

Реализиране на комуникация между процеси

- Да се установи комуникация между обекти, които работят в различни процеси, независимо дали са на един и същ компютър или на отдалечени компютри, е отделна задача при изграждането на разпределени приложения.

Три средства за реализиране на комуникация между процеси

ASP.NET, Enterprise Services, и .NET Remoting са трите средства за реализиране на комуникация между процеси.

- ASP.NET предоставя инфраструктура, хоствана от Internet Information Services (IIS) и работи добре със всички основни типове.
- Enterprise Services е разработен за COM + архитектурата.
- .NET Remoting е обща и разширяема система за комуникация между процеси, която може да бъде самостоятелно хоствана или хоствана в IIS, разработена за разпределени обектно-ориентирани приложения.

NET REMOTING ОТДАЛЕЧЕНО ИЗВИКВАНЕ

Съдържание

- Какво е .NET Remoting?
- Remoting инфраструктурата
- Remoting канали. Регистрация на канал
- Активация на обекти. Активация от сървъра. Активация от клиента
- Маршализация (Marshaling). Marshal-by-Value обекти. Marshal-by-Reference обекти
- Живот на обектите (Lifetime). `ILease`
- Създаване на Remoting сървър и клиент
- Remoting конфигурационен файл

Разпределени приложения

- Повечето днешни приложения са разпределени
 - Състоят се от няколко раздалечени компонента, които си взаимодействат
- Модели за разпределени приложения
 - Модел "клиент/сървър"
 - Модел "разпределени обекти"
 - DCOM – ползва се в Microsoft Windows
 - CORBA – отворен стандарт, доста сложен
 - .NET Remoting – ползва се в .NET Framework
 - Модел "Web услуги"

Какво е .NET Remoting?

- Remoting технологията в .NET Framework осигурява прозрачен достъп до отдалечени обекти:
 - Обекти от друго приложение;
 - Обекти от друг процес на същия компютър или на който и да е компютър от същата мрежа;
 - Обекти на отдалечена машина;
- Използва се за улесняване на комуникацията при разпределени приложения
- .NET Framework предоставя специализирана инфраструктура за Remoting технологията

Remoting инфраструктурата

- .NET Remoting инфраструктурата се състои от:
 - Канали – пренасят съобщения от и към отдалечени обекти
 - Форматери – кодират и декодират съобщенията в някакъв формат
 - Прокси класове – предават извикванията на методи към отдалечените обекти
 - Механизми за активация – осигуряват отдалечено инстанциране на обекти
 - Маршализация – осигурява пренос на обекти, техните свойства, полета и т.н.

Remoting канали

- Каналите (channels) в .NET Remoting
 - Транспортират съобщения от и към отдалечени обекти
 - Могат да бъдат:
 - TCP – използват чист TCP сокет
 - HTTP – използват протокола HTTP
 - Други – дефинирани от потребителя
 - Използват се както от клиентските, така и от сървърните приложения
 - Използват TCP портове в нашето приложение
 - Трябва да бъдат регистрирани преди използване

Регистрация на канал

- Регистрацията на Remoting канал може да стане по 2 начина:

- Чрез класа `System.Runtime.Remoting.Channels.ChannelServices`

```
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Tcp;  
...  
TcpChannel channel = new TcpChannel(3333);  
ChannelServices.RegisterChannel(channel);
```

- Чрез конфигурационен файл

```
RemotingConfiguration.Configure(  
    "MyClient.exe.config");
```

Форматери

Ролята на форматерите в Remoting инфраструктурата е да сериализират съобщенията между двете страни в определен формат. Форматерите биват:

- Бинарен форматер
 - Голяма производителност
 - Слаба съвместимост с други технологии (специфичен за .NET Remoting)
- SOAP форматер
 - XML базиран формат – добра съвместимост с други технологии
 - По-лоша производителност

Форматери (2)

- Форматери по подразбиране:
 - HTTP каналите използват по подразбиране SOAP формater
 - Това позволява използване на отдалечения метод като Web услуга
 - TCP каналите използват по подразбиране бинарен формater
- Могат да се указват и ръчно:

```
SoapServerFormatterSinkProvider provider =  
    new SoapServerFormatterSinkProvider();  
IDictionary props = new Hashtable();  
props["port"] = 12345;  
TcpChannel chan = new TcpChannel(props, null, provider);
```

Активация на обекти

- "Активация" означава създаване на отдалечен обект (инстанциране) и подготвяне на обекта за използване
- Активацията бива два вида:
 - Активация от страна на сървъра (server-side activation)
 - Активация от страна на клиента (client-side activation)
- Начини за активация:
 - Чрез оператора `new`
 - Чрез `Activator.GetObject(...)`
 - Чрез `Activator.CreateInstance(...)`

Активация на обекти (2)

- При активация на отдалечен обект Remoting инфраструктурата динамично създава прокси клас при клиента
- Прокси класът изглежда като истинския клас, но препраща обръщенията към отдалечения обект:
 - достъп до полета, свойства, събития и други елементи
 - извикване на методи
 - възникналите изключения
- За програмиста отдалечеността на обекта е прозрачна

Активация от сървър

- При активация от сървър (server-side activation)
 - Сървърът създава автоматично обекта при клиентска заявка
- Има два режима на сървърна активация:
 - Single-Call – за всяка клиентска заявка се създава отделна инстанция
 - Singleton – една и съща инстанция обслужва всички клиенти
 - Инстанцията се създава само веднъж
 - Данните на инстанцията са общи (споделени) за всички клиенти

Активация от сървъра (2)

- Сървърът задава за всеки обект, че изисква сървърна активация:
 - Регистриране на Single-Call server activated object:

```
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof(CommonTypes.Library), "Library",  
    WellKnownObjectMode.SingleCall);
```

- Регистриране на Singleton server activated object:

```
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof(CommonTypes.Library), "Library",  
    WellKnownObjectMode.Singleton);
```


Активация от клиента

- При активация от клиента (client-side activation)
 - Обектите се създават по заявка от клиента (ръчно, изрично)
 - Всяка заявка за създаване на обект създава нова, отделна инстанция
 - Всеки клиент работи със свои собствени обекти на сървъра
 - Регистриране на client-activated обект:

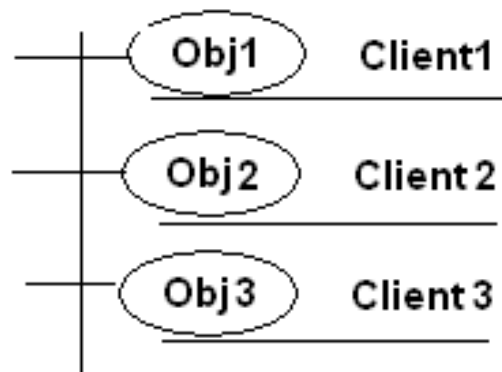
```
RemotingConfiguration.RegisterActivatedServiceType(  
    typeof(CommonTypes.Library) );
```

Маршализация (Marshaling)

- Маршализацията осигурява пренос на обекти между Remoting клиента и сървъра. Използва при:
 - предаване на параметри
 - връщане на стойност
 - достъп до свойства и полета
- Има 3 типа обекти:
 - Marshal-by-Value обекти – пренасят се по стойност (сериализирани)
 - Marshal-by-Reference обекти – пренасят се по отдалечена референция (`ObjRef`)
 - Not-Marshaled обекти – не се пренасят

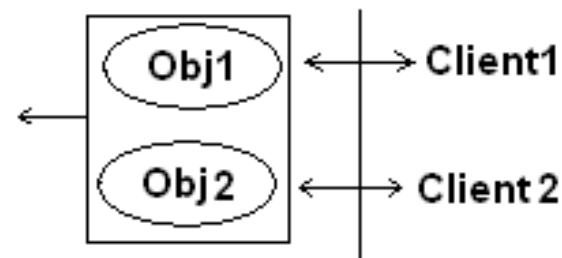
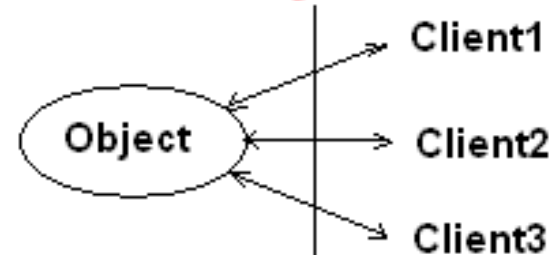
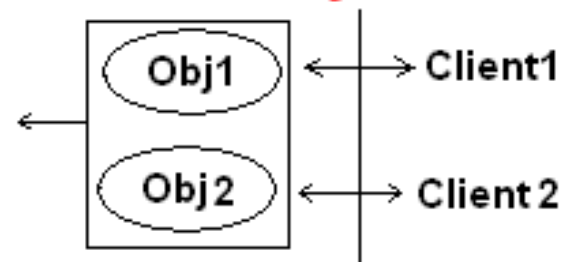
Server

Client

By Value

Server

Client

By Reference**Singleton****SingleCall**

Обобщена таблица

	<u>Marshal-by-value</u> (MBV)	Marshal-by-reference (MBR) (proxy обекти при клиента, реални обекти на сървъра)
<i>Activated from client</i>	<activated> Обект при клиента	<activated> Обект на сървъра, докато клиента не унищожи обекта или не завърши сесията.
<i>Activated from server</i>	<i>no</i>	<wellknown mode="Singleton" > създава се един обект, който се използва от всички клиенти
		<wellknown mode=" Singleton "> създава се обект за всяко обръщение на всеки клиент

Marshal-by-Value обекти

- Marshal-by-Value обекти
 - Всички сериализирани типове ([Serializable], ISerializable)
 - Пренасят се по стойност (копие от тях се предават до отдалечената машина)
 - При промяна се променя само локалното копие на обекта
 - Всичките им полета и свойства се предават наведнъж.

```
[Serializable]  
public struct Book {  
    public string mAuthor;  
    public string mTitle;  
}
```

Marshal-by-Reference обекти

Marshal-by-Reference обекти

- Всички типове, които наследяват класа `System.MarshalByRefObject` (включително сериализируеми типове)
- Пренасят се по отдалечена референция, по която се създава прокси клас
- Всички операции се пренасят към отдалечения обект през прокси класа
 - Всяко четене или промяна на поле или свойство предизвиква отдалечено извикване
 - лоша производителност

```
public class Book : MarshalByRefObject { ... }
```

Живот на обектите (Lifetime)

- В .NET Remoting се използва техниката "lease-based lifetime" ("живот, отпуснат назаем")
 - Използва се само при Singleton Server-Activated обекти и Marshal-by-Reference Client-Activated обекти
 - При създаване на обект му се дава "време на живот"
 - След изтичане на това време обектът се унищожава от Garbage Collector
 - Времето за живот се настройва при създаване на обекта и не може да се променя след това

Интерфейсът ILease

`InitialLeaseTime` – начален живот (5 мин. по подразбиране)

- `RenewOnCallTime` – удължаване на живота при извикване – (2 мин.)
- `SponsorshipTimeout` – настройва на т. нар. "спонсори" (2 мин.)
- `CurrentLeaseTime` – оставащо време
- `CurrentState` – състояние на времето за живот (`Initial`, `Active`, `Expired`, `Renewing`)

```
ClientActivatedType caObject = new ClientActivatedType();  
ILease serverLease = (ILease)  
    RemotingServices.GetLifetimeService(caObject);  
serverLease.RenewOnCallTime =  
    new TimeSpan(0, 10, 0); // 10 min.
```


Създаване на Remoting сървър

1. Регистриране на Remoting канал

```
TcpChannel channel = new TcpChannel(12345);  
ChannelServices.RegisterChannel(channel);
```

2. Регистриране на обектите, които ще бъдат достъпни отдалечено

```
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof(CommonTypes.Library), "Library",  
    WellKnownObjectMode.Singleton);
```

Създаване на Remoting клиент

1. Регистриране на Remoting канал

```
TcpChannel channel = new TcpChannel();  
ChannelServices.RegisterChannel(channel);
```

2. Активиране на отдалечен обект:

```
Library remoteLibrary = (Library)  
    Activator.GetObject(typeof(Library),  
        "tcp://localhost:12345/Library");
```

или така:

```
RemotingConfiguration.RegisterWellKnownClientType(  
    typeof(CommonTypes.Library),  
    "tcp://localhost:12345/Library");  
Library remoteLibrary = new Library();
```

Външно конфигуриране

- Конфигурирането на Remoting инфраструктурата може да става с външен за приложението файл
- Конфигурационният файл може да съдържа:
 - Канали и форматери
 - Схема на живот (lease-based lifetime)
 - Регистрация на клиентски обекти
 - Регистрация на сървърните обекти
- Зарежда се програмно (изрично):

```
RemotingConfiguration.Configure("MyConfig.xml");
```

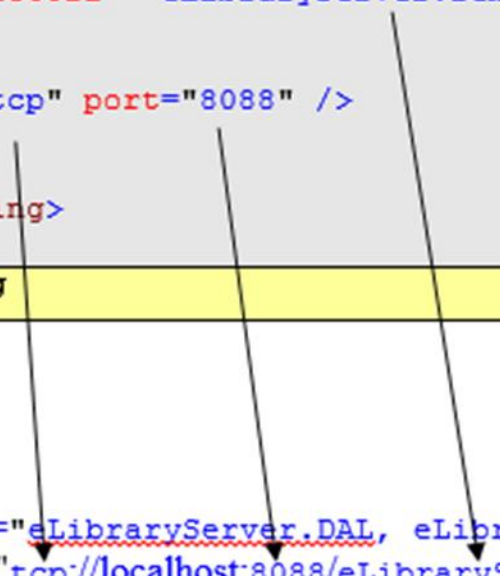
Remoting конфигурационен файл

eLibraryHost.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="SingleCall"
          type="eLibraryServer.DAL, eLibraryServer"
          objectUri=" eLibraryServer.rem" />
      </service>
    </application>
    <channels>
      <channel ref="tcp" port="8088" />
    </channels>
  </system.runtime.remoting>
</configuration>
```

eLibraryClient.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <wellknown type="eLibraryServer.DAL, eLibraryServer"
          url="tcp://localhost:8088/eLibraryServer.rem" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```



The diagram illustrates the configuration mapping between the server and client files. Three arrows originate from the server configuration and point to the client configuration:

- An arrow from the `mode="SingleCall"` attribute in the server's `<wellknown>` element points to the `type` attribute in the client's `<wellknown>` element.
- An arrow from the `type="eLibraryServer.DAL, eLibraryServer"` attribute in the server's `<wellknown>` element points to the `type` attribute in the client's `<wellknown>` element.
- An arrow from the `objectUri=" eLibraryServer.rem"` attribute in the server's `<wellknown>` element points to the `url` attribute in the client's `<wellknown>` element.