Canais revistos

- **Por razões de segurança, a partir do Framework 1.1 (VS 2003) foi modificada a politica de *deserialização* automática de objectos criando dois níveis.**
  - **Low (nível por omissão):**
  - **Full**
- **Nos seguintes casos, será necessário alterar o nível por omissão para Full:**
  - **Invocação de um método remoto com argumentos ObjRef (*MarshalByRefObject*)**
  - **Objectos que implementem a interface *ISponsor***
  - **Objectos que alterem a cadeia de *sinks* do canal entre o cliente e o servidor**

A mudança do nível para *Full* é feita ao nível do *Formatter* do canal associado à chegada do pedido:
- **Canal do servidor**
- **Canal do cliente no caso de existirem *callbacks***

Ver em *Visual Studio Help* ou *MSDN*

Automatic Deserialization in .NET Framework Remoting

See Also

Language Filter: C#

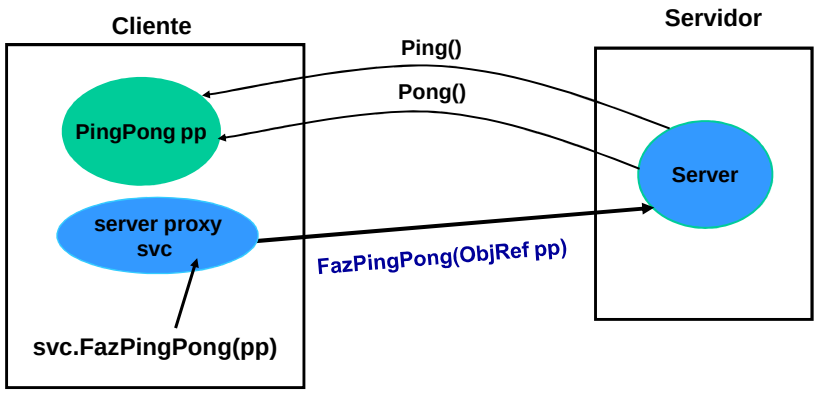
The following lists describe the .NET Framework remoting deserialization levels:

- **Low** (default level)
  The default deserialization level in .NET Framework remoting supports deserialization of the following types:
  - Remoting infrastructure objects. These are the types required to make remoting work at a basic level.
  - Primitive types, and reference and value types that are composed of primitive types.
  - Reference and value types that are marked with the SerializableAttribute attribute but do not implement the ISerializable interface.
  - System-provided types that implement ISerializable and make no other demands outside of serialization.
  - Custom types that have strong names and live in an assembly that is not marked with the **AllowPartiallyTrustedCallersAttribute** attribute.
  - Custom types that implement **ISerializable** and make no other demands outside of serialization.
  - Types that implement the ILease interface and are not MarshalByRefObject objects.
  - ObjRef objects used for activation (to support client-activated objects); that is, the client can deserialize the returned ObjRef but the server cannot.
- **Full**
  The **Full** deserialization level in .NET Framework remoting supports all other scenarios, including the deserialization of the following additional types:
  - **ObjRef** objects passed as parameters.
  - Objects that implement the ISponsor interface.
  - Objects that are inserted between the proxy and client pipeline by the IContributeEnvoySink interface.
  - Delegate types passed as parameters.
  - Objects that inherit from MarshalByRefObject passed as parameters.
  - **ISerializable** types passed as parameters.
  - Types stored in the GAC and not marked with the **AllowPartiallyTrustedCallersAttribute** attribute.

  If your application requires the use of remoting features that are available only at the **Full** deserialization level, you must provide the type of authentication and the level of encryption necessary to protect any resources that might be at risk by using these advanced features in remote scenarios.

You can set the deserialization level programmatically or by using an application configuration file.

Exemplo de Uso de *Callbacks*



Nota: Por razões de desempenho e de possíveis *firewalls* envolvidos esta solução deve ser evitada, isto é, as chamadas do servidor ao cliente (*callbacks*) devem só ser usadas quando estritamente necessárias e possíveis, por exemplo, em contextos intranet.

Interfaces

```
using System;

namespace PingPong {

   public interface IPingPong {
      bool EstPing { get; set; }
      void Ping();
      void Pong();
   }

   public interface IServer {
     void FazPingPong(IPingPong pp);
   }
}
```

Servidor

```
using System;
using System.Collections;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using PingPong;
namespace Server {

   class RemoteServer :MarshalByRefObject, IServer {

      public void FazPingPong( IPingPong pp ) {
         if (pp.EstPing) pp.Ping(); else pp.Pong();
         pp.EstPing=!pp.EstPing;
      }
   }
   . . .
```

Note que esta simples afectação implica dois acessos do servidor ao cliente. Um exemplo a evitar e que pode acontecer quando não temos em atenção que desenhar objectos distribuídos requer cuidados diferentes dos usados normalmente.

Servidor: Main

```
. . .
class ServerMain {
   static void Main() {
      SoapServerFormatterSinkProvider serverProv = new SoapServerFormatterSinkProvider();
      serverProv.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.Full;
      SoapClientFormatterSinkProvider clientProv = new SoapClientFormatterSinkProvider();
      IDictionary props = new Hashtable();
      props["port"] = 1234;
      HttpChannel ch = new HttpChannel(props, clientProv,serverProv);
      ChannelServices.RegisterChannel(ch, false);

      RemotingConfiguration.RegisterWellKnownServiceType(
            typeof(RemoteServer),
            "RemoteServer.soap",
            WellKnownObjectMode.Singleton);
       // Espera pedidos
       Console.WriteLine("Server: Espera pedidos...Prima Enter para terminar\n");
       Console.ReadLine();
      }
   }
}
```

Cliente

```
using System;
using System.Collections;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using PingPong;
namespace ClientePingPong
{
   class PingPongImpl : MarshalByRefObject, IPingPong {
      private bool estping;
      public bool EstPing {get {return estping;} set {estping=value;}}
      public void Ping() {
         Console.WriteLine("PING");
      }
      public void Pong() {
         Console.WriteLine("PONG");
      }
   }
  . . .
```


Cliente: Main

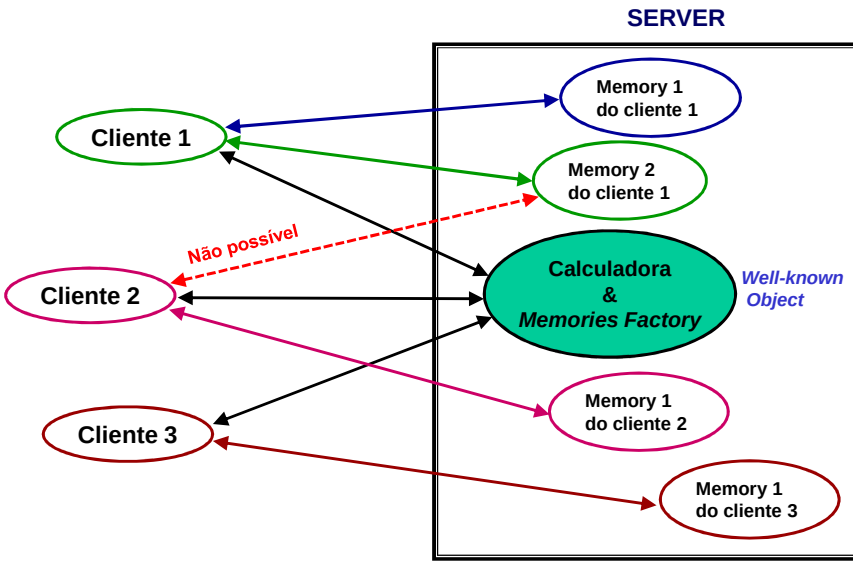
```
. . .
class Cliente {
  static void Main( )  {

      SoapServerFormatterSinkProvider serverProv = new SoapServerFormatterSinkProvider();
      serverProv.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.Full;
      SoapClientFormatterSinkProvider clientProv = new SoapClientFormatterSinkProvider();
      IDictionary props = new Hashtable();
      props["port"] = 0;
      HttpChannel ch = new HttpChannel(props,clientProv,serverProv);
      ChannelServices.RegisterChannel(ch, false);

      IServer svc = (IServer) Activator.GetObject(
            typeof(IServer),
            "http://localhost:1234/RemoteServer.soap");

      IPingPong pp = new PingPongImpl(); pp.EstPing=true;
      for (int i=0; i < 10; i++)
          svc.FazPingPong(pp);
      Console.ReadLine();
     }
  }
}
```

Exercicios

1) Considere que o cliente da Calculadora pode necessitar de guardar valores temporários em memórias (como nas máquinas de calcular). Implemente uma solução por forma a que essas memórias sejam implementadas como objectos no lado do servidor, usando duas estratégias alternativas: com *Client Activated Objects* ou usando o padrão *Factory*, em que o objecto Calculadora passa a ser fábrica de objectos que implementam a funcionalidade de memória.

2) Implemente uma aplicação de Difusão de Mensagens com as seguintes funcionalidades:
   - ✓ O servidor aceita os seguintes serviços:
     - ***RegistaMessageBox*** – Permite aos clientes registarem uma Caixa de Mensagens, onde pretendem receber mensagens;
     - ***SendMessage*** – Permite aos clientes enviar mensagens para todos os clientes registados incluindo o próprio que enviou;
     - ***UnRegistaMessageBox*** – Permite aos clientes terminarem a sua disponibilidade para receber mensagens;
   - ✓ Desenvolva uma aplicação cliente em modo Consola que implemente uma Caixa de Mensagens e ilustre a funcionalidade do servidor.

Exercício 1) Proposta de Arquitectura



Exercício 2) Proposta de Arquitectura