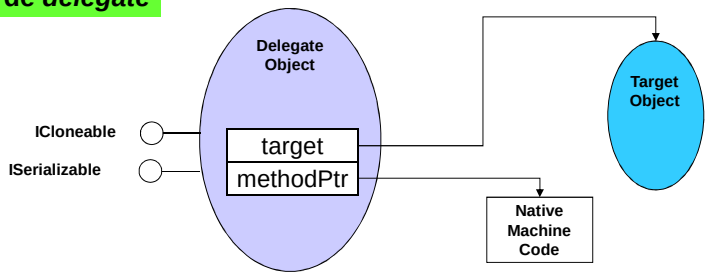
Invocação Assíncrona

A invocação assíncrona é realizada através de *delegates* para invocar métodos de forma não bloqueante (*asynchronous delegates*)

Conceito de *delegate*



***delegate* <*return type*> <nome do *delegate*> ([parâmetros]);**

Exemplo de utilização de *delegates*

```
public class Calc {
   public int Add(int a, int b) {…}
   // . . .
   public int Mult(int a, int b) {…}
}
```

Chamada indirecta de métodos através de um *delegate*

```
// declaração do tipo delegate compatível com métodos com
// dois argumentos inteiros e que devolvem um inteiro
public delegate int  DelBinaryOp(int x, int y);

Calc  mycalc = new Calc();
// fazer Bind para um objecto da classe Calc no método Add()
DelBinaryOp del = new DelBinaryOp(mycalc.Add);
. . .
if (del != null) del(12, 3);  // invoca o método mycalc.Add(12,3)
del = new DelBinaryOp(mycalc.Mult);
if (del != null) del(12, 3);  // invoca o método mycalc.Mult(12,3)
```

Asynchronous delegates

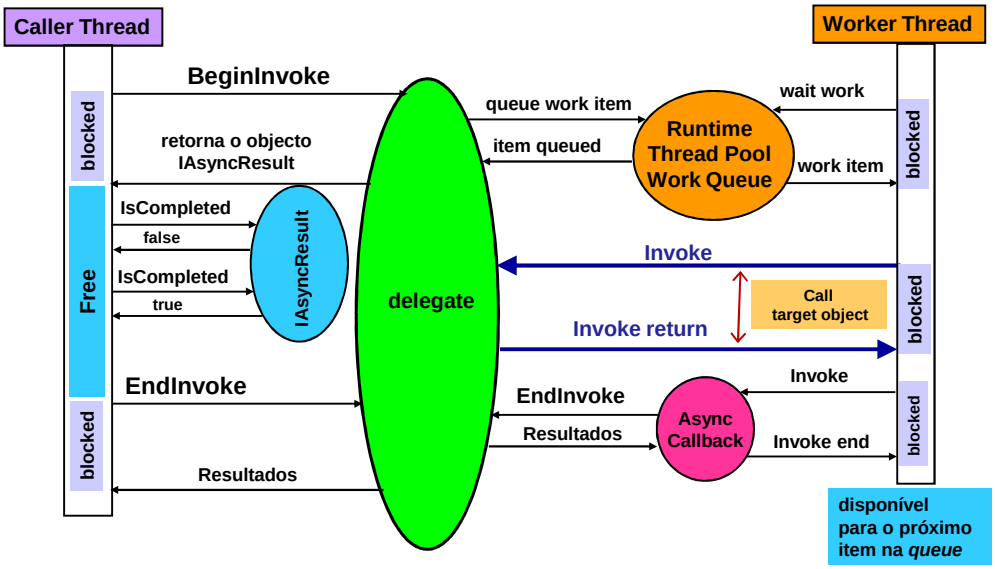
Para cada *delegate* o compilador gera os seguintes métodos:

- IAsyncResult BeginInvoke(int a, int b, AsynCallback cb, object state)
  - cb – permite definir outro *delegate* (*AsyncCallback*) que executará um método no objecto que realizou o *BeginInvoke* quando a operação estiver concluída. Se não se usar *callbacks* colocar null;
  - state – objecto que será passado ao *callback* em (IAsyncResult)ar.AsyncState. Deve usar-se *null* quando não se usa *Callbacks*.
- <return type> EndInvoke(IAsyncResult ar)
  - ar – objecto que foi retornado no *BeginInvoke*

Exemplo

```
public delegate int  DelBinaryOp(int x, int y);
Calc  mycalc = new Calc();
// fazer Bind para um objecto da classe Calc no método Add()
DelBinaryOp del = new DelBinaryOp(mycalc.Add);

IAsyncResult ar = del.BeginInvoke(12, 3, null, null);
// Fazer outras acções
int res = del.EndInvoke(ar);  //obter retorno do método
```


Modelo de execução do *BeginInvoke*



Interface

```
namespace IRemObject
{
   public interface INumber
   {
      void setValue(int newvalue);
      int getValue();
      int add(int a, int b);
   }
}
```

Exemplo: ***AsynchronousCalls.zip***

Server

```
class RemoteNumber : MarshalByRefObject, INumber   {

      private int number;

      public RemoteNumber() { Console.WriteLine("RemoteNumber.Constructor"); number = 0; }

      public void setValue(int newvalue)    {
        // Simulação de processamento lento
         Console.WriteLine("waiting 5 segundos antes de alterar valor");
         Thread.Sleep(5 * 1000);
         number = newvalue;
      }
      public int getValue()      {
         Console.WriteLine("RemoteNumber.getValue(): valor corrente {0}", number);
         return number;
      }
      public int add(int a, int b)   {
         Console.WriteLine("waiting 5 segundos antes de alterar valor");
         Thread.Sleep(5 * 1000);
         return a + b;
      }
   }
```

Server (cont.)

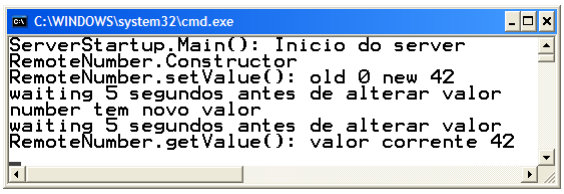
```
. . .
class ServerStartup
  {
     static void Main()
     {
        Console.WriteLine("ServerStartup.Main(): Inicio do server");

        HttpChannel ch = new HttpChannel(1234);
        ChannelServices.RegisterChannel(ch,false);

        RemotingConfiguration.RegisterWellKnownServiceType(
           typeof(RemoteNumber),
           "RemoteNumber.soap",
           WellKnownObjectMode.Singleton);

        Console.ReadLine();
     }

  }
```


Cliente com invocação Síncrona (*SynchronousCall*)

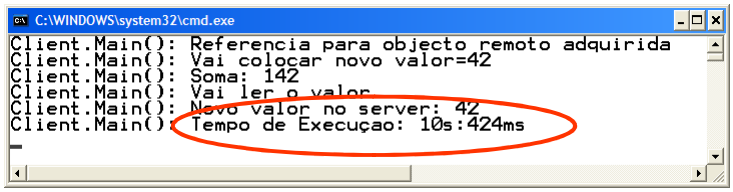
```
static void Main()   {

   HttpChannel ch = new HttpChannel(0);  ChannelServices.RegisterChannel(ch,false);
   INumber robj = (INumber)Activator.GetObject(
                                          typeof(INumber),
                                          "http://localhost:1234/RemoteNumber.soap");
   Console.WriteLine("Client.Main(): Vai colocar novo valor=42");

   DateTime start = System.DateTime.Now;
   robj.setValue(42);
   int soma = robj.add(42, 100);
   Console.WriteLine("Client.Main(): Soma: {0}", soma);
   Console.WriteLine("Client.Main(): Vai ler o valor");
   int tmp = robj.getValue();
   Console.WriteLine("Client.Main(): Novo valor no server: {0}", tmp);
   DateTime end = System.DateTime.Now;
   TimeSpan texec = end.Subtract(start);
   Console.WriteLine("Client.Main(): Tempo de Execução: {0}s:{1}ms",
                                                   texec.Seconds, texec.Milliseconds);
   Console.ReadLine();
}
```

Execução Síncrona

Server



Cliente



Cliente com invocação Assíncrona (*AsynchronousCall*)

```
class Client  {

      delegate void DelSetValue(int newvalue);
      delegate int DelAdd(int a, int b);

      static void Main()  {

         HttpChannel ch = new HttpChannel();  ChannelServices.RegisterChannel(ch,false);
         INumber robj = (INumber)Activator.GetObject(
                                                typeof(INumber),
                                                 "http://localhost:1234/RemoteNumber.soap");
         DateTime start = System.DateTime.Now;
         Console.WriteLine("Client.Main(): Vai colocar novo valor=42");

         // Invocação Assíncrona do método setValue()
         DelSetValue svdel = new DelSetValue(robj.setValue);
         IAsyncResult svasyncres = svdel.BeginInvoke(42, null, null);
         // Invocação Assíncrona do método add()
         DelAdd adddel = new DelAdd(robj.add);
         IAsyncResult addsyncres = adddel.BeginInvoke(42,100,null, null);
         // Continua a realizar Acções
           . . .
```

Cliente assíncrono (cont.)

```
. . .
    // Obter os resultados da invocação assíncrona
        Console.WriteLine("Client.Main(): Obter resultado do SetValue()");
        svdel.EndInvoke(svasyncres); // setValue devolve void
        Console.WriteLine("Client.Main(): Obter resultado de add()");
        int soma = adddel.EndInvoke(addsyncres);
        Console.WriteLine("Client.Main(): soma: {0}", soma);

        Console.WriteLine("Client.Main(): Vai ler o valor");
        int tmp = robj.getValue();
        Console.WriteLine("Client.Main(): Novo valor no server: {0}", tmp);

        DateTime end = System.DateTime.Now;
        TimeSpan texec = end.Subtract(start);
        Console.WriteLine("Client.Main(): Tempo de Execução: {0}s:{1}ms",
                                                    texec.Seconds, texec.Milliseconds);


        Console.ReadLine();
    }
}
```


Execução Assíncrona

Server

```
C:\WINDOWS\system32\cmd.exe
ServerStartup.Main(): Inicio do server
RemoteNumber.Constructor
RemoteNumber.setValue(): old 0 new 42
waiting 5 segundos antes de alterar valor
waiting 5 segundos antes de alterar valor
number tem novo valor
RemoteNumber.getValue(): valor corrente 42
```

Cliente

```
C:\WINDOWS\system32\cmd.exe
Client.Main(): Referencia para objecto remoto adquirida
Client.Main(): Vai colocar novo valor=42
Client.Main(): Obter resultado do SetValue()
Client.Main(): Obter resultado de add()
Client.Main(): soma: 142
Client.Main(): Vai ler o valor
Client.Main(): Novo valor no server: 42
Client.Main(): Tempo de Execuçao: 6s:168ms
```

Cliente com *Callback* (*ClientWithCallBack*)

```
class Client   {

      public static DateTime start;
      delegate void DelSetValue(int newvalue);

      // Async Callback:  Executado  por outra Thread quando terminar a chamada
      public static void setValueCompleted(IAsyncResult ar)   {
         Console.WriteLine("chamada do callback");
         string estado = (string)ar.AsyncState;
         Console.WriteLine("estado: " + estado);
         AsyncResult ar2 = (AsyncResult)ar;
         DelSetValue del = (DelSetValue)ar2.AsyncDelegate;
          try {
               del.EndInvoke(ar2);
          } catch(Exception ex)  {
               Console.WriteLine("Ocorreu exception  na chamada do método");
           }
         DateTime end = System.DateTime.Now;
         TimeSpan texec = end.Subtract(start);
         Console.WriteLine("Client.Main(): Tempo de Execução: {0}s:{1}ms",
                                                          texec.Seconds, texec.Milliseconds);
      }
   . . .
```

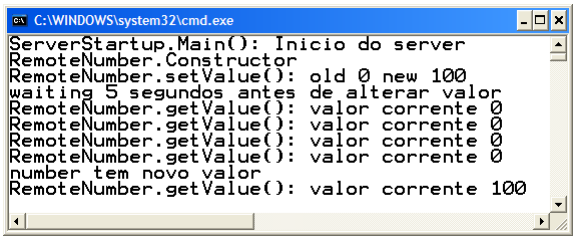

Cliente com *callback* (cont.)

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

         HttpChannel ch = new HttpChannel();
         ChannelServices.RegisterChannel(ch,false);
         INumber robj = (INumber)Activator.GetObject(
                                              typeof(INumber),
                                              "http://localhost:1234/RemoteNumber.soap");
         Console.WriteLine("Client.Main(): Vai colocar novo valor=100");
         // Invocação Assíncrona do método SetValue
         DelSetValue svdel = new DelSetValue(robj.setValue);
         AsyncCallback mycallback = new AsyncCallback(setValueCompleted);
         start = System.DateTime.Now;
         IAsyncResult svasyncres = svdel.BeginInvoke(100, mycallback, "set value com 100");
         while (!svasyncres.IsCompleted)  {
             // O cliente continua a fazer trabalho até a operação terminar
             Console.WriteLine("espera fim operação");
             Console.WriteLine("novo valor= {0}", robj.getValue()); // chamada síncrona
             Thread.Sleep(1000);
         }
         Console.WriteLine("novo valor= {0}", robj.getValue());
   }
}
```

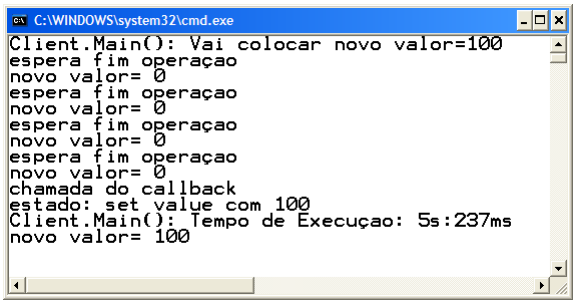
estado

Execução Assíncrona com *Callback*

Server



Cliente



Callbacks e actualização de controlos em *Windows Forms*

- Quando criamos um objecto gráfico (*Windows Forms*), este fica associado à *Thread* que criou o objecto. Quando outra *Thread* diferente pretende actualizar o *Form* é necessário garantir *thread safe* no objecto *Form*.
- Assim, a classe *Form* tem a propriedade *thread safe*, ***IsInvokeRequired,*** que permite saber se é necessário invocar o método ***Invoke,*** também *thread safe*, com um *delegate* para actualizar o *Form* de forma *thread safe*.
- No caso das chamadas assíncronas com *CallBack*, pode ser necessário actualizar os controlos existentes no *Form*, pelo que deve ser seguido o padrão atrás descrito ou outro equivalente.

Padrão elegante para actualizar um Controlo *richTextBox*

```
private delegate void PutTextDel(string txt);
public void PutTextByCallBack(string txt)   {
    if (this.InvokeRequired)     {
        PutTextDel del = new PutTextDel(this.PutTextByCallBack);
        object[] args = new object[] { txt };
         this.Invoke(del, args);
     }  else  richTextBox.AppendText(txt + "\r\n");
}
```

A classe ***BackgroundWorker*** introduzida a partir do *framework 2.0* permite executar tarefas noutra *thread* e actualizar o *Form* de forma *thread safe (Ver MSDN)*