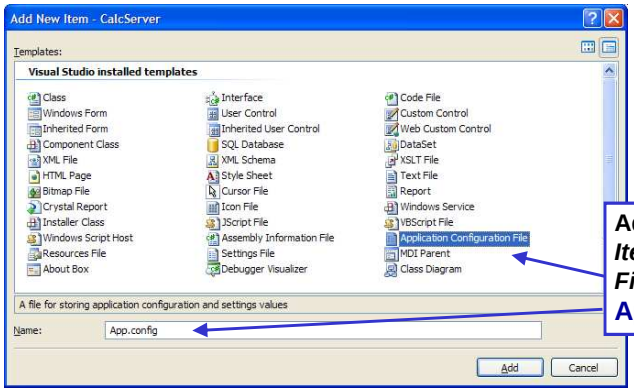
Ficheiros de Configuração

- **É possível especificar os parâmetros da maioria dos aspectos do *.NET remoting* através de ficheiros de configuração;**
- **Estes ficheiros têm formato XML com a seguinte estrutura:**

```
<configuration>
   <system.runtime.remoting>
      <application>
            <lifetime />
            <channels />
            <service   />
            <client  />
       </application>
    </system.runtime.remoting>
</configuration>
```

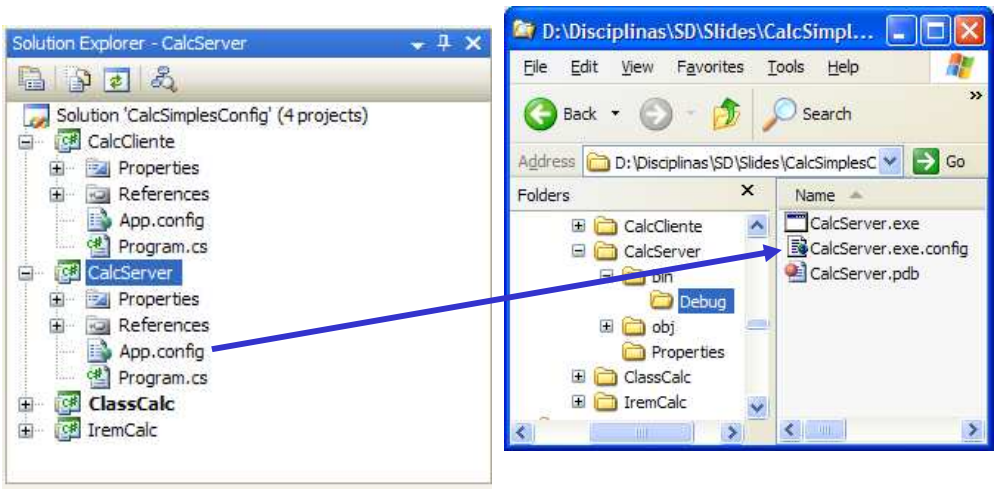

Criação de ficheiro de configuração no *Visual Studio .NET*



Adicionar ao projecto um novo *Item* (*Application Configuration File*) com o nome obrigatório App.config

- **Neste ficheiro editar a configuração em XML;**
- **Quando for feito *Build* ao projecto o VS.NET coloca automaticamente junto ao executável (…\bin\Debug) o ficheiro de configuração com nome *<ApplicationName>.config*, onde *ApplicationName* contém a extensão .EXE**

Resultado após *Build* do projecto *CalcServer*



Exemplo: Calculadora Simples com Configuração

Interface do objecto Remoto

```
using System;

namespace IRemCalc
{
   public interface ICalc
   {
      int Add(int a, int b);
      int Sub(int a, int b);
      int Mult(int a, int b);
      float Div(int a, int b);

   }
}
```

Classe *Calc* que implementa a interface *ICalc*

Classe que após compilada fica no *Assembly* ClassCalc.dll

```
using System;

namespace ClassCalc {

   public class Calc: MarshalByRefObject, ICalc {

      public Calc() {  Console.WriteLine("Calc construtor");   }

      public int Add(int a, int b) {   return a + b;   }
      public int Sub(int a, int b) {   return a - b;    }
      public int Mult(int a, int b) {   return a * b;   }
      public float Div(int a, int b) {
      try {
            return (float)(a/b);
       } catch (DivideByZeroException e) {
            throw  new DivideByZeroException("Divisão por Zero !",e);
         }
      }
   }
}
```


Ficheiro de Configuração do Servidor

calcserver.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application>
      <channels>
        <channel  ref="http" port="1234" />
      </channels>
      <service>
        <!-- Comentario:     "Namespace.Type, Assembly"  -->
        <wellknown type="ClassCalc.Calc, ClassCalc"
                   mode="Singleton"
                   objectUri="RemoteCalc.soap" />
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```

Define que o servidor vai usar um canal *http* no porto 1234 e regista um serviço de um objecto SAO (*wellknown object*) em modo *Singleton* do tipo ClassCalc.Calc existente no *assembly* ClassCalc.dll

Código do Servidor

```
using System;
using System.Runtime.Remoting;

namespace CalcServer {

   class RemCalc {

      static void Main( ) {

         string configfile = "CalcServer.exe.config";
         RemotingConfiguration.Configure(configfile, false);

         Console.WriteLine("Inicio do Server Calc. Espera pedidos");
         Console.ReadLine();
      }
   }
}
```

Referencia o *assembly* ClassCalc.dll

sem controlo de *security*

Ficheiro de Configuração do Cliente

calccliente.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application>
      <channels>
        <channel ref="http" port="0" />
      </channels>
      <client>
        <!-- Partilhar a Classe -->
        <!--                          "Namespace.Type, Assembly"  -->
        <!-- <wellknown type="ClassCalc.Calc, ClassCalc"
              url="http://localhost:1234/RemoteCalc.soap" /> -->
        <!-- Partilhar a Interface -->
        <wellknown type="IRemCalc.ICalc, IRemCalc"
                   url="http://localhost:1234/RemoteCalc.soap" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Código do Cliente – partilha da Implementação

```
using System;
using System.Runtime.Remoting;
using ClassCalc;

namespace CalcCliente {

   class Cliente  {

      static void Main( )  {
          string configfile = "CalcCliente.exe.config";
          RemotingConfiguration.Configure(configfile, false);

          Calc robj = new Calc( );
          if (RemotingServices.IsTransparentProxy(robj))  Console.WriteLine("robj é remoto");
          Console.WriteLine("5+8={0}",robj.Add(5,8));

          try {
              Console.WriteLine("5/2={0}",robj.Div(5,0));
          } catch (DivideByZeroException e){
              Console.WriteLine("Divisão por zero: {0}",e.Message);
          }
      }
   }
}
```

Referencia o *assembly* ClassCalc.dll

Pressupõe que a implementação da classe é partilhada entre o cliente e o servidor

Deve evitar-se partilha da implementação

Para tipos remotos, neste caso *Calc*, o operador *new* passa a ter um comportamento diferente, instanciando o objecto remoto.

Partilhar só a Interface

```
namespace IRemCalc {
   public interface ICalc {
       int Add(int a, int b);
       int Sub(int a, int b);
       int Mult(int a, int b);
       float Div(int a, int b);
   }
}
```

```
namespace ClassCalc {

   public class Calc: MarshalByRefObject, ICalc {

    // . . . igual

   }
}
```

```
<configuration>
  . . .
 <client>
   <wellknown type="IRemCalc.ICalc, IRemCalc“ url="http://localhost:1234/RemoteCalc.soap" />
 </client>
   . . .
</configuration>
```

Activação no cliente

```
RemotingConfiguration.Configure("cliente.exe.config", false);
WellKnownClientTypeEntry[ ] entries=
                        RemotingConfiguration.GetRegisteredWellKnownClientTypes();
// em vez de: Calc robj = new Calc();
ICalc robj = (ICalc)Activator.GetObject(entries[0].ObjectType, entries[0].ObjectUrl);
```

tag <lifetime> - Configuração do *lifetime* dos objectos

Os valores especificados no ficheiro de configuração afectam todos os objectos alojados na aplicação, enquanto o *override* do método *InitializeLifetimeService* permite modificar o tempo de vida das instâncias de uma classe em particular.

Atributos:

- leaseTime – *Time-To-Live* (TTL) inicial (5 minutos por omissão)
- sponsorshipTimeout – tempo de espera pela resposta do *sponsor* (2 minutos por omissão)
- renewOnCallTime – renovação do TTL quando um método é chamado (2 minutos por omissão)
- leaseManagerPollTime – intervalo entre testes sobre os TTLs dos objectos (10 segundos por omissão)

Os atributos são opcionais e podem ser especificados nas seguintes unidades:
D - dias
H - Horas
S - Segundos (unidade por omissão)
MS - Milissegundos

Exemplo:

leaseTime="0"
Para tempo infinito

```
<lifetime
    leaseTime="2H"
    renewOnCallTime="90MS"
    leaseManagerPollTime="20S"
/>
```


Exercício

- No exemplo *CalcSimplesConfig.zip*, o ficheiro de configuração do servidor tem a possibilidade de alterar o tempo de vida do objecto servidor;
- O servidor é *Singleton* porque tem de manter estado para suportar um campo do tipo inteiro;
- No cliente existe um *loop* que periodicamente faz chamadas ao servidor:

```
for (int i = 0; i < 100; i++) {
        robj.Val = i;
        System.Threading.Thread.Sleep(4 * 1000);
        Console.WriteLine(robj.Val);
}
```

- Teste o exemplo alterando o tempo de vida por forma a perceber as implicações de o objecto não ter um tempo de vida adequado.

tag <channels> - Definição de Canais de transporte

Na *tag* <*channels*> é possível especificar vários canais através da *tag* <*channel*>

```
<channels>
    <channel ref="tcp"  port="1234" />
    <channel ref="http" port="1235" />
    <channel ref="ipc" portName="MyIpcChannel" />
</channels>
```

Canal *InterProcess Communication*
Só disponível em .NET 2.0

Atributos da *tag* <*channel*> :

- *ref* – referência a um canal pré definido ("tcp", "http", "ipc"). No entanto é possível implementar novos tipos de canais;
- *type* – quando temos um novo canal é obrigatório este atributo, para indicar o "namespace.Classe, Assembly";
- *port* – número do porto associado ao canal do lado do servidor. Quando se usa este atributo do lado do cliente deve-se colocar 0 para que os objectos do lado do cliente possam receber *callbacks* a partir do servidor.
- portName - Nome lógico atribuído ao Canal *InterProcess Communication*

Atributos específicos

Só para canais HTTP:

- *clientConnectionLimit* – número de conexões simultâneas que podem ser abertas para o servidor (2 por omissão);
- *proxyName* – nome do *proxy server*;
- *proxyPort* – número da porta do proxy server

Só para canais TCP:

- *rejectRemoteRequests* – indica se o server aceita pedidos remotos. Quando a *true* o server só aceita pedidos de clientes locais à máquina.

Exemplo: lado server

```
<channels>
   <channel ref="http" port="1234" />
</channnels>
```

Exemplo: lado cliente

```
<channels>
   <channel ref="http" port="0"
            clientConnectionLimit="100" />
</channnels>
```

ClientProviders / ServerProviders

- O *.NET Remoting framework* é baseado em mensagens que são passadas através de vários níveis. Estes níveis podem ser substituídos, estendidos ou é mesmo possível acrescentar novos níveis;
- O vários níveis são implementados usando *message sinks*;
- Uma mensagem passa através de uma cadeia de *sinks*, sendo possível em cada um deles aceder ao conteúdo da mensagem ou mesmo alterá-lo;
- Os *ClientProviders* e *ServerProviders* usados no ficheiro de configuração, permitem especificar uma cadeia de *sinks* para fluxo das mensagens, bem como um *Formatter* para serializar a mensagem.

```
<channels>
  <channel ref="http" port=1234" >
    <serverProviders>
      <provider type="MySinks.MyProvider, Server"  />
      <formatter ref="soap" />
    </serverProviders>
  </channel>
</channnels>
```


Configuração de Canal usando Server *Formatter* com *Full Serialization*

```
<configuration>
  <system.runtime.remoting>
    <application>
      <channels>
        <channel ref="http" port="0">
          <clientProviders>
            <formatter ref="soap" />
          </clientProviders>
          <serverProviders>
            <formatter ref="soap"  typeFilterLevel="Full" />
          </serverProviders>
        </channel>
      </channels>
      <client>              <!-- Namespace.Type, Assembly  -->
        <wellknown type="PingPong.IServer, IPingPong"
                   url="http://localhost:1234/RemoteServer.soap" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

```
// No código do cliente
RemotingConfiguration.Configure("ClientePingPong.exe.config", false);

WellKnownClientTypeEntry[] entries=
        RemotingConfiguration.GetRegisteredWellKnownClientTypes();
IServer svc = (IServer)Activator.GetObject(
                                        entries[0].ObjectType,
                                        entries[0].ObjectUrl
                                        );
```

Ficheiro de configuração do Cliente no exemplo *PingPong*, anteriormente apresentado (*callbacks*).

Utilização de *Generic Types* em Canais com *SoapFormatter*

- Na documentação do MSDN verifica-se existirem restrições (gera excepções) na utilização de canais com *Formatter* SOAP:
  http://msdn.microsoft.com/en-us/library/ms172342(VS.85).aspx
- Assim, evitem usar *generic types* nas intertaces dos objectos remotos, ou usem a configuração de canais HTTP ou TCP com *formatters Binary*.

Programaticamente:

```
BinaryServerFormatterSinkProvider serverProv = new BinaryServerFormatterSinkProvider();
serverProv.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.Full;
BinaryClientFormatterSinkProvider clientProv = new BinaryClientFormatterSinkProvider();
IDictionary props = new Hashtable();
props["port"] = 1234;
HttpChannel ch = new HttpChannel(props, clientProv, serverProv);
```

Ficheiros de configuração:

```
<channels>
    <channel ref="http" port=« 1234" >
        <clientProviders>
            <formatter ref="binary" />
        </clientProviders>
        <serverProviders>
            <formatter ref="binary"   typeFilterLevel="Full" />
        </serverProviders>
    </channel>
</channels>
```


Exercício

No exemplo *PIngPong.zip* já apresentado pode analisar a configuração do Cliente através de ficheiro de configuração.
Altere o servidor para este também suportar configuração através de ficheiro de configuração.

Comparando com a configuração do cliente não esqueça de configurar o Canal usando um *ServerFormatter* com *Full Serialization*

Alterando os ficheiros de configuração teste os diferentes canais e tipos de *Serialization*

tag <service> - Registo de objectos (SAO ou CAO)

Lado servidor

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <!-- Pode conter vários <wellknown> e <activated> -->
        <wellknown />
        <activated  />
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```


tag <wellknown> - Registo de *Server Activated Objects* (SAO)

- Suporta os mesmos atributos que:
  RemotingConfiguration.RegisterWellKnownServiceType(. . .)
  - *type* - informação sobre o tipo "namespace.class, assembly". Quando o *assembly* (strong assembly) está no *Global Assembly Cache* (GAC) têm de ser também indicados: versão, cultura, e chave pública.
  - *mode* - tipo de activação: *SingleCall* ou *Singleton*
  - *objectUri* – identificador único para o objecto. Este identificador deve ter um nome com extensão .soap ou .rem para poder ser alojado no IIS.

```
. . .
  <service>
      <wellknown type="ClassCalc.Calc, ClassCalc"
                 mode="Singleton"
                 objectUri="RemoteCalc.soap" />
  </service>
. . .
```

tag < activated > - Registo de *Client Activated Objects* (CAO)

- O identificador (URI) destes objectos é o nome do servidor o único atributo a registar é o tipo do objecto:
  - *type* - informação sobre o tipo "namespace.class, assembly". Quando o *assembly* (strong assembly) está no *Global Assembly Cache* (GAC) têm de ser também indicados: versão, cultura, e chave pública.

```
. . .
  <channels>
     <channel ref="tcp" port="1234" />
  </channels>
  <service>
       <activated type="ClassCalc.Calc, ClassCalc"  />
  </service>
. . .
```

Permite a um cliente criar instâncias da classe *Calc* em tcp://<hostname>:1234/<nome aplicação server>

tag <client> - Activação de objectos (SAO ou CAO)

Lado cliente

```
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <!-- Pode conter vários <wellknown> e <activated> -->
        <wellknown />
        <activated  />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Após o registo para activação de objectos *wellknown* ou *activated* o operador *new* tem o seguinte comportamento:
O *runtime* intersecta a chamadas ao *new* e verifica se o objecto a instanciar é um *type* remoto. Em caso positivo o *runtime* instancia o objecto remoto em vez do objecto local.

tag <wellknown> - Activação de *Server Activated Objects* (SAO)

- Suporta os mesmos atributos que *Activator.GetObject(. . .)*
  - *type* - informação sobre o tipo "namespace.class, assembly". Quando o *assembly* (strong assembly) está no *Global Assembly Cache* (GAC) têm de ser também indicados: versão, cultura, e chave pública.
  - *url* – localização do objecto incluindo o *uri* atribuído no servidor.

Note que um *type* não pode ser registado em diferentes *endpoints*

```
. . .
  <client>
        <wellknown type="ClassCalc.Calc, ClassCalc"
                   url="http://localhost:1234/RemoteCalc.soap" />
  </client>
. . .
```


tag < activated > - Activação de *Client Activated Objects* (CAO)

- Para cada servidor existente coloca-se:
  <client url=" …../nome-do-servidor" >
  - *type* - informação sobre o tipo "namespace.class, assembly". Quando o *assembly* (strong assembly) está no *Global Assembly Cache* (GAC) têm de ser também indicados: versão, cultura, e chave pública.

```
. . .
  <client url="http://localhost:1234/ServerCalc" >
        <activated type="ClassCalc.Calc, ClassCalc"  />
  </client>
. . .
```

tag < appSettings > definições especificas da aplicação

```
<configuration>
   <system.runtime.remoting>
      <application>
            <lifetime />
            <channels />
            <service   />
            <client  />
       </application>
    </system.runtime.remoting>
   <appSettings>
      <add key="chave #1" value="valor da chave #1" />
      <add key="chave #2" value="valor da chave #2" />
      <add key="Tempo Lifetime" value="10" />
      <add key="Tempo RenewOnCall" value="4" />
      <add key="Tempo SponsorshipTimeout" value="4" />
   </appSettings>
   <configSections>
         <section name="sampleSection" type="System.Configuration.SingleTagSectionHandler"/>
   </configSections>
   <sampleSection setting1="value 1" setting2="value 2" setting3="value 3 "/>

</configuration>
```


tag < appSettings > Acesso na aplicação

```
using System;
using System.Configuration;

namespace AppSettings  {

   class App  {
      static void Main( ) {

         // Acesso ao valor do par (key,value) no AppSettings do ficheiro de configuração
         Console.WriteLine(ConfigurationSettings.AppSettings["chave #1"]);
         Console.WriteLine(ConfigurationSettings.AppSettings["Tempo Lifetime"]);
          // Acesso ao valor do par (key,value) numa nova section do ficheiro de configuração
         Hashtable ht = (Hashtable)ConfigurationSettings.GetConfig("sampleSection");
         Console.WriteLine(ht["setting1"] + " - " + ht["setting2"] + " - " + ht["setting3"]);

      }
   }
}
```

devolve: valor da chave #1

devolve: 10

Channel IPC (*InterProcess Communication*) - .NET 2.0

- A utilização de canais HTTP ou TCP quando se usa *.NET Remoting* numa mesma máquina introduz um *overhead* desnecessário;
- A partir do .NET 2.0 passou a existir um novo canal optimizado para comunicação entre processos que se executam numa mesma máquina.
- Por omissão o canal *IpcChannel* usa o *binary formatter*

No exemplo seguinte, o canal *IpcChannel* é configurado programaticamente no lado do Cliente e com ficheiro de configuração no lado do servidor, ilustrando assim a duas formas de iniciar um canal *IpcChannel*.

Interface partilhada entre Cliente e Servidor

```
namespace IRemObject  {

  public interface IRemOla
  {
      string ola(string nome);
   }

}
```

Servidor

```
namespace IpcServer  {
      // Classe que implementa a interface
      public class RemoteOlaMundo : MarshalByRefObject, IRemOla   {
         public string ola(string nome) {   return "Ola " + nome;   }
      }

      class Program {
         static void Main(string[] args)  {
            Console.WriteLine("Configuração do IpcServer...");
            RemotingConfiguration.Configure("IpcServer.exe.config", false);
            Console.WriteLine("Verificar os channels: configurados");
             // Para todos os Channel no ficheiro de configuração
            foreach (IChannel channel in ChannelServices.RegisteredChannels)  {
               Console.WriteLine("Channel registado: " + channel.ChannelName);
               if (channel is IpcChannel)  {
                  if (((IpcChannel)channel).ChannelData != null) {
                     ChannelDataStore dataStore = (ChannelDataStore)((IpcChannel)channel).ChannelData;
                     foreach (string uri in dataStore.ChannelUris) { Console.WriteLine("Channel com URI: " + uri);  }
                  }else {  Console.WriteLine("Channel sem dados de definição!");    }
               }else {  Console.WriteLine("Não existe um Ipc Channel");    }
            }
            Console.WriteLine("--- Espera Pedidos...");    Console.ReadLine();
         }
      }
}
```


Ficheiro de configuração do servidor

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <system.runtime.remoting>
    <application name="MyServer">
    <channels>
        <channel ref="ipc" portName="MyIpcChannel" />
     </channels>
     <service>
       <wellknown type="IpcServer.RemoteOlaMundo, IpcServer"
                  objectUri="RemObject.rem"
                  mode="SingleCall" />
     </service>
</application>
  </system.runtime.remoting>

</configuration>
```

Cliente – Inicia o canal programaticamente

```
namespace IpcCliente
{
   class Program  {
      static void Main(string[] args)  {
         try   {
            Console.WriteLine("Configuração do Ipc Channel...");
            IpcClientChannel clientChannel = new IpcClientChannel();
            ChannelServices.RegisterChannel(clientChannel, false);

            Console.WriteLine("Obter proxy para remote object...");
            IRemOla robj = (IRemOla)Activator.GetObject(
                                                typeof(IRemObject.IRemOla),
                                                "ipc://MyIpcChannel/RemObject.rem");

            Console.Write("Qual o seu nome? ");   string nome= Console.ReadLine();
            Console.WriteLine( robj.ola(nome) );
         }catch (Exception ex)  {  Console.WriteLine("General Exception: " + ex.Message);  }
         Console.ReadLine();
      }
   }
}
```


Estudar exemplo: *IpcChannel.zip*

- No exemplo ***IpcChannel***, demonstrado nas aulas, existem algumas características interessantes que deve analisar, por exemplo:
  - O servidor regista 3 canais, ***Ipc, Http*** e ***Tcp***, através do ficheiro de configuração;
  - No inicio do servidor é obtido do *runtime* informação que permite identificar os vários *URL* possíveis para acesso ao objecto remoto;
  - Existem 3 clientes que podem em simultâneo aceder ao servidor:
    - ✓ Cliente que se executa na máquina do servidor e que usa o canal ***Ipc***;
    - ✓ Cliente que se executa em qualquer máquina e que usa o canal ***Tcp***;
    - ✓ Cliente que se executa em qualquer máquina e que usa o canal ***Http***;