.NET Remoting

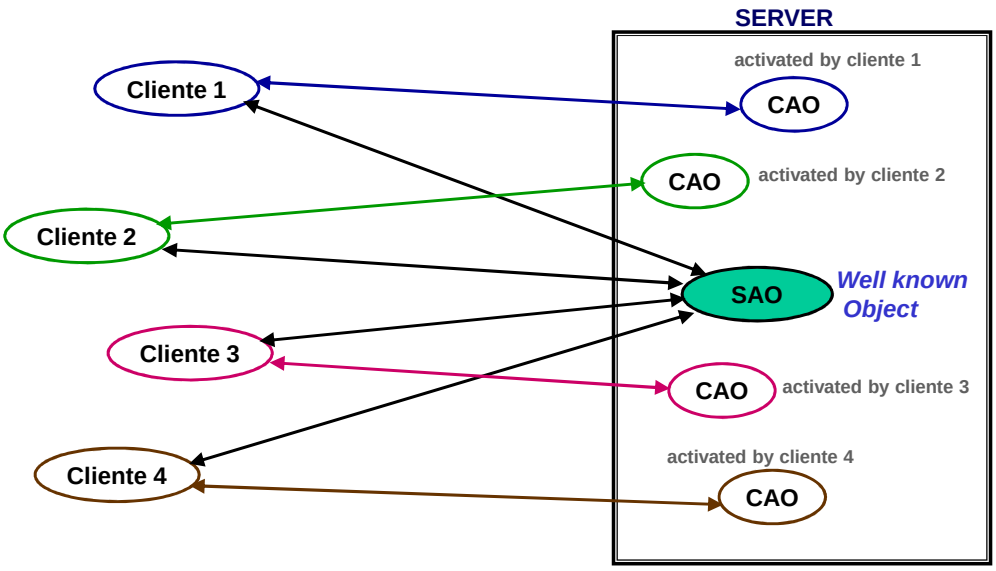
Client Activated Objects (CAOs)

Client Activated Objects (CAOs)

Os objectos *Client Activated Objects* (CAO) permitem que cada cliente active instâncias remotas, não partilhadas por outros clientes. Os objectos remotos CAO persistem para além da chamada de um método, isto é, os objectos *CAO* podem manter estado (*Stateful Objects*).

- **Se partilharmos a implementação (partilha do *Assembly*), os objectos CAO podem ser instanciados com o operador new() e podem ter construtores com argumentos.**
- **A partilha do *Assembly* também pode ser útil em cenários em que interessa podermos instanciar objectos localmente ou em alternativa remotamente.**
- **No entanto, estamos a violar um dos princípios da programação distribuída (devemos ter dependência de interfaces e não de implementações).**

Objectos SAOs versus CAOs



Objecto remoto (*Marshal By Reference*)

O Assembly resultante *ClassAlunoCAO.dll* vai ser partilhado entre o Servidor e o Cliente

```
namespace Alunos  {

   public class Aluno : MarshalByRefObject  {
      private string myNome;
      public string Nome { get {return myNome;} set {myNome=value;}}

      public Aluno( )    {
         Console.WriteLine("Default Constructor ()");
         myNome="Sem Nome";
      }
      public Aluno(string nome)   {
         Console.WriteLine("Construtor com nome: Aluno {0}", nome);
         myNome = nome;
      }
      public string AlunoHello( )   {
         Console.WriteLine("execução do método AlunoHello() {0}", myNome);
         return "Hello " + myNome;
      }
   }
}
```

Dois construtores

Servidor que regista um *type* para activações remotas CAO

Servidor

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using Alunos;
namespace ServerAluno  {
   class ServerMain  {
      static void Main( )  {
         //Criar o Canal Http
         HttpChannel ch = new HttpChannel(1234);
         // Registar o canal
         ChannelServices.RegisterChannel(ch, false);
         // Registar nome do servidor
          RemotingConfiguration.ApplicationName="ServerCAO";
          // Registar o type Aluno como Client Activated Object (CAO)
          RemotingConfiguration.RegisterActivatedServiceType(  typeof(Aluno)  );

         // Espera pedidos
         Console.WriteLine("Server: Espera pedidos...Prima Enter para terminar\n");
      }
   }
}
```


Cliente que activa e instancia objectos CAO

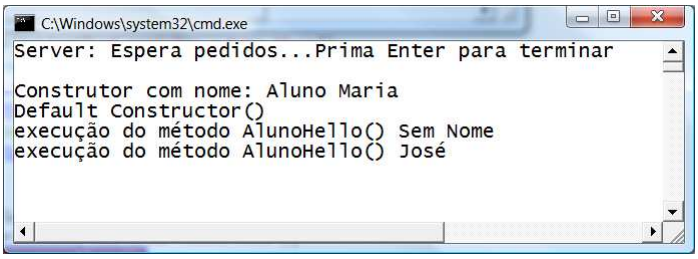
Cliente

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using Alunos;
namespace ClientAluno  {
   class ClientMain  {
      static void Main( ) {
         HttpChannel ch = new HttpChannel(0);   ChannelServices.RegisterChannel(ch, false);
         Console.WriteLine("vai registar o type Aluno");
         RemotingConfiguration.RegisterActivatedClientType(
                                                typeof(Aluno),  "http://localhost:1234"
                                                // ou caso se especifique o nome do serviço
                                                // "http://localhost:1234/ServerCAO"
          );
         Aluno maria = new Aluno("Maria");     Aluno jose = new Aluno();
         Console.WriteLine(maria.AlunoHello());    Console.WriteLine(jose.AlunoHello());
         jose.Nome="José";   Console.WriteLine(jose.AlunoHello());
         Console.ReadLine();
      }
   }
}
```

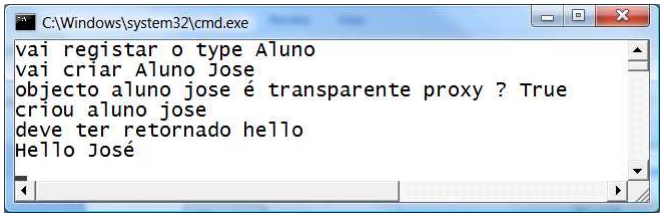
Veja nos slides seguintes o duplo comportamento do operador new(), consoante o tipo Aluno está ou não registado como remoto.

Com *RegisterActivatedClientType(typeof(Aluno), http://localhost:1234)*

Servidor

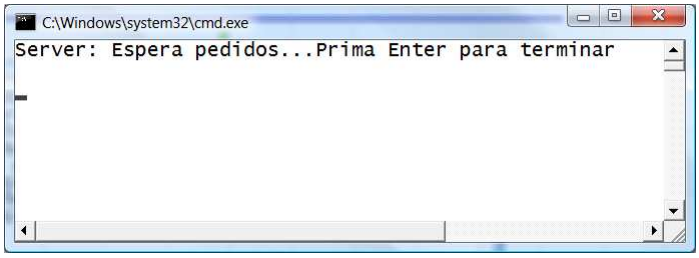


Cliente

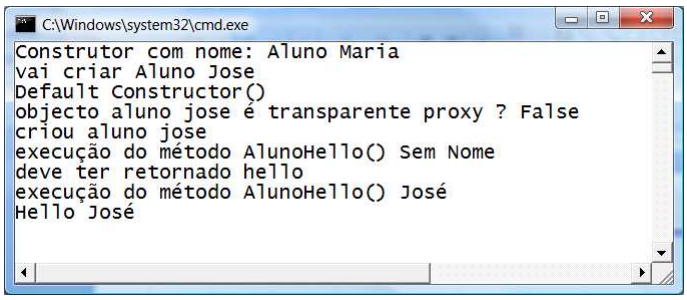


Sem *RegisterActivatedClientType(typeof(Aluno), http://localhost:1234)*

Servidor



Cliente



Objectos CAO e a partilha de *assemblies* entre Cliente e servidor

- Se por um lado podemos instanciar no cliente objectos com o operador *new* (idêntico a objectos locais), existem sérios problemas:
  - Tem de existir partilha do *assembly* que implementa o objecto remoto, significando que não é possível, como vimos nos objectos SAO, partilhar só a interface. Assim teremos de distribuir (*deployment*) a DLL com a implementação do objecto;
  - Não permite, assim, proteger o código do objecto (através do ILDASM seria possível inspeccionar o código e verificar possíveis segredos algorítmicos);
  - Cria problemas de distribuição de versões, implicando a redistribuição de um *assembly* (DLL) por todos os clientes.

Duas Soluções:

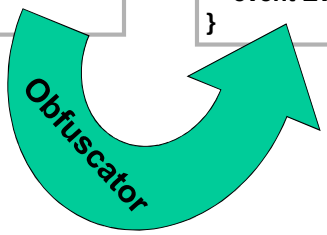
- **Obfuscators – Essencialmente para protecção de código;**
- **SOAPSUDS – Utilitário para gerar *assemblies* com *proxies* dos objectos remotos, ou mesmo código fonte, com a informação para poder referenciar no lado do cliente.**

Obfuscators

- Um *obfuscator* de código transforma uma aplicação numa outra funcionalmente idêntica à primeira, mas muito mais difícil de compreender.
- Um *obfuscator* unicamente atrasa o inevitável. Com tempo e dinheiro todo o código pode ser *reverse engineered*, mesmo para código nativo binário X86.
- O que um *obfuscator* faz é aumentar o custo do esforço requerido para fazer *reverse engineer* do código que ele produz.
- Um *obfuscator* não pode remover toda a *metadata* dos *assemblies*, mas pode modificá-la. Por exemplo um método que o programador nomeou como *CheckPasswordForValidity* é uma boa ajuda para saber onde a aplicação valida *passwords*.
- Um *obfuscator* pode mudar o nome do método para *"aaaaaaaaaa"* ou outra sequência aleatória dificultando assim a tarefa a quem pretender fazer *reverse engineered*.

O que faz um *Obfuscator* ?

```
internal class Person {
   int           age;
   double        salary;
   string        name;
   Person        spouse;
   Person[ ]     GetChildren ( ) { ... }
   string        SpouseName {
      get { . . . }
   }
   event EventHandler OnBirthday;
}
```




```
internal class a {
   int           b;
   double        c;
   string        d;
   a             e;
   a[ ]          f ( ) { ... }
   string        g {
      get { . . . }
   }
   event EventHandler h;
}
```


Utilitário SOAPSUDS

Na directoria onde se encontra o *assembly ClassAlunoCAO.dll* com a classe Aluno, podemos executar:

```
…\bin\Debug> soapsuds -ia:ClassAlunoCAO -nowp -oa:ClassAluno_metadada.dll
```

Daqui resulta o *assembly ClassAluno_metadata.dll* que é suficiente para referenciar na aplicação cliente e assim instanciar objecto remotos.

Existe ainda a hipótese de se obter um ficheiro em código fonte:

```
…\bin\Debug> soapsuds -ia:ClassAlunoCAO -nowp -gc
```

Daqui resulta um ficheiro de nome *ClassAunoCAO.cs* que tem as informações suficientes para poder incluir no projecto da aplicação Cliente

Para ter acesso às *Tools* do Visual Studio 2010, executar o *batch* file:

"C:\Program Files\Microsoft Visual Studio 10.0\Common7\Tools\vsvars32.bat"

Ficheiro ClassAlunoCAO.cs gerado pelo utilitário *SOAPSUDS*

```
using System;
using System.Runtime.Remoting.Messaging;
using System.Runtime.Remoting.Metadata;
using System.Runtime.Remoting.Metadata.W3cXsd2001;
namespace Alunos {

   [Serializable, SoapType(XmlNamespace=@ . . .)]
   public class Aluno : System.MarshalByRefObject    {
      public String Nome    {
         [SoapMethod(SoapAction=@ . . .)]
         get{return Nome;}
         [SoapMethod(SoapAction=@ . . .)]
         set{Nome= value;}
      }
      [SoapMethod(SoapAction=@ . . .")]
      public String AlunoHello()
      {
         return((String) (Object) null);
      }
   }
}
```

Property Nome

Método AlunoHello

?

Mas a classe Aluno, atrás definida, tinha um construtor com um argumento do tipo *string* que não aparece aqui !

Infelizmente não é possível usar construtores com argumentos quando se usa o utilitário *sopasuds* para gerar *metadata* de um *assembly*.

Padrão de implementação usando um objecto *Factory*

- Consiste em ter um objecto que cria instâncias (*factory)* da classe que queremos usar;
- O objecto *factory* deve ser um *Server Activated Object* (SAO);

```
namespace FactoryDesign {

   class MyClass {
      // implementação do objecto real
   }

   class MyFactory {

      public MyClass getNewInstance() {
         return new MyClass();
      }
   }
}
```

```
class App {

   static void Main( ) {
      // criar com operador new
      MyClass obj1 = new MyClass();

      // Criar usando um objecto factory
      MyFactory fact = new MyFactory();
      MyClass obj2 = fact.getNewInstance();
   }
}
```

Os objectos deixam de ser criados com o operador *new*

Exemplo Alunos com Interfaces e seguindo o padrão objecto *Factory*

Interfaces

```
using System;

namespace InterfaceFactory {

   public interface IRemAluno {
       string Nome { get; set; }
       void SetNome(string nome);
       string AlunoHello( );
   }

   public interface IRemAlunoFactory {
     IRemAluno GetNewInstanceAluno( );  // construtor por omissão
     IRemAluno GetNewInstanceAluno(string nome);  // construtor com argumento
   }
}
```


Servidor: classe Aluno

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using InterfaceFactory;
namespace ServerFactory  {
      class Aluno : MarshalByRefObject, IRemAluno     {
        private string myNome;
        public string Nome { get {return myNome;} set {myNome=value;}}
        public Aluno() {  // construtor por omissão
           Console.WriteLine("Construtor default aluno()");
           myNome="Sem Nome";
        }
        public Aluno(string nome) {  // construtor com argumento
           Console.WriteLine("Construtor com nome: Aluno {0}", nome);
           myNome = nome;
        }
        // implementação da Interface IRemAluno
        public void SetNome(string nome) {   Nome=nome;   }
        public string AlunoHello() {
           Console.WriteLine("execução do método AlunoHello() {0}",myNome);
           return "Hello " + myNome;
        }
     } // end class Aluno
     . . .
```

Servidor: Classe *factory*

```
. . .

class RemoteAlunoFactory : MarshalByRefObject, IRemAlunoFactory {

    public RemoteAlunoFactory( ) {
      Console.WriteLine("Construtor do objecto AlunoFactory\n");
    }
    // Implementação da Interface IRemAlunoFactory
    public IRemAluno GetNewInstanceAluno( ) {
      Console.WriteLine("Construtor por omissão do objecto Aluno\n");
      return new Aluno();
    }
     public IRemAluno GetNewInstanceAluno(string nome) {
        Console.WriteLine("Construtor do objecto Aluno com argumento {0}\n",nome);
        return new Aluno(nome);
     }
  }

  . . .
```


Servidor: Main

```
. . .
class ServerFactoryMain {

      static void Main() {
         //Criar o Canal Http
         HttpChannel ch = new HttpChannel(1234);
         // Registar o canal
         ChannelServices.RegisterChannel(ch, false);
         // Registar o type RemoteAlunoFactory como Server Activated Object (SAO)
         RemotingConfiguration.RegisterWellKnownServiceType(
              typeof(RemoteAlunoFactory),
              "RemoteAlunoFactory.soap",
              WellKnownObjectMode.Singleton);
              // WellKnownObjectMode.SingleCall);

         // Espera pedidos
         Console.WriteLine("Server: Espera pedidos...Prima Enter para terminar\n");

         Console.ReadLine();
      }
  }
} // end namespace ServerFactory
```

Cliente

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using InterfaceFactory;
namespace ClienteFactory  {
    class ClientRemAlunos   {

            static void Main( ) {
              HttpChannel ch = new HttpChannel(0);
              ChannelServices.RegisterChannel(ch, false);
              Console.WriteLine("Vai criar Alunos Factory\n");
              IRemAlunoFactory fact = (IRemAlunoFactory) Activator.GetObject(
                                                typeof(IRemAlunoFactory),
                                                "http://localhost:1234/RemoteAlunoFactory.soap");
              Console.WriteLine("Vai criar Aluno sem nome\n");
              IRemAluno maria = fact.GetNewInstanceAluno();
              maria.SetNome("Maria");
              Console.WriteLine("Vai criar Aluno com nome\n");
              IRemAluno jose = fact.GetNewInstanceAluno("jose");
              Console.WriteLine(maria.AlunoHello());
              jose.Nome= "mudei o nome ao jose“;
              Console.WriteLine(jose.AlunoHello());
            }
    }
}
```