončí kompletní serializací všech datových položek všech existujících objektů ve vašem programu – přenášet to celé po síti je šílenost.
Detaily viz jakákoliv podrobná kniha o programování v dotnetu.

Remotable třídy

Každá veřejná složka třídy dědící z MarshalByRefObject může být použita vzdáleně.

```
public class RemoteObject : MarshalByRefObject ...
```

Funguje to pomocí proxy: Na klientovi je proxy objekt, který se zvenku tváří stejně jako ta skutečná instance vaší třídy. Když něco z toho použijete, kód uvnitř proxy zajistí zavolání toho skutečného objektu.



Na obrázku: Skutečná realizace používá „channel sink“, což je řada objektů zajišťující formátování a fyzické poslání zprávy na vzdálený počítač.

- Formatter sink – naformátuje objekty do přenosového formátu
- Custom sink – zde můžete doplnit vlasní zpracování (třeba šifrování) dat
- Transport sink – zajišťuje skutečný přenos (obvykle po síti přes IP protokol)

Tento mechanizmus je na obou stranách – klient i remote objekt to mají.

Component Host

Kromě Remotable třídy je třeba mít také dedikovaný server – je to EXE, ve kterém je kód, který při spuštění zaregistruje vaši remotable třídu na daném počítači (pod nějakým identifikátorem, podle kterého ho pak klient najde).
Klient tu třídu pak identifikuje URI adresou s těmito složkami:

- Identifikátor protokolu (viz výše – transport sink)
- Identifikátor cílového bodu komunikace (tj. IP adresa počítače, číslo portu apod.)
- Identifikátor hostu (jméno, podle kterého se najde běžící component host)
- Identifikátor třídy (jméno, pod kterým je registrovaná třída)

Všechna ta jména se dají nastavit – nemusí být pořád stejná. Čili můžete zkompilovat jeden program a pak při běhu to teprve pojmenovávat – hodí se, když chcete mít víc serverů najednou na jednom počítači atp.

Aktivační režimy

Remoting je opravdu obecný. Projevuje se to i nabídkou aktivačních režimů. Aktivační režim = „jak se to jako zavolá“. Jsou tři možnosti:

1. SingleCall = remote objekt je stateless a vytváří se při každém volání metody
2. Client-activated = remote objekt se chová stejně jako lokální objekty, tj. vytváří se pomocí konstrukce new a je stateful
3. Singleton = remote objekt funguje jako singleton, tj. při prvním zavolání je vytvořena instance a ta je pak už pořád používána při všech voláních

Obecně se doporučuje používat SingleCall režim, pokud je to možné. Client-activated režim je provozně náročný, ale má samozřejmě mnoho výhod pro programátora, protože jako jediný je stateful. Singleton je de facto stejný jako SingleCall – i když si pamatuje své datové položky mezi voláními, pamatuje si to „dohromady" pro všechny instance. (Je to prostě singleton – to přece znáte.)

Poznámka: Je úplně jedno, co z těch tří režimů použijete. Stejně by to měl být jen vstupní komunikační bod vaši komponenty. Všechno důležité dáte do jiných tříd – nezapomeňte, že i u SingleCall režimu můžete mít ve vaší komponentě uchován stav – máte tam přece dedikovaný component host a v něm může být cokoliv.

Kdo aktivuje objekty?

V režimech SingleCall a Singleton jde o „server–activated" objekty, tj. vytváří je server. Ten třetí případ je jasný podle názvu ☺.

Formátovače ☺ (viz formatter sink)

Dotnet podporuje SOAP a binary. Další si můžete napsat sami.
Ve verzi .NET 2.0 je SOAP formatter označen jako obsolete (tj. není doporučeno jej používat.)

Transportovače ☺ (viz transport sink)

Dotnet podporuje TCP (sokety) a HTTP. Další si můžete napsat sami (třeba jako nadstavbu nad vestavěnými třídami, které umožňují použít HTTP, FTP, UDP a TCP).
Poznámka: HTTP lze použít jen se SOAP. (Pochopitelně, binární data přenesete přes TCP.)

Jak se komunikace konfiguruje?

Nejjednodušší je vložit si konfiguraci Remotingu přímo do .config souboru vaší aplikace. (Tedy dvě konfigurace – pro klienta a pro component host). Je zde jeden háček: Nemusíte pak sice nic programovat, ale prase aby se v tom .config vyznalo. Jediná možnost, která se mi osvědčila, je mít nějakou šablonu „standardní konfigurační soubor" a tu si pak vždycky okopírovat, když potřebujete něco konfigurovat. Ručně to nenapíšete.

Příklad 3: Konfigurační XML klienta.

```
<configuration>
   <system.runtime.remoting>
      <application name="SimpleClient">
         <client url="tcp://localhost:8080/SimpleServer">
            <activated
             type="RemoteObjects.RemoteObject, RemoteObjects"/>
         </client>
         <channels>
            <channel ref="http client" port="8080">
              <clientProviders>
                 <formatter ref="binary">
              </clientProviders>
```

```
            </channel>
         </channels>
      </application>
   </system.runtime.remoting>
</configuration>
```

Příklad 4: Konfigurační XML component hostu.

```
<configuration>
   <system.runtime.remoting>
      <application name="SimpleServer">
         <service>
            <activated type="RemoteObjects.RemoteObject,
                              RemoteObjects"/>
         </service>
         <channels>
            <channel ref="http server" port="8080">
               <serverProviders>
                 <formatter ref="binary">
               </serverProviders>
             </channel>
         </channels>
      </application>
   </system.runtime.remoting>
</configuration>
```

Položky <service> a <channels> můžete použít víckrát a zaregistrovat tak víc objektů a/nebo víc komunikačních kanálů. Konfigurace tímto způsobem má však i několik nevýhod:

- Máte pevné číslo portu, tj. nemůžete spustit server dvakrát.
- Nemůžete mít víc kanálů stejného typu (např. 2x „http server“ nejde)
- Nemůžete přiřadit <service> jen k některým kanálům – všechno je dostupné na všech kanálech.

Tyto nevýhody lze vyřešit tak, že místo konfiguračního XML to celé uděláte „ručně“ programově. Není to ani tak složité, čili osobně tomu dávám přednost.

Obousměrná komunikace

Obousměrná komunikace, čili přidání možnosti, aby server zavolal něco na klientovi, funguje dosti krkolomně. Samozřejmě se to v C# deklaruje jako event, jenže vyžaduje to obousměrné navazování spojení. Čili pokud jeden počítač mám doma za NATP routerem (tj. bez veřejné IP adresy), tak mám smůlu, neboť co není z venku vidět, to nemůže odchytávat eventy.
Osobně mám pocit, že obousměrná komunikace v Remotingu prostě neexistuje. To, co tam existuje, je totiž na nic.

XML Web Services

XML webové služby (XWS) jsou věcí podstatně jednodušší než Remoting. Všude tam, kde je možné XWS použít, tak je to také doporučeno – dělat věci v Remotingu je přecijen složitější.
Jak je patrné z názvu, jde o webové služby, čili základem je web a http server. XWS nemůžete provozovat (nabízet) bez http serveru. To je možná trošku nevýhoda, ale zato všechno ostatní jsou výhody. Nic jednoduššího nehledejte!

- XWS jsou stateless – přesně ve stylu práce HTTP serveru (jako SingleCall v Remotingu)
- XWS mají menší funkcionalitu, takže se snáze programují

- XWS používá SOAP komunikaci, takže je v runtimu pomalejší než binární komunikace v Remotingu
- XWS jsou cross-platformní
- XWS podporuje „hledání“ a „propagaci“ služeb na internetu
- XWS nepoužívá dedikovaný server – je hostováno na běžném HTTP serveru
- XWS nepotřebuje žádné nastavení firewallu – používá obvykle jen port 80

Microsoft nedoporučuje používat Remoting v těchto dvou případech:
- Systém v heterogenním prostředí (nejen .NET platforma)
- Systém překračující trust zones (ven z lokální sítě)

XWS v těchto případech funguje dobře.

Visual Studio: Založíme projekt typu ASP.NET Web Service
Poznámka: Microsoft považuje XWS za doplněk/rozšíření systému ASP.NET.

V prázdném projektu se podívejte na metodu HelloWorld – atribut [WebMethod]

Jak to vyzkoušet

Visual Studio spolupracuje přímo s lokální instalací IIS. Takže zmáčkněte F5, tím přejdete na webové rozhraní vaší služby – dobře si to prozkoumejte. ☺
Poznámka: VS2003 na to vyžaduje administrátorská práva. VS2005 už údajně lze používat i bez administrátorských práv (nezkoušel jsem – jsem admin ☺).

A co klient?

Klientskou aplikaci můžete udělat jednoduše tak, že použijete třídy pro práci s XML, SOAP a HTTP přímo v .NETu. Nikdy jsem však neviděl praktické nasazení XWS – znám to jen jako hračku. ☺

Úkol pro vás

Udělejte stejný chat, jako jsem vám připravil v Remotingu, pomocí XWS. Je to velmi jednoduché, pomocí wizardů ve Visual Studiu to budete mít za chvilku. Potřebujete jen počítač, na kterém máte běžet IIS s podporou ASP.NET a uživatelská práva, se kterými to jde dělat.

ASP.NET

Toto je microsoftí náhrada za JSP a PHP. Je to v podstatě o tom, že obsah HTML stránek připravujete programem v C#. U malých programů je to vyloženě smrt, u větších projektů to začne být díky použití objektově orientovaného C# čím dál lepší. Přesto, osobně radši napíšu všechno v PHP – tak velké weby, aby mě to primitivní PHP otravovalo, jsem zatím nedělal.

Pozor na jednu věc: Používají se teď hlavně verze ASP.NET 1.1 a 2.0. Nová „dvojka“ se hodně liší. Pokud se to chcete teprve začít učit, jděte rovnou na novou verzi – je to 10x méně pracné. Ale pozor! Některé knihy či tutoriály mluví o ASP.NET 2.0 a přitom používají pořád ten starý a blbý způsob dle stylu ASP.NET 1.1.

Literatura:

1. Matthew MacDonald. *Microsoft .NET Distributed Applications: Integrating XML Web Services and .NET Remoting.* Microsoft Press, 2003. ISBN 0-7356-1933-6.
2. Tom Barnaby. *Distributed .NET Programming in C#.* Apress, 2002. ISBN 1-59059-039-2.
3. Scott McLean, James Naftel, and Kim Williams. *Microsoft .NET Remoting.* Microsoft Press, 2003.
4. R. Allen Wyke, Sultan Rehman, Brad Leupen. *XML Programming (Core Reference).* Microsoft Press, 2001. ISBN 0-7356-1185-8.
5. Scott Short . *Building XML Web Services for the Microsoft .NET Platform.* Microsoft Press, 2002. ISBN 0-7356-1406-7.
6. Adam Sills, et al. *XML .NET Developer's Guide.* Syngress Publishing, 2002. ISBN 1-928994-47-4.
