PROGRAMAR

REVISTA PORTUGUESA DE PROGRAMAÇÃO • WWW.PORTUGAL-A-PROGRAMAR.ORG

EDIÇÃO #33- FEVEREIRO 2012

ISSN 1647-0710

KINECT HACK

COLUNAS

AMBIENTES PRODUTIVOS **CORE DUMP**

ITERATORS **VISUAL (NOT) BASIC**

DISPOSABLE STRUCTS **ENIGMAS DO C#**

A PROGRAMAR

GERAÇÃO DE NÚMEROS ALEATÓRIOS (PARTE 3)

PROGRAMAÇÃO ORIENTADA A OBJECTOS EM PASCAL

HERANÇA EM JAVASCRIPT

COMUNIDADES

DESENVOLVIMENTO EM SHAREPOINT 2010 (PARTE 2) **SHAREPOINTPT**

SQL AZURE FEDERATIONS NA PRÁTICA **AZUREPT**

BIZTALK SERVER PRINCÍPIOS BÁSICOS DOS MAPAS (PARTE 2) **NETPONTO**

EDITORIAL

Contacto
revistaprogramar@portugal-a-programar.org

Website
http://www.revista-programar.info

ISSN
1 647-071 0

Reinventado

Há 31 anos aparecia o primeiro computador a receber o acrónimo *Personal Computer*, ou PC, no ano de 1981, lançado pela IBM. Muito antes do Computador Pessoal, outros computadores já existiam, e foram muitas as inovações que fizeram do computador o "PC". Em 1968 Douglas Engelbart, inventou um dispositivo apontador, que agora é conhecido como "rato", e faz parte dos PC's como os conhecemos, e usamos todos os dias, para todas das tarefas que realizamos num PC.

O Altair 8800, deliciou o mundo e os leitores da *Popular Electronics*, e apesar de pouco mais fazer do que ligar e desligar algumas luzes foi o impulso para toda uma evolução. Em volta dessa máquina um grupo de estudantes e hobistas interessados fundaram o *Homebrew Computer Club*, que influenciou o mundo da tecnologia como o conhecemos. A essas maquinas seguiram outras, sempre em constante evolução até aparecer o que conhecemos como Computador Pessoal, com teclado, rato, monitor, unidade de disquetes e disco rígido.

Seguindo a previsão de Gordon E. Moore e até ultrapassando a mesma, os pc's evoluíram mas sempre tiveram muita coisa em comum com os PC's do principio, e o conceito de PC. Ao longo dos anos assistimos a toda uma evolução sem uma ruptura com o passado dos PC's.

Neste momento os tablets abrem todo um novo "mundo", quase fazendo acreditar que vamos assistir ao fim dos PC's como os conhecemos, para dar lugar a uma nova maquina que nos acompanha para todo o lado, onde escrevemos ou até ditamos o que pretendemos, sem teclado, sem rato, sem drive de disquetes, mas acopulável a uma *Docking Station*, com teclado, e aos poucos o PC como a maioria de nós o conheceu, acabará por dar lugar ao novo gadget que encherá o nosso espaço, que não lhe conheço ainda o nome pois não é um tradicional PC, nem um tablet, muito menos um Ultra-Mobile PC. A única coisa que parece certa, é que no futuro o PC como o conhecemos, deixará o engenho de Engelbart, substituindo-o por uma tela táctil, os cabos deixarão de povoar os espaços de trabalho e para onde quer que formos o computador "verdadeiramente pessoal" irá connosco.

Da minha parte resta-me agradecer a oportunidade que me foi dada, e acreditar que a Revista PROGRAMAR irá mais longe e mais além, pois esta é a minha primeira edição como coordenador. É um legado difícil de seguir, o que o António Silva deixou, mas estou convicto que com toda a equipa levaremos a Revista PROGRAMAR para o "infinito e mais além", tal como ele o desejou na sua ultima edição como coordenador. Estou convicto que estaremos à altura dos novos desafios neste mundo em mudança e que traremos a cada edição novos artigos, sempre para agradar aos nossos tão estimados leitores.

António Santos
<antonio.santos@revista-programar.info>

ÍNDICE

TEMA DE CAPA

A PROGRAMAR

COLUNAS

COMUNIDADES

EVENTOS

Semana Informática do IST
Saiba mais informações sobre este evento www.sinfo.ist.utl.pt e na pagina 59 desta edição

EVENTOS

04 Fev 2012	6° Evento da Comunidade AzurePT
08 Fev 2012	Terceiro Evento da Comunidade HTML5PT
11 Fev 2012	4ª Reunião Presencial da Comunidade NetPonto - Coimbra
18 Fev 2012	1° Evento da Comunidade AzurePT - Porto
18 Fev 2012	27ª Reunião Presencial da Comunidade NetPonto - Lisboa
22 Fev 2012	XIX encontro da comunidade SQLPort,
17 Mar 2012	SQL Pass Saturday Portugal #115
24 Mar 2012	28ª Reunião Presencial da Comunidade NetPonto - Lisboa

Para mais informações/eventos: http://bit.ly/PAP_Eventos

NOTICIAS

Google apresenta alternativa ao JavaScript

A Google apresentou hoje uma nova linguagem de programação para desenvolvimento de aplicações Web, que visa afirmar-se como uma alternativa ao popular JavaScript. Os interessados podem começar a experimentar a novidade ainda em fase de testes.

Dart é o nome da proposta da gigante das pesquisas, dada a conhecer esta segunda-feira, numa conferência na Dinamarca e detalhada numa mensagem publicada no blog da companhia.

Segundo avançou o programador Lars Bak, a nova linguagem destina-se a todo o tipo de escalas, desde os projectos pequenos e não estruturados aos grandes e complexos. Os objectivos enunciados passam pela criação de uma "linguagem estruturada mas simples" para a programação Web.

Oferecer uma alternativa com a qual os programadores se sintam familiarizados e que se apresente como intuitiva, sendo, por isso, fácil de aprender e "assegurar que a nova linguagem garante alta performance em todos os modernos *browsers* e ambientes, desde os dispositivos portáteis à execução ao nível do servidor", explica o responsável online.

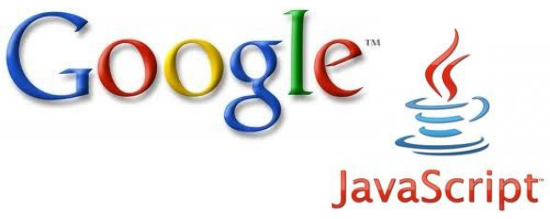
A acompanhar a apresentação oficial foi colocado à disposição dos programadores um site dedicado à nova linguagem, onde se incluem ferramentas para criação de programas em Dart, amostras de código e tutoriais, especificações da linguagem e fóruns de discussão.

Os programas criados em Dart poderão ser corridos numa máquina virtual dedicada ou recorrendo a um compilador que traduza o código para JavaScript para que este possa ser lido pelos browsers que não suportam a nova linguagem.

O browser da Google deverá servir como um veículo para levar a novidade à Web. Segundo adiantou Lars Bak, a máquina virtual de Dart ainda não está integrada no Chrome, mas os responsáveis planeiam explorar essa possibilidade.

Escrito ao abrigo do novo Acordo Ortográfico

Fonte: Tek Sapo



"As pessoas vão ter de viver com o que escreveram e partilharam hoje daqui a 20 anos"

O Conselheiro de Informação para a Inovação e Tecnologia de Hillary Clinton considera que a longo prazo a memória indelével da Internet vai ter implicações na forma como as pessoas socializam. Porque o que se diz na Internet não desaparece.

Numa conversa online onde o SAPO Notícias esteve presente como participante, o Conselheiro de Informação para a Inovação e Tecnologia da secretária de Estado norte-americana defendeu hoje que as redes sociais podem gerar complicações no futuro. “As pessoas vão ter de viver com o que escreveram e partilharam hoje daqui a 20 anos, porque a memória da Internet não desaparece”, explicou. “Se partilhamos uma frase ou imagem, esses conteúdos vão estar na rede daqui a vários anos.”

O especialista em Redes Sociais avançou que é preciso estar “atento” e ser responsável pelo que se diz online, para evitar "conflitos" no futuro. “As crianças de oito anos podem daqui a 10 anos ser confrontadas com aquilo que disseram no passado”, indicou.

Durante a conferência, o perito destacou a importância das redes sociais no processo democrático. “A partir do momento que toda a gente consegue com o seu telemóvel registar imagens ou vídeos e partilhá-los na Internet, isso significa que temos mais acesso a informações e mais democracia”, indicou, dando o exemplo da Primavera Árabe.

A Internet "não pode ser controlada"

Relativamente à cibersegurança, Alec Ross destacou que “as redes sociais podem ter um efeito nuclear”. “Estas ferramentas podem ser usadas para boas ou más causas e não é possível controlar isso”, explicou. Para o especialista, a melhor forma de quebrar a partilha de informações falsas é “repor imediatamente verdade”, porque a Internet “não deve, nem pode ser controlada.”

Alec Ross salientou que as redes socias têm um papel fundamental no desenvolvimento económico. “Quanto mais ligada estiver a população entre si, melhor será o empreendedorismo e maior será o crescimento económico”, asseverou. O norte-americano de 40 anos acrescentou ainda que a estratégia dos Estados Unidas nas redes sociais é de "ter voz e ouvir”. O especialista referiu-se às redes sociais como a forma de comunicação por excelência do Século XXI e desvalorizou a ideia de que possam ser mais um veículo de propaganda política.

“A propaganda política não funciona nas redes sociais porque há centenas de fontes e as pessoas que estão na internet têm cada vez mais literacia e formação”, concluiu.

Nuno de Noronha Xavier

Este artigo foi escrito ao abrigo do novo acordo ortográfico.

Fonte: Tek Sapo

TEMA DA CAPA

Kinect Hack

Kinect Hack

O Kinect é um dispositivo para a consola Microsoft Xbox 360, que através da detecção de movimentos e gestos realizados pelo utilizador ou até através do reconhecimento por voz, permite que o leitor consiga interagir com a consola e com os jogos de vídeo que suportam este hardware de uma forma natural, substituindo assim os controlos tradicionais que as consolas disponibilizam.

Foi anunciado pela primeira vez em Junho de 2009 conferência E3 sob o nome de código "Natal". Colocado à venda em Novembro de 2010, o Kinect garantiu um lugar na história, em apenas 60 dias foram comercializados 8 milhões de unidades.

À primeira vista, o Kinect assemelha-se a uma webcam com um formato não muito convencional, mas na verdade, trata-se de muito mais do que isso. É composto por uma câmara RGB, um projector de raios IR que em conjunto com uma segunda câmara monocromática de IR constrói a informação em 3D com os meta-dados e um microfone vectorial (composto por quatro microfones colocados estrategicamente no dispositivo), que o tornam capaz de isolar e identificar as vozes dos vários jogadores e, para finalizar, possui um pequeno motor que lhe permite ajustar automaticamente o seu ângulo de visão para detectar todos os jogadores presentes.

As características técnicas que o Kinect apresenta a um preço acessível, suscitaram imediatamente muito interesse a uma comunidade que reconheceu de imediato as potencialidades que um dispositivo como este proporciona.

Sendo assim, ainda durante o mês de lançamento, uma empresa ofereceu uma recompensa a quem conseguisse desenvolver um *driver open-source* que permitisse o Kinect ser utilizado foram do seu âmbito, ou seja, fora do ecossistema da Xbox 360, para que o dispositivo pudesse ser utilizado para outros fins que não o entretenimento.

Apenas alguns dias após o seu lançamento, foi disponibilizado um driver para o Sistema Operativo Linux (Libfreenect) que permitia a utilização tanto da câmara RGB como dos sensores de profundidade 3-D. Em Dezembro do mesmo ano, a empresa PrimeSense, que é proprietária de tecnologia semelhante à apresentada pelo Kinect, lançou a sua própria versão do driver *Open-Source* para Kinect, bem como um Middleware dotado de um conjunto de funcionalidades até então apenas disponíveis na consola da Microsoft, como a capacidade de detecção de gestos e *tracking* de indivíduos.

Toda esta comoção gerada à volta do Kinect e a sua utilização fora do seu ambiente natural, levou à criação da organização sem fins lucrativos OpenNI (*Open Natural Interaction*) e em última instância, pressionou a Microsoft para relevar/alterar os seus planos para o Kinect e disponibilizar o Microsoft Kinect SDK.

Assim, o leitor tem ao seu dispor a escolha entre duas soluções para desenvolver as suas aplicações com o Kinect, a Framework OpenNI ou o Kinect SDK da Microsoft, actualmente em versão Beta, no entanto já foi anunciado que a versão comercial vai estar disponível em Fevereiro de 2012, acompanhada de uma nova versão de hardware, suportado oficialmente Microsoft.

É importante salientar ao leitor que, antes de enveredar por qualquer uma destas hipóteses, é indispensável verificar os termos e condições de garantia do dispositivo.

Kinect Demo

É importante que o leitor tenha em mente que as escolhas executadas para desenvolver este artigo não são as únicas ao seu dispor, a Framework OpenNI é suportada em ambientes Windows, Linux e Mac OSX com as linguagens C++, C# e Java, Python, ActionScript, entre outras.

O objectivo de esta aplicação é demonstrar ao leitor as potencialidades do Kinect e como utilizar este potencial para criar aplicações inovadoras.

Figura 1 – Aplicação a desenvolver no decorrer do artigo

O projecto é desenvolvido com recurso a C#/WPF com a utilização da Framework OpenNI como base de suporte do Kinect.

Para simplificar e não sujeitar o leitor a um grande conjunto de informação sobre OpenNI, pois esse não é o objectivo a

que se propõe o artigo, não vão ser incluídos demasiados detalhes sobre esta Framework, no entanto, é aconselhada a visita o website http://www.OpenNI.org, onde poderá encontrar uma vasta documentação sobre o tema.

Arquitectura

Esta aplicação assenta sobre uma solução que é composta por um conjunto de 8 projectos em C#. Os principais motivos que motivaram a essa opção, foram em primeiro lugar, uma gestão cuidada da organização, divisão de responsabilidades e garantir que a arquitectura da aplicação permite que a informação chega ao leitor de forma simples e clara, em segundo lugar, dado que o código desenvolvido se tornou demasiado extenso para ser exposto por completo no artigo, optou-se assim por disponibilizar o código fonte no endereço http://kinecthack.codeplex.com/ onde leitor pode executar o seu download e, utilizar livremente, o exemplo demonstrado como ponto de partida para uma investigação mais aprofundada sobre o tema.

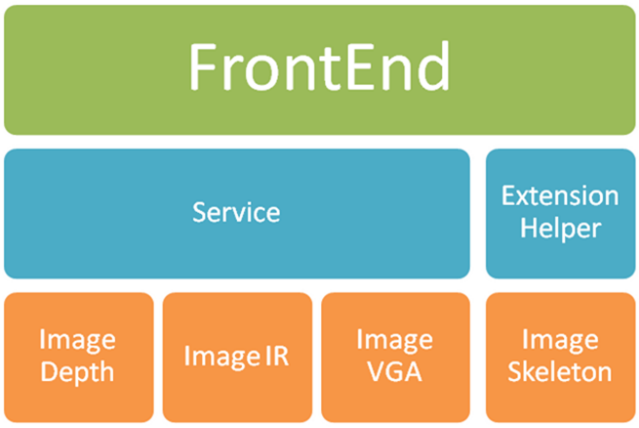


Figura 2 – Arquitectura da aplicação

A camada de *FrontEnd* é uma aplicação WPF onde está definido o *User Interface* da aplicação e as acções que são despoletadas mediante a interacção do utilizador com a aplicação.

Quando o utilizador executa uma operação, como ligar a câmara VGA, a camada de *FrontEnd*, através da camada de *Service*, fornece uma imagem da fonte VGA do Kinect (*Image* VGA). Esta lógica da arquitectura aplica-se a todas as fontes de informação ilustradas na Figura 2 identificadas com a cor laranja.

Assim, a lógica da aplicação está na camada *Service* e os dados são processados na camada onde a fonte de informação que foi pedida.

Com esta arquitectura é possível, por exemplo, sobrepor a informação da fonte VGA e *Skeleton* e apresenta-la à camada de *FrontEnd*.

Existem três fontes de dados que é possível explorar: RGB, Depth e IR, todas elas disponibilizam a informação necessária para construir uma imagem e os seus metadados.

FrontEnd

Desenvolvido em WPF, o FrontEnd é o interface entre o Kinect e o utilizador, onde é possível alterar as fontes de informação do Kinect e assim controlar as suas funcionalidades.

```
public partial class MainWindow : Window
 {
   public enum CameraStream
    {
      VGA,
      IR,
      Depth,
      Skl
     }

   private bool isRecording = false;
   private bool isPlaying = false;

   private PapKinect service;
   private CameraStream CameraType;

   public MainWindow()
    {
     InitializeComponent();
    }

    private void btnDepth_Click(object sender,
                                RoutedEventArgs e)
     {
       try
        {
         if (service != null)
          {
            service.StopServices();
          }
         CameraType = CameraStream.Depth;
         service = new PapKinect
               (PapKinect.KinectServiceType.Depth);
         service.KinectDataUpdated += new
         EventHandler(service_KinectDataUpdated);
         service.ProcessDepth();
        }
       catch (Exception ex)
       {
        MessageBox.Show(ex.Message);
       }

     }

   private void btnIR_Click(object sender,
                            RoutedEventArgs e)
   {
    try
       {
        if (service != null)
         {
          service.StopServices();
         }
        CameraType = CameraStream.IR;
        service = new PapKinect
              (PapKinect.KinectServiceType.IR);
        service.KinectDataUpdated += new
        EventHandler(service_KinectDataUpdated);
        service.ProcessInfraRed();
       }
```

```
       catch (Exception ex)
        {
         MessageBox.Show(ex.Message);
        }

      }

private void btnVga_Click(object sender,
                    RoutedEventArgs e)
 {
  try
   {
    if (service != null)
     {
      service.StopServices();
     }
    CameraType = CameraStream.VGA;
    service = new PapKinect
    (PapKinect.KinectServiceType.VGA);
    service.KinectDataUpdated += new EventHandler
    (service_KinectDataUpdated);
    service.ProcessVGA();
   }
  catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
   }
 }

private void btnSkeleton_Click(object sender,
                           RoutedEventArgs e)
 {
  try
   {
    if (service != null)
     {
      service.StopServices();
     }
    CameraType = CameraStream.Skl;
    service = new PapKinect
    (PapKinect.KinectServiceType.Skl);
    service.KinectDataUpdated += new EventHandler
    (service_KinectDataUpdated);
    service.ProcessSkeleton();
   }
  catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
   }
  }

private void btnRecorder_Click(object sender,
                           RoutedEventArgs e)
   {
     try
      {
       if (!isRecording)
        {
         string filename = string.Empty;

         MessageBox.Show("Click again to stop
                                 recording.");

         Microsoft.Win32.SaveFileDialog dlg = new
         Microsoft.Win32.SaveFileDialog();
         dlg.DefaultExt = ".oni";
         dlg.Filter = "OpenNI Recordings (.oni)
                                       |*.oni";
          if (dlg.ShowDialog() == true)
           {
            filename = dlg.FileName;
           }

          isRecording = true;
           if (service == null)
   {
    CameraType = CameraStream.Depth;
    service = new PapKinect
    (PapKinect.KinectServiceType.Depth);
    service.KinectDataUpdated += new EventHandler
    (service_KinectDataUpdated);
    }
   service.StartDepthRecording(filename);
  }
  else
  {
   isRecording = false;
   service.StopRecording();
   MessageBox.Show("Stopped recording.");
  }
 }
 catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
  }

 }

private void btnPlayer_Click(object sender,
                           RoutedEventArgs e)
   {
    try
     {
      if (!isPlaying)
       {
        MessageBox.Show("Click again to stop
                                    player.");
        string filename = string.Empty;
        Microsoft.Win32.OpenFileDialog dlg = new
                 Microsoft.Win32.OpenFileDialog();
        dlg.DefaultExt = ".oni";
        dlg.Filter = "OpenNI Recordings (.oni)
                                       |*.oni";
         if (dlg.ShowDialog() == true)
          {
           filename = dlg.FileName;
          }
         else
          return;

          isPlaying = true;
          if (service == null)
           {
            service = new PapKinect
            (PapKinect.KinectServiceType.Depth);
            service.PlayRecording(filename);
            service.GetNewPlayerFrame += new
             EventHandler
                    (service_GetNewPlayerFrame);
           }
          }
         else
          {
           isPlaying = false;
           service.StopPlayer();
           service.GetNewPlayerFrame -=
           service_GetNewPlayerFrame;
           MessageBox.Show("Player stoppped.");
          }
         }
        catch (Exception ex)
        {

         MessageBox.Show(ex.Message);

        }

       }

void service_GetNewPlayerFrame(object sender,
```

```
                                    EventArgs e)
   {
    if (service!=null)
      {

       kinectData.Source =
       service.GetPlayerDepthFrame();

      }

     }

void service_KinectDataUpdated(object sender,
                                    EventArgs e)
 {

  switch (CameraType)

  {

   case CameraStream.VGA:
    kinectData.Source = service.GetVGAFrame();
    break;

   case CameraStream.IR:
    kinectData.Source = service.GetInfraRedFrame();
    break;

   case CameraStream.Depth:
    kinectData.Source = service.GetDepthFrame();
    break;

   case CameraStream.Skl:
    kinectData.Source = service.GetSkeletonFrame();
    break;

  }

 }

}
```

Existe uma instância da classe PapKinect, que permite aceder à camada *Service*. Esta camada disponibiliza um conjunto de serviços ao *FrontEnd*, mas o mais importante é sem dúvida o evento *KinenctDataUpdated*, que é disparado sempre que existe uma nova *frame* para ser apresentada, e que mediante o tipo de fonte de informação que o utilizador escolhe para visualizar, vai executar um pedido ao método correspondente e apresentar a imagem no controlo *KinectData*.

O evento *GetNewPlayerFrame* é utilizado apenas quando a aplicação está a reproduzir um vídeo que foi gravado através de esta aplicação, e como o próprio nome indica, quando é chamado, o evento executa um pedido à camada *Service* para esta lhe fornecer um novo *frame* de vídeo.

Service

Como o próprio nome indica, trata-se de uma camada de serviço, o seu objectivo é cumprir com os pedidos que lhe são requisitados pelo *FrontEnd*, mantendo contidas dentro de si, as regras de lógica e de negócio da aplicação, criando uma clara separação de responsabilidades entre as várias camadas.

Existem nesta camada um conjunto de métodos públicos, que são os serviços disponibilizados. Esses serviços assentam num conjunto de variáveis que são instanciadas consoante o serviço que é requisitado, no entanto, estas variáveis estão agrupadas de uma forma lógica.

```
public enum KinectServiceType
{
    VGA,
    IR,
    Depth,
    Skl
}

private KinectServiceType ServiceType;
public event EventHandler KinectDataUpdated;
public event EventHandler GetNewPlayerFrame;
```

O primeiro grupo identifica uma enumeração que é utilizada no construtor da classe Service e serve para identificar qual é o tipo de fonte de informação que o serviço vai utilizar para executar o seu trabalho, de seguida temos a definição dos eventos que a camada Service vai disponibilizar e que já foram explicados anteriormente.

```
private Context ctx;
private DepthGenerator depthGenerator;
private IRGenerator irGenerator;
private ImageGenerator vgaGenerator;
```

O segundo grupo identifica as variáveis que pertencem a classes da Framework OpenNI, a classe Context é uma classe que podemos considerar como base de acesso e configuração do Kinect, é através de esta classe que são carregadas as configurações do dispositivo e é esta a classe que gere todas as fontes de informação o leitor tem ao seu dispor.

Cada fonte de informação disponibilizada pelo OpenNI através do Kinect, tem o seu próprio gerador, a classe DepthGenerator identifica a fonte de informação gerada pelo projector IR e câmara monocromática, a classe IR identifica a fonte de infra-vermelhos e a classe ImageGenerator identifica a fonte de VGA.

```
private PapDepth serviceDepth;
private PapIR serviceIR;
private PapVGA serviceVGA;
private PapSkeleton serviceSkl;
private PapRecorder serviceRecorder;

private Bitmap depthBitmap;
private WriteableBitmap vgaBitmap;
private WriteableBitmap irBitmap;
```

De seguida são identificadas as variáveis que pertencem às classes que têm como responsabilidade processar a informação de contexto e do gerador correspondente, bem como os Bitmaps que vão conter as imagens depois de a informação cedida pelos geradores ser processada.

```
public void ProcessDepth()
{
    depthGenerator = ctx.FindExistingNode
       (NodeType.Depth) as DepthGenerator;
    serviceDepth = new PapDepth();
    depthBitmap = serviceDepth.StartDataProcessing
                          (ctx, depthGenerator);
}
```

A lógica de esta camada está dividida por dois grandes grupos, os métodos de inicialização e os métodos de execução. Sempre que o leitor selecciona uma funcionalidade no FrontEnd, é inicializado na camada de Service o processo correspondente, esse processo cria o contexto de execução e inicializa o gerador necessário, a partir de um ficheiro XML de configuração, para cumprir a tarefa requisitada, existindo por isso um método com o prefixo Process para cada uma das fontes de informação.

Seguindo a lógica anteriormente definida, existe um método com o prefixo Get para cada uma das fontes de informação, estes métodos são requisitados sempre que existem novos dados disponíveis no Kinect que necessitam de ser processados pela sua fonte de informação.

ImageDepth

É uma das camadas de processamento de fonte de informação, esta tem como objectivo processar a informação que o DepthGenerator fornece e transforma-la numa imagem.

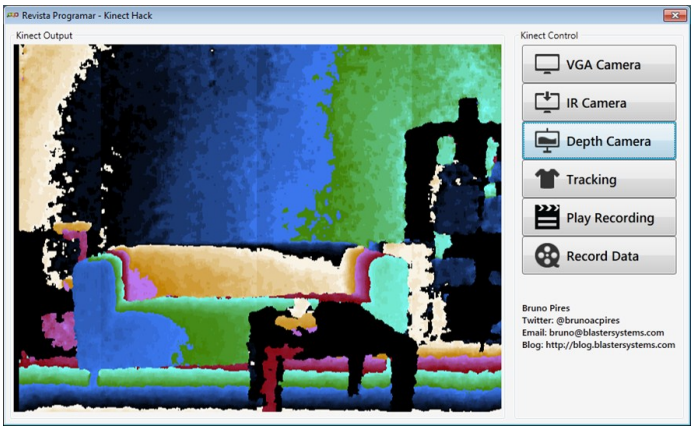


Figura 3 – Histograma 3D

```
public Bitmap StartDataProcessing(Context ctx,
DepthGenerator generator)
{
    if (generator != null)
    {
        var mapMode = generator.MapOutputMode;
        return new Bitmap(mapMode.XRes, map
        Mode.YRes,
        Sytem.Drawing.Imaging.PixelFormat._
        Format24bppRgb);
    }
    else
    {
        throw new Exception("There is no node
                  defined for Depth, verify OpenNI
                  config file.");
    }
}
```

Como na camada Service o contexto e gerador já foram inicializados, nesta camada apenas é inicializado o Bitmap que vai ser utilizado para guardar a informação que o gerador fornece.

```
public unsafe BitmapImage UpdateDepth
(DepthGenerator generator, Bitmap depthBitmap)
{
 try
  {
   var metadata = new DepthMetaData();
   var rect = new Rectangle(0, 0,
    depthBitmap.Width, depthBitmap.Height);
   var data = depthBitmap.LockBits(rect,
    System.Drawing.Imaging.ImageLockMode.WriteOnly,
    System.Drawing.Imaging.PixelFormat.
    Format24bppRgb);
    generator.GetMetaData(metadata);


    ushort* pDepth = (ushort*)
     generator.DepthMapPtr.ToPointer();

    for (int i = 0; i < metadata.YRes; i++)
     {
      byte* pDest = (byte*)data.Scan0.ToPointer() +
        i * data.Stride;

      for (int j = 0; j < metadata.XRes; j++,
       pDepth++, pDest += 3)
        {
         pDest[0] = (byte)(*pDepth >> 0);
         pDest[1] = (byte)(*pDepth >> 1);
         pDest[2] = (byte)(*pDepth >> 2);
        }
      }

     depthBitmap.UnlockBits(data);
     return depthBitmap.getBitmapImage();
   }
  catch (Exception ex)
   {
    throw ex;
   }
}
```

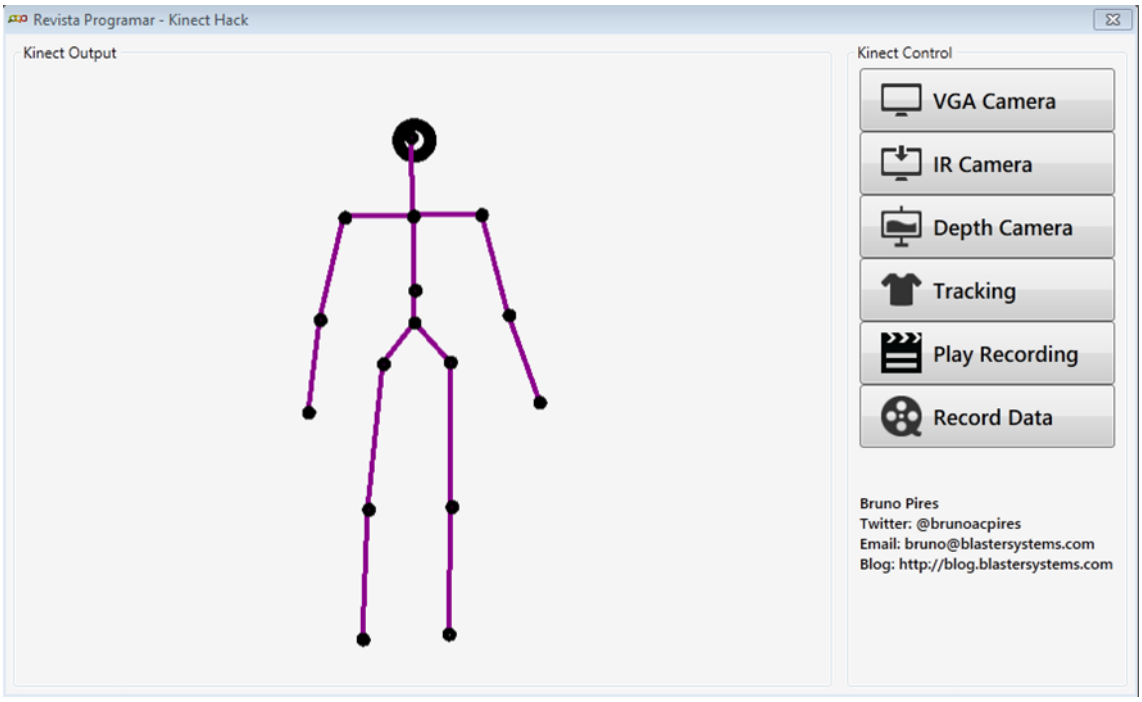
Quando o Kinect disponibiliza novos para processamento, o evento KinectDataUpdated é executado e é requisitado a esta camada que processe a informação fornecida pelo Kinect, que neste caso está contida no DepthGenerator.

De seguida é obtida a metadata contida no DepthGenerator e são percorridos todos os pixéis do Bitmap de destino, coluna a coluna, e é atribuída uma cor a cada pixel consoante a distância a que cada pixel identificado no metadata se encontra do Kinect e, assim, a imagem devolvida trata-se nada mais, nada menos do que um histograma 3D.

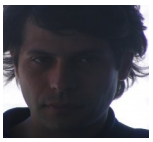
Notas finais

O que torna a *metadata* que o Kinect disponibiliza tão interessante é a capacidade que fornece ao leitor de identificar a distância a que se encontra cada um dos pixéis/objectos do sensor num sistema geométrico (com coordenadas x,y,z), tudo isto em conjunto com um algoritmo de detecção e tracking de pessoas e/ou gestos, como é disponibilizado pelo OpenNI Nite, permite revelar o verdadeiro potencial do Kinect ou dispositivos semelhantes como o Asus Xtion (também suportado pelo OpenNI).

Dado que o código escrito para este artigo é mais extenso do que o comum, e sendo um dos objectivos motivar o leitor e leva-lo a aprofundar os conhecimentos sobre este tema, bem como proporcionar-lhe uma explicação sobre os pontos principais do projecto e exemplificar a lógica da arquitectura, encontra-se disponível o código fonte de esta solução no endereço http://kinecthack.codeplex.com/.



AUTOR



Escrito por Bruno Pires

Exerce funções de consultor de IT na Novabase desde 2008, com experiência de maior relevo nas áreas da banca e televisão digital, onde ganhou competências nas mais várias tecnologias. Membro da Comunidade NetPonto (http://netponto.org) e autor do blog http://blog.blastersystems.com - Twitter: @brunoacpires

DIVULGA A REVISTA PROGRAMAR E GANHA UM KINECT!

Para poderes ganhar um Kinect só tens de divulgar a revista PROGRAMAR e enviar um email para o endereço concurso.kinect@revista-programar.info com o endereço da divulgação*.

*** REGULAMENTO/CONDIÇÕES DE PARTICIPAÇÃO**

A divulgação da revista pode ser feita em blogs/páginas pessoais e nas redes sociais (twitter, facebook, linkedin, etc), **devendo referir obrigatoriamente** "Revista PROGRAMAR nº 33" e o endereço "http://tiny.cc/ProgramarED33". Devem então enviar para o email concurso.kinetic@revista-programar.info o endereço com a participação, sendo **apenas aceite por utilizador**:

- um email de divulgação nas redes sociais
- um email de divulgação no blog/página pessoal

Assim, caso divulguem nos dois locais, terão duas inscrições para poder ganhar um Kinect.

A participação é válida até ao **dia 15 de Fevereiro de 2011** e o resultado do sorteio será divulgado no dia seguinte. Os moderadores da comunidade Portugal-a-Programar e staff da revista PROGRAMAR não podem participar nesta iniciativa. Todos os casos omissos serão resolvidos pelo staff da revista e não são passíveis de recurso.

A PROGRAMAR

Geração de números aleatórios (Parte 2)

Herança, em JavaScript

Programação Orientada aos Objectos em Pascal

GNA - GERAÇÃO DE NÚMEROS ALEATÓRIOS (Parte 3)

Nos artigos anteriores foram mostrados os métodos de geração de números pseudo-aleatórios meio do quadrado do meio e produto do meio. Em continuação será mostrado neste artigo o método RANDU.

INTRODUÇÃO

O algoritmo RANDU foi muito utilizado nas décadas de 1960 e 1970 para gerar valores “aleatórios” em computadores de grande porte (mainframe) da IBM.

Este é considerado por alguns como sendo um dos, senão, o pior método de geração de valores pseudo-aleatórios, segundo informado na WIKIPEDIA (2008), por gerar valores pares ou ímpares, mas nunca mistos. Se a semente fornecida for ímpar, todos os valores gerados são ímpares, se a semente fornecida for par, todos os valores gerados são pares. Assim é necessário ter cuidado com os valores gerados pelo método. No entanto, deve-se levar em consideração que se o método fosse tão ruim como alguns dizem, por qual motivo uma empresa com o porte da IBM o usaria por tantos anos?

O método RANDU opera com valores baixos e com valores de qualquer quantidade de dígitos, o que é uma vantagem sobre os métodos quadrado do meio e produto do meio. A faixa de operação de abrangência dos valores dependerá da capacidade do computador em uso.

Para obter valores pseudo-aleatórios com o método RANDU deve-se fazer uso da fórmula aritmética:

$$x_{i+1} = (2^{16}+3)x_i - 2^{31}\left\lfloor \frac{(2^{16}+3)x_i}{2^{31}} \right\rfloor$$

Onde xi é o valor inicial de uma semente e xi+1 é o valor calculado da próxima semente que iterativamente passará a ser o valor da próxima semente xi.

A operação de divisão encontra-se definida na fórmula dentro da função parte inteiraque pode ser representada como:

$\lceil x \rceil$ Parte inteira superior

$\lfloor x \rfloor$ Parte inteira inferior

Parte inteira inferior, também chamada chão, e parte inteira superior, também chamada teto apresentada por KNUTH (1999, p. 47).

A tabela seguinte apresenta a sequência de dez valores gerados por meio do método RANDU, a partir da semente 32534.

Iteração	Valor interagido	RANDU
0	32534	2132245826
1	2132245826	2055763910
2	2055763910	1734305618
3	1734305618	493893110
4	493893110	239509986
5	239509986	1286989222
6	1286989222	1271378162
7	1271378162	340333270
8	340333270	1337014402
9	1337014402	664119686

Apesar das críticas, esse método quando estendido a um ciclo iterativo de 1000 valores não gera valores “zerados”.

CONCLUSÃO

Neste artigo foi apresentado o método de geração de números pseudo-aleatórios meio do produto, sendo este mais eficiente que o método do meio do quadrado, pois demora mais a apresentar valores zerados.

BIBLIOGRAFIA

KNUTH, D. E. The Art of Computer Programming: Fundamental Algorithms. vol 1, 3. Ed, Massachusetts: Addison-Wesley, 1999.

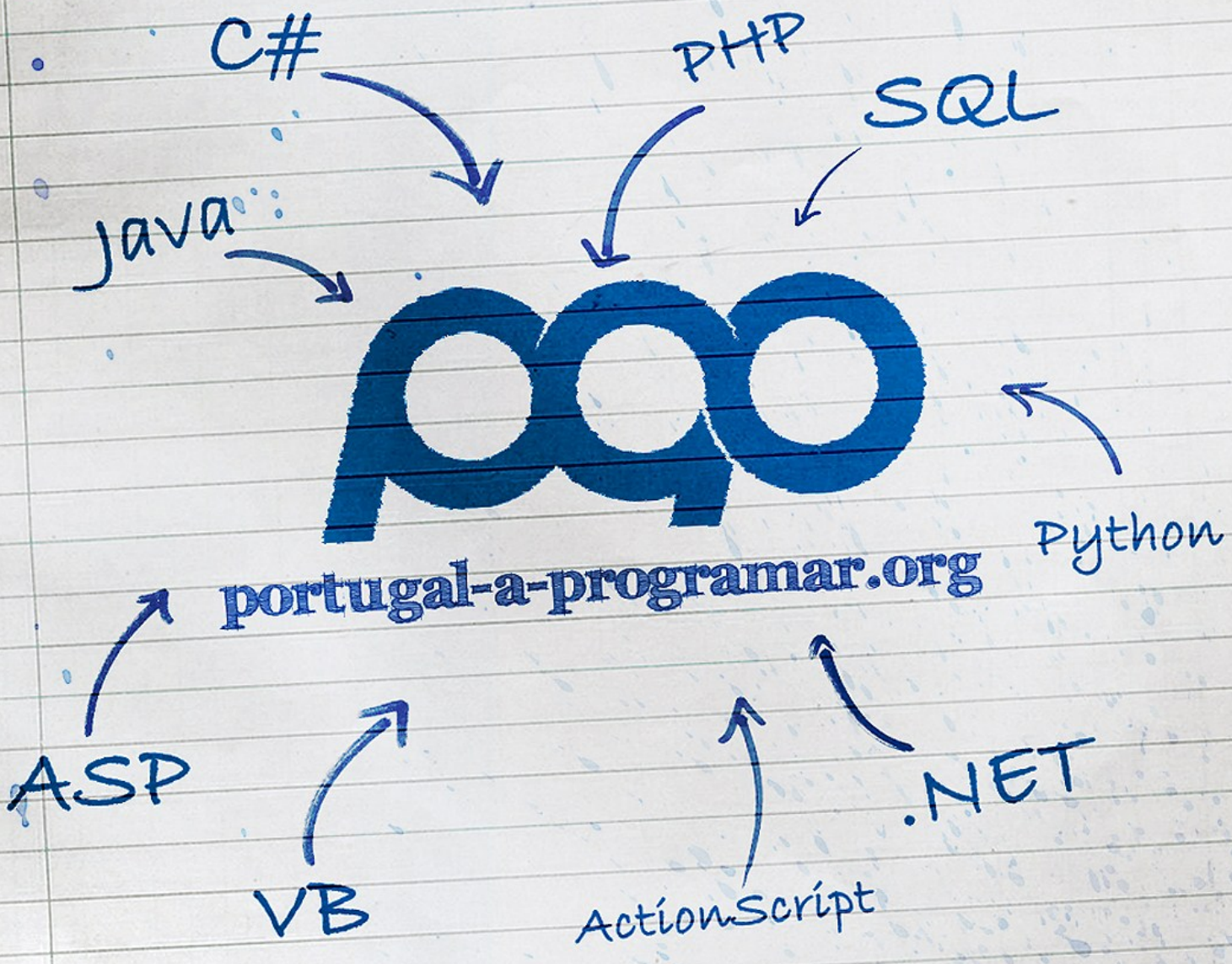
WIKIPEDIA, a enciclopédia livre. RANDU. Disponível em: <http://en.wikipedia.org /wiki/RANDU>. Acesso em: 3 jan. 2012, 16:54:52.

AUTOR

Escrito por Augusto Manzano

Natural da Cidade de São Paulo, tem experiência em ensino e desenvolvimento de programação de software desde 1 986. É professor da rede federal de ensino no Brasil, no Instituto Federal de Educação, Ciência e Tecnologia. É também autor, possuindo na sua carreira várias obras publicadas na área da computação.

A maior comunidade portuguesa de programação!



Visite-nos!

Herança, em JavaScript

Introdução

Apresentando-se o JavaScript como uma linguagem de programação que desfruta de uma sintaxe semelhante à do Java, C, C++, etc., sendo ainda uma linguagem orientada a objectos, surge com esta a possibilidade de desenvolver a herança entre objectos.

Herança

A herança é um princípio da programação OO onde se especifica uma entidade base (no Java ou C# essa entidade é representada por uma classe), onde esta contém toda a informação comum a outras entidades (também representada por classes no Java ou C#) mais particulares.

Essas entidades particulares herdam da entidade base toda a informação que esta contém, restando apenas adicionar as novas características.

Para uma melhor compreensão, nada melhor que utilizar exemplos práticos.

Herança, em Javascript

Imagine-se um *Ponto*, caracterizado pelas suas coordenadas *x* e *y*. Para caso de exemplo, imagine-se dois outros tipos de pontos mais específicos: um *PontoCor* que contém para além das coordenadas a sua cor, e um *Ponto3D*, onde as suas coordenadas são definidas a 3 dimensões, sendo necessário a especificação de uma terceira coordenada, *z*.

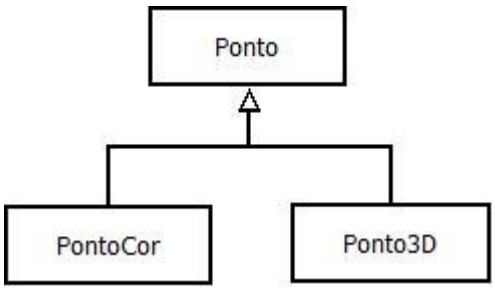


Fig.1 – Representação hierárquica das entidades *Ponto*, *PontoCor* e *Ponto3D*.

Se para a representação em Java do modelo de desenho da *Fig.1*, seria necessário desenvolver três classes (*Ponto*, *PontoCor* e *Ponto3D*), em JavaScript será necessário criar três funções construtoras que serão utilizadas na iniciação dos objectos do tipo *Ponto*, *PontoCor* e *Ponto3D*.

Função Construtora *Ponto*

```
var Ponto = function(x,y){
  var _x = x;
  var _y = y;

  this.obterX = function(){
    return _x;
  };

  this.obterY = function(){
    return _y;
  };

  this.iguais = function(ponto){
    if(this == ponto)
      return true;

    if (_x == ponto.obterX()
      && _y == ponto.obterY())
      return true;

    return false;
  };

}
```

Vejamos a definição da função construtora para o objecto, *Ponto*.

Como descrito atrás, o objecto Ponto contém as variáveis _x e _y, que simulam as coordenadas do ponto. São parametrizadas na iniciação do objecto e estão acessíveis através das propriedades obterX e obterY. Disponibiliza também o método de comparação, iguais.

A criação de objectos do tipo Ponto e a obtenção das coordenadas será a seguinte:

Exemplo 1

```
var p1 = new Ponto(2, 3);
var px = new Ponto(2, 3);

var x = p1.obterX();
var y = p1.obterY();
var iguais = p1.iguais(px);

alert("X:" + x + " Y:" + y + "\n"
      + "Iguais: " + iguais);
```

Quando iniciados os objectos, uma representação da sua estrutura, poderá ser algo do género ilustrado na *Fig.2*, em que a variável *p1* tem uma referência para um objecto *Ponto*

que contém a informação das propriedades, na forma:

|nome |valor |.

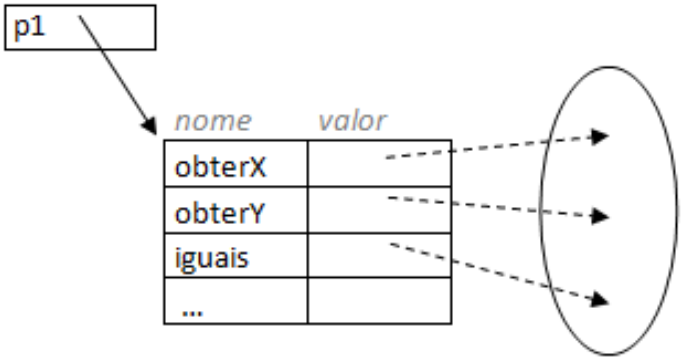


Fig.2 – Representação dos objectos *Ponto* em memória.

Uma vez que as variáveis _x e _y estão definidas localmente dentro da função construtora Ponto, os seus tempos de vida, serão o tempo de vida de execução da função. Desta forma é permitido aceder a estas devido à existência do mecanismo de "captura de contexto".

Relembre-se que em cada iniciação de um objecto Ponto, o *this* será sempre o objecto que é iniciado. Quer isto dizer que no Exemplo 1, p1 é o *this* do objecto iniciado tendo por isso acesso às propriedades obterX e obterY, tal como há propriedade iguais.

Assumindo que este código será integrado e executado no contexto de um ficheiro HTML, o browser deverá mostrar a informação das coordenadas, bem como a informação de comparação dos pontos, ou seja, se o ponto this e o ponto recebido por parâmetro, são ou não iguais (tanto em referencia como em coordenadas), que neste exemplo retornará true. Veja-se a mensagem seguinte, retornada pelo browser:

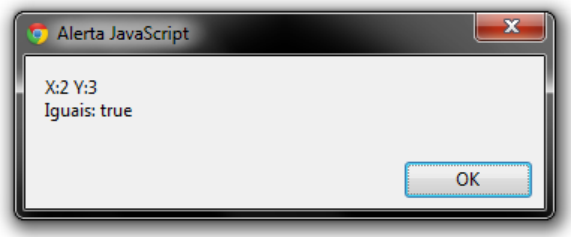


Fig.3 – Janela de *alert* no *browser*.

Função Construtora PontoCor

Continuando com a definição das funções que representam os objectos ilustrados na Fig.1, apresenta-se em seguida a função construtora para o objecto PontoCor.

Na definição anterior, foi anotado em comentário junto à primeira instrução dentro do *scope* da função, a palavra "Herança". De facto, a instrução:

Ponto.call(this, x, y);

é a chave da herança no JavaScript, no contexto deste artigo[1]. Qualquer função que pretenda herdar as características de *Ponto,* necessita apenas de incluir esta instrução (preferencialmente deverá ser a primeira instrução dentro da função; não que seja obrigatório, mas por uma questão estética).

A função *call* permite invocar uma função de forma explícita, onde no primeiro parâmetro é passado aquele que será o *this* dentro da função que é chamada, sendo os restantes parâmetros, os argumentos definidos para a função.

Isto torna tudo possível: o *this* da função *PontoCor* será o *this* dentro da função *Ponto* e deste modo irá ter acesso às propriedades que *Ponto* dispõe, garantido assim, a herança das propriedades.

1 A herança em JavaScript, poderá ser também efectuada através da utilização de protótipos, mas isso seria assunto para um outro artigo.

Nota

```
/*
      Ponto.call(this, x, y);
```

Poderia ser substituída por duas outras instruções:

```
this.Super = Ponto;
      this.Super(x,y);
*/
```

Em seguida, apresenta-se um exemplo de iniciação de objectos *PontoCor*, bem como a chamada às propriedades que foram herdadas de *Ponto*.

Exemplo 2

```
var p2 = new PontoCor(2, 3, "Azul");
var px = new Ponto(2, 3);

var x = p2.obterX();
var y = p2.obterY();
var cor = p2.obterCor();
var iguais = p2.iguais(px);

alert("X:" + x + " Y:" + y + " Cor:"
```

```
      + cor + “\n”
      + "Iguais: " + iguais);
```

Como é expectável, os valores obtidos para as variáveis, irão conter o valor com que o objecto foi iniciado. Em relação à igualdade entre pontos, é utilizada a comparação que foi herdada de *Ponto* uma vez que um *PontoCor* e um *Ponto* são iguais, se as suas coordenadas forem iguais. A Fig.4, mostra o resultado da execução do *browser* de um ficheiro HTML onde estava contido o código anterior.

Fig.4 – Janela de alert no browser.

Tal como referido anteriormente, quando é iniciado um objecto, ele irá conter todas as propriedades na forma |nome |valor |. No caso de PontoCor (e qualquer outro objecto que pretenda usufruir do mecanismo de herança), irá conter adicionalmente, as propriedades herdadas da entidade base.

Veja-se na Fig.5, a representação da estrutura do objecto com as suas propriedades, juntamente com as propriedades que herdou de Ponto.

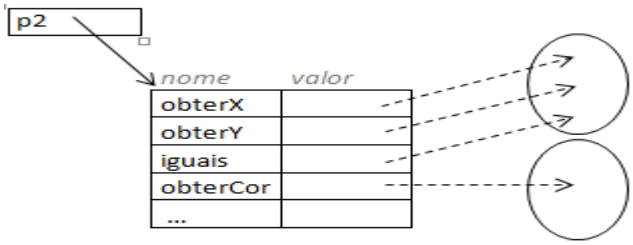


Fig.5 – Representação dos objectos PontoCor em memória.

Tal como no objecto Ponto, a variável _cor fica inacessível através de qualquer objecto PontoCor. Contudo, também irá ser acedida dentro do método obterCor devido há já referida “captura de contexto” que é realizada pela função.

Função Construtora Ponto3D

```
var Ponto3D = function(x,y,z){

  Ponto.call(this, x, y);

  var _z = z;
```

```
  this.obterZ = function(){
    return _z;
  };

  // Sobrecarga de métodos
  this.super_iguais = this.iguais;

  this.iguais = function(ponto3D){
    if (!(ponto3D instanceof Ponto3D))
    return false;
    return this.super_iguais(ponto3D)
        && _z == ponto3D.obterZ();
  };

}
```

Concluindo a definição dos objectos propostos inicialmente (Fig.1), veja-se em seguida a definição para a função construtora, Ponto3D.

Tal como na definição da função construtora de *PontoCor*, também a função construtora *Ponto3D* inclui na sua definição a instrução:

Ponto.call(this, x, y);

uma vez que irá também herdar as características de um *Ponto,* às quais irá ainda juntar a nova coordenada *z* (que programaticamente será a variável *_z*). Deste modo, terá acesso às propriedades *obterX*, *obterY* e *iguais,* e às quais irá juntar *obterZ,* que também este executa “captura de contexto”.

Recordando um pouco o que sucede com a herança em outros ambientes programáticos (por exemplo no Java ou C#), poderá, por vários motivos, surgir a necessidade de alterar/manipular o comportamento de uma função que é herdada. O processo de recriar uma função com o mesmo nome de uma função herdada, designa-se de Sobrecarga (chamado *override* nas plataformas Java ou C#). Neste contexto, ocorreu a necessidade de sobrecarregar a função *iguais*, uma vez que a comparação de dois *Ponto3D* não é igual à comparação feita entre pontos a duas coordenadas.

De modo a não perder a referência para a função herdada *iguais* e como forma de reutilizar esse troço de código já desenvolvido, foi mantida a referência numa nova propriedade, *super_iguais*, que é depois utilizada na nova função referenciada pela propriedade *iguais*.

Apesar de não se poderem criar novos tipos em JavaScript, a instrução:

x instanceof FuncX

Verifica se uma determinada variável/propriedade *x* repre-

senta um objecto iniciado por uma determinada função construtora *FuncX*.

Segue-se o Exemplo 3, onde são ilustrados exemplos de iniciação de objectos *Ponto3D*, bem como o acesso às suas propriedades.

Exemplo 3

```
var p3 = new Ponto3D(1, 3, 7);
var py = new Ponto3D(1, 3, 7);
var px = new Ponto(1, 3);

var x = p3.obterX();
var y = p3.obterY();
var z = p3.obterZ();

var iguais = p3.iguais(px);
var iguais3D = p3.iguais(py);

alert("X:" + x + " Y:" + y + " Z:"
      + z + "\n"
      + "Iguais: " + iguais + "\n"
      + "Igual(3D): " + iguais3D);
```

Tal como é expectável, as propriedades que retornam os valores das coordenadas do ponto, irão retornar os valores com que o ponto foi iniciado, nomeadamente 1, 3 e 7, para x, y e z, respectivamente. Em relação à comparação de objectos utilizando a propriedade iguais, foram realizadas duas comparações: uma primeira onde se compara se o ponto p3 é igual ao ponto px, onde é expectável que seja retornado false visto que um Ponto3D não é um Ponto, e uma segunda comparação, onde se compara se o ponto p3 é igual ao ponto py, onde ai sim, é expectável que o retorno seja verdadeiro.

Por forma a comprovar este facto, o código do exemplo anterior, foi inserido num ficheiro HTML, e executado pelo browser, que retornou a seguinte mensagem:

Fig.6 – Janela de *alert* no *browser*.

Em relação a uma representação dos objectos iniciados a partir da função construtora Ponto3D, os objectos não escapam a uma estrutura algo afigurada com as representações dos objectos que se falaram ao longo do artigo. Contudo, existe um pormenor a ter em atenção: a propriedade iguais, não irá ter o código da sua função na mesma zona onde está código dos outros métodos herdados.

Isto, porque convém não esquecer que a função foi sobrecarregada para ter um comportamento diferente daquele que tinha nos objectos Ponto. Logo, o seu código irá estar num zona junto com código da função obterZ. Veja-se então na Fig.7, o que acaba de ser descrito.

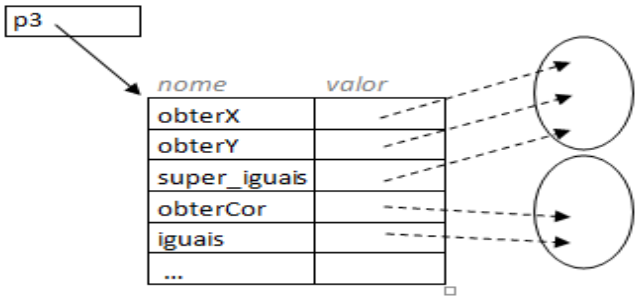


Fig.7 – Representação dos objectos *Ponto3D* em memória.

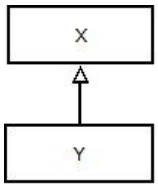
Reprensentação dos Objectos

Repare-se na representação dos objectos ao longo do artigo. Os valores das propriedades não contém o código fonte das funções, referenciando uma zona representada pela circunferência. Isto sucede, uma vez que o código das funções é mantido numa zona comum a qualquer objecto que seja iniciado a partir da função construtora. Assim, evita-se que todas as funções de um objecto contenham o código fonte dentro deste, evitando um desperdício excessivo de memória pela replicação do código fonte por todos os objectos.

Conclusão

Durante o artigo, foi ilustrado um exemplo de desenho para o caso onde se pretendeu usar o mecanismo de herança, na definição de pontos. Contudo, desenvolver este mecanismo torna-se mais simples (espera-se) depois de ler na integra o artigo.

Terminando:



Qualquer entidade Y que pretenda herdar características de uma base X, basta apenas incluir a chamada explicita à função base passando-lhe o *this*, tais como os restantes parâmetros de iniciação, ou seja:

```
var X = function(n){
  ...
```

```
}
var Y = function(n,m){

  X.call(this, n);
  ...
}
```

Espera-se que o leitor tenha desfrutado deste artigo, e que melhor ainda, fique com o conhecimento para desenvolver "Herança, em JavaScript".

Sugestão

/*

Caso o leitor pretenda testar o código presente no artigo, bem como outro tipos de funcionalidades do JavaScript, poderá desfrutar da utilização de uma biblioteca *opensource* para realização de testes unitários em JavaScript, designada *QUnit*.

*/

Bibliografia:

FLANAGAN, David. JavaScript: The Definitive Guide, 5th Edition. O'Reilly, August 2006

Toda a restante informação foi adquirida durante as aulas de licenciatura.

AUTOR

Escrito por Luís Cunha

Finalista do curso Engª Informatica e de Computadores, pelo Instituto Superior de Engenharia de Lisboa (ISEL). Desempenha funções de Consultor de TI. http://www.facebook.com/sucunha90

Enigmas de C# - Disposable Structs

por Paulo Morgado

Dada a seguinte estrutura:

```
public struct DisposableStruct : IDisposable
{
    public bool isDisposed;

    public void Dispose()
    {
        Console.WriteLine(
            this.isDisposed
            ? "Already disposed!!!"
            : "Disposing...");

        this.isDisposed = true;
    }
}
```

Qual é a saída do seguinte código?

```
using (var v = new DisposableStruct())
{
    v.Dispose();
}

var s = new DisposableStruct();
using (s)
{
    s.Dispose();
}
```

Veja a resposta e explicação na página 36!

Elege o melhor artigo desta edição

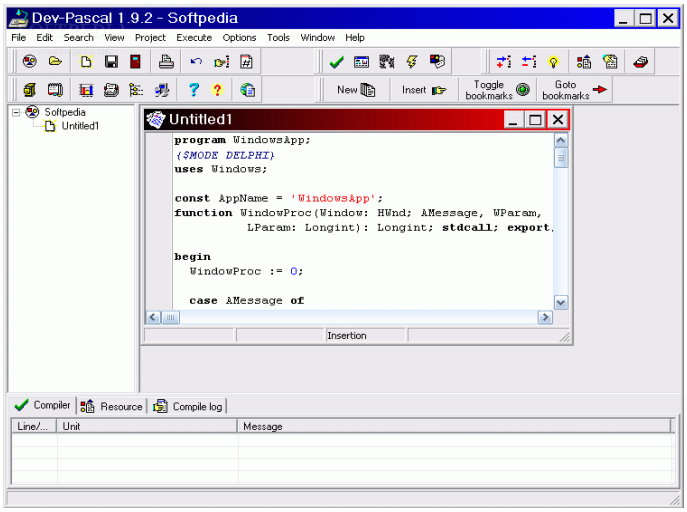
Revista PROGRAMAR

http://tiny.cc/ProgramarED33_V

Programação Orientada aos Objectos em Pascal

Introdução

Há quem pense no Pascal como uma linguagem fraca e antiquada que não acompanhou os novos tempos. Mas, na realidade, o Pascal evoluiu em vários dialectos, sendo o mais proeminente o Delphi.



Hoje em dia é possível programar nesta linguagem segundo o paradigma POO (Programação Orientada aos Objectos), programar DLLs (Dynamic Link Libraries), fazer programas não só em consola mas também com recurso a GUI. Permite igualmente a ligação a bases de dados, como o MySQL, bem como a criação de IDEs. O grande exemplo é o, infelizmente descontinuado, IDE Dev-Pascal, da Bloodshed, um IDE open-source, escrito totalmente em Delphi e recorrendo a uma velha versão do FPC (Free Pascal Compiler) e do GPC (GNU Pascal Compiler) como compiladores opcionais integrados.

Posta esta breve introdução, os objectivos para o presente artigo são:

1. Dar a conhecer noções teóricas básicas sobre o paradigma POO;

- Apresentar o dialecto Object Pascal;
- Utilizar ferramentas do Free Pascal e do Delphi para criar programas segundo o paradigma POO
- Criar uma classe útil para os leitores.

Doravante, a linguagem Object Pascal será designada tão-somente por Pascal por duas razões:

2. Genericamente, qualquer dialecto derivado do Pascal (leia-se Standard Pascal) ser designado exacta e somente por Pascal;

3. Para simplificar a leitura do presente artigo.

Os trechos de código aqui presentes foram todos testados recorrendo ao Free Pascal 2.4.4.

Em todos os códigos aqui presentes foi utilizado o modo de compatibilidade com Delphi do Free Pascal, pelo que estará presente nestes a directiva de compilação **{$mode delphi}.**

Preliminares em POO

O âmbito do presente no artigo não é uma explicação alongada deste paradigma. Contudo, vale uma explicação no âmbito do Pascal e daquilo que será utilizado adiante.

Serão então abordados os conceitos principais deste paradigma, sendo estes: classes, métodos, propriedades, instâncias, herança e visibilidade (ou protecção).

Classes, métodos, propriedades e instâncias

Uma classe é um conjunto de objectos com características iguais. Um método da classe não é mais do que uma capacidade da classe, ou seja, um procedimento ou função. Uma propriedade é uma característica do objecto. Uma instância da classe é um objecto com todas as características dessa mesma classe.

De forma prática, vamos considerar este paradigma num caso da vida real.

Considere-se a classe Ser Humano:

- **Propriedades** – altura, peso, idade, sexo, cor do cabelo.
- **Métodos** – andar, falar, sentar, dormir, cozinhar, trabalhar.

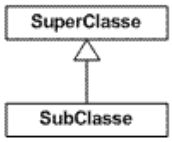
Igor Nunes é uma instância desta classe, por exemplo. Sendo uma instância, Igor Nunes terá exactamente as propriedades altura, peso, idade, etc., bem como os métodos andar, falar, dormir, etc. Já a instância João terá exactamente as mesmas propriedades e métodos. O próprio leitor é uma instância desta classe!

Herança

Uma das capacidades mais impressionantes e úteis do paradigma POO é a possibilidade de uma classe herdar métodos e propriedades de outra. Neste âmbito, temos então duas classes, uma que deriva de outra:

Classe principal (ou Main Class) – classe que têm os métodos e propriedades gerais;

Classe derivada (ou Inherited Class) – classe que vai "buscar" os métodos e propriedades da anterior – deriva da classe principal.



Representação de herança entre classes em UML.

Voltando à classe Ser Humano:

1. Mamífero é uma Main Class de Ser Humano;
2. Caucasiano é uma classe derivada de Ser Humano;
3. Reino Animal é uma Main Class de Mamífero;

Isto significa então que, respectivamente:

1. Ser Humano é um Mamífero;
2. Caucasiano é um Ser Humano;
3. Mamífero é do Reino Animal.

Visibilidade de métodos e propriedades

A classe pode ter métodos e propriedades com visibilidades diferentes:

- **Públicas** – podem ser vistos por qualquer outra classe, programa ou unidade;
- **Privadas** – só podem ser vistos pela própria classe, isto é, pelos restantes métodos e propriedades, independentemente da sua visibilidade. Só estão ocultos para fora da classe, e não para dentro da classe. São igualmente ocultos para classes derivadas.
- **Protegidas** – são visíveis tão-somente pela classe e por qualquer outra que derive desta.

Considerando a classe Ser Humano, temos que, por exemplo:

- O método falar é **protegido**, já que não é visível a outras classes como Leão, Pinguim ou Urso, mas é visível às classes que lhe têm herança, como por exemplo Etnias e Raças;
- A propriedade altura é **pública**, já que qualquer um pode ver e saber quanto mede um ser humano em altura;

De referir que, se nenhum bloco de visibilidade for criado, então tudo o que for declarado na classe é considerado público por default.

Colocada a teoria essencial do paradigma POO, segue-se a exemplificação da sua implementação em Pascal, não antes de dar a conhecer o dialecto do Pascal que suporta este mesmo paradigma.

O Object Pascal

Apresentado em 1986, o Object Pascal foi uma linguagem de programação com a mesma sintaxe do Standard Pascal, de 1971, e já com as modificações introduzidas pelo Extended Pascal, e foi concebido por uma equipa da Apple Computer (hoje Apple Inc.), liderada por Larry Tesler, em conjunto com o criador do Pascal, Niklaus Wirth.

O seu nome original teria sido Clascal, e foi concebido devido ao facto de este ser necessário para uma Framework da época, a MacApp.

Deu-se assim o suporte do Pascal ao paradigma POO.

Com a introdução do POO no Pascal, esta linguagem tornou-se muito mais poderosa e moderna, sendo cada vez mais um exemplo da estruturação de um programa, módulo ou biblioteca.

A sintaxe do Pascal variou consoante os dialectos que foram surgindo. Contudo, podemos ter dois padrões de referência, sendo o utilizado neste artigo o segundo:

- Free Pascal;
- Delphi.

Passemos então à implementação em código.

Classes

Um conjunto novo de palavras reservadas surgiu, e entre elas estão as palavras reservadas **Object** e **Class**. É de referir que um **Object** pode ser qualquer coisa, sendo, por-

tanto, a mãe de todos os tipos de dados, incluindo procedimentos e funções.

Para agora, a palavra que nos interessa é a **Class**. Com esta palavra reservada criamos uma classe, tal como o seu nome sugere.

Em Pascal, uma classe deverá ser sempre declarada num tipo (**type**) já que as **instâncias** da classe serão criadas como variáveis (**var**).

Este tipo não começa com a palavra reservada **begin** mas sim com a própria palavra **class**, mas termina obrigatoriamente com **end**. Dentro deste bloco que se cria teremos a declaração, mas não programação, dos métodos da classe. Igualmente, aqui estarão as propriedades.

Para definir as visibilidades, podemos criar três blocos. Nenhuma delas necessita de estar num bloco **begin-end** já que se considera que o bloco termina quando é encontrada a declaração de um outro bloco, ou o fim da classe.

Na prática, a estrutura de uma classe será a seguinte:

```
type Nome_Classe = class private
 // parte privada
protected
// parte protegida
public
// parte pública
end;
```

Métodos

A parte principal de uma classe é o conjunto dos seus **métodos** – as "habilidades" da classe, os processos que esta é capaz de realizar. E, em Pascal, os métodos não são mais do que **procedimentos e funções**.

Na classe, estes apenas são declarados. Nunca são programados nesta parte:

```
type Nome_Classe = class
  private function funcao(const texto :
     tipo_dado) : tipo_output;
public procedure procedimento ( const
                argumentos : tipo_dado);
end;
```

Neste caso não temos métodos protegidos. Como podemos verificar, os métodos da classe são declarados dentro dos seus blocos.

Então, onde são programados? A resposta é: tal como outro procedimento e função quaisquer: fora do bloco principal do programa.

Neste caso, queremos que a função privada **conversor** converta um carácter ASCII para o seu sucessor ("A" passará a ser "B", por exemplo), e **escrever** será o método público que irá escrever o resultado da função privada **conversor**.

```
PROGRAM exemplo;
type CEx = class
   private function conversor ( const
              texto : string) : string;
public
procedure escrever(const texto : string);
end;
procedure CEx.escrever(const texto : string);
 begin writeln(conversor(texto));
end;
function CEx.conversor(const texto : string) :
 string; var i : integer; t : string;
  begin t := '';
   for i:=1 to length(texto) do
     t := t + char(ord(texto[i])+1);
     result := t;
end;
BEGIN
(* Bloco principal de execução *)
END.
```

Como podemos reparar, apesar de **conversor** ser uma função privada, **escrever** pode-lhe aceder já que ambos estes métodos pertencem à mesma classe. Em suma, o programa principal poderá aceder ao procedimento **escrever** mas nunca à função **conversor**.

Além disso, e muito importante de reter, é o facto de o método de cada classe ser programado identificando em primeiro lugar qual a classe a que pertence o método, e só depois dizendo o nome do método.

```
procedure/function classe.método({argumentos})
```

Fica a nota para o facto de não se ter utilizado a função padrão succ que nos daria de imediato o carácter ASCII seguinte – percebemos desta forma como funciona esta função para o caso dos caracteres.

Propriedades

Uma propriedade, por si só, não possui nenhum valor. Uma propriedade é um "atalho" para uma variável que, e essa sim, possui um valor específico. Por exemplo, o nosso peso é uma medida de massa que aparece na balança, mas, por detrás desse valor, está um mecanismo que mediu, registou e fez output desse mesmo valor. Neste exemplo, abordámos tudo o que há numa propriedade:

- Input de um dado (com o seu devido registo);
- Output desse mesmo dado (com a sua devida leitura).

Uma **propriedade** é, então, um conjunto de dois métodos (um input e outro output) que atribuem e lêem o valor de uma variável da classe.

```
property propriedade : tipo_dado read
  metodo_leitura write metodo_escrita;
```

Para melhor entender este conceito, vamos partir de um exemplo prático e que será já parte da nossa implementação prática:

```
type TAngle = class
  private VDEG : real; // Variável da qual depende
a propriedade
    procedure DefinirValorDEG(const valor : real);
  public
    property ValorDEG : real
      read VDEG write DefinirValorDEG;
end;

procedure TAngle.DefinirValorDEG
(const valor : real);
(* Atribui valor à propriedade em graus *)
begin
  self.VDEG := valor;
end;
```

Isto é o que acontece na prática e na grande maioria das propriedades. Como podemos ver, a propriedade, de seu nome **ValorDEG**, tem uma variável "por detrás" que possui o seu valor: a variável privada **VDEG**.

Aqui começamos já a ganhar noção da vantagem das **visibilidades** das propriedades: o que nos interessa é que se veja a propriedade da classe, e não a variável na qual está atribuída o valor dessa propriedade. Assim, mantemos essa variável **privada**.

Para fazer o *output* da propriedade, bastará ler o valor da sua variável **VDEG**, pelo que o nome do **método de escrita** é o próprio nome da variável. O método de escrita deverá ser sempre uma **função**, mas como a variável em si é uma função, este processo torna-se válido.

Já a escrita necessita de um **procedimento**. Neste caso, criámos o procedimento **DefinirValorDEG** que tem única e exclusivamente um argumento. Todo e qualquer método de escrita deverá ter um e um só argumento, sendo este do mesmo tipo que a variável da qual depende a propriedade. É este argumento que recebe o valor a ser atribuído à propriedade, leia-se à variável.

Neste caso, **DefinirValorDEG** recebe **valor** e atribuí-o à variável **VDEG**. Para não confundir com nenhuma variável global com o mesmo nome que pode ser criado para o programa, utilizamos a palavra reservada **self**, que indica ao compilador que a variável **VDEG** é a referente à classe cujo método está ali a ser programado: neste caso, a classe **TAngle**.

Construtores

Uma instância de uma classe, que será vista a seguir, não pode surgir do nada. A instância tem de ter um construtor de modo a que a variável que criarmos seja, de facto, uma instância. Caso não façamos a construção, a variável do tipo da classe será uma referência nula e, logo, surgirá uma excepção na execução do programa.

O tipo de dados geral **TObject** tem o seu construtor: o **Create**. Vamos invocar este construtor nas nossas classes para podermos construir o nosso. O processo de herança será feito pela palavra reservada **inherited**.

O construtor deverá ser público, e é declarado pela palavra reservada **constructor**. Poderá receber argumentos ou não, depende do processo de construção da classe.

Regra geral, com o construtor damos os valores por *default* às propriedades da classe. Estes valores poderão vir dos argumentos.

```
type Nome_Classe = class
  private // parte privada
  protected // parte protegida
  public
    constructor Create({argumentos do construtor});
end;
constructor Classe.Create({argumentos do constru-
tor});
begin
 inherited Create; // Invoca construtor da classe
ascendente TObject.
 (* Código do construtor *)
end;
```

Instância

Por fim, agora que a classe está criada, vamos criar instâncias desta. Uma **instância** é criada por uma variável do tipo da classe. Em Pascal:

```
var Instância : Nome_Classe;
```

No bloco de execução, antes de utilizar qualquer método ou atribuir ou ler qualquer propriedade, é necessário criar a instância através do construtor:

```
Instância := Nome_Classe.Construtor({argumentos});
```

Note-se bem que, no construtor, é o nome da classe que se coloca e não o nome da instância!

E, daqui em diante, pode-se utilizar qualquer método da classe, atribuir e ler propriedades desta, mas, **para cada instância, cada valor**. Se tivermos dez instâncias, cada uma pode ter um valor diferente para uma mesma propriedade. Isto é perceptível no caso da classe *Ser Humano*:

- O João pesa 75Kg;
- O André pesa 64Kg;
- A Joana pesa os seus leves 52Kg;
- Já a Mariana chega aos 86Kg;

Destrutor

Já que neste artigo não vamos criar destrutores, fica a referência que, para qualquer classe, podemos utilizar o destrutor geral: o **Free**.

```
Instância.Free;
```

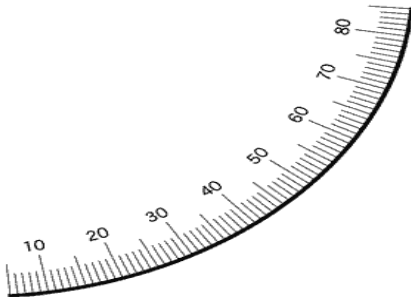
Antes de terminar um programa, as instâncias devem ser, então, "destruídas":

Implementação prática

Terminada que está a introdução teórica à utilização de Programação Orientada aos Objectos em Pascal, a melhor forma de consolidar estes conhecimentos é com um exercício prático.

Criemos, então, uma classe, de seu nome **TAngle**, e que permita guardar o valor de um ângulo em graus e radianos e tenha as devidas funções de conversão. Para tal, teremos:

- **Métodos:**
  - Conversor radianos -> graus;
  - Conversor graus -> radianos.
- **Propriedades:**
  - Valor do ângulo em graus;
  - Valor do ângulo em radianos.



Para as propriedades, necessitaremos dos devidos e respectivos **métodos de escrita**:

Definir valor em graus;

Definir valor em radianos.

E, para a criação de uma instância da classe, necessitaremos de um **construtor**, construtor esse que terá dois argumentos:

Valor do ângulo;

Tipo de ângulo (graus ou radianos).

Para o tipo de ângulo, vamos criar especialmente o tipo de dados **TAngulo**, e que não deve ser confundido com a classe **TAngle**, e que assumirá os "valores" **deg** ou **rad**, respectivamente para graus e para radianos.

Além disso, o objectivo é que, quando se cria ou se modifica o ângulo, o valor mude automaticamente para os respectivos valores no outro tipo de ângulo. Ou seja, se atribuirmos em graus, o valor em radianos é modificado automaticamente para o valor correspondente.

Por fim, no programa, vamos testar rapidamente o construtor e os conversores.

Passando tudo isto para Pascal:

```
{$mode delphi} // Utiliza a nomenclatura do Delphi.
PROGRAM Teste_Classes;
uses crt, math; // Unidades necessárias
type TAngulo = (Deg, Rad); // Tipo de ângulo: Graus
                           //ou Radianos.
TAngle = class private (* Só visível para a classe
                        *) // Variáveis onde são //
                        //guardados os valores das
                       //propriedades.
VRAD : real; // Radianos
VDEG : real; // Graus
// Métodos que permitem atribuir valores às
                                 //propriedades.
procedure DefinirValorRAD(const valor : real);
 procedure DefinirValorDEG(const valor : real);
 public (* Visível para todo o programa *)
// Propriedades
  property ValorRAD : real read VRAD write
   DefinirValorRAD; property ValorDEG : real read
VDEG write DefinirValorDEG;
// Conversores function Deg2Rad(const deg :
real)  : real; function Rad2Deg(const rad :
real)  : real;
// Construtor da classe
 constructor Criar(const valor : real;
TipoAng : TAngulo);
 end;
constructor TAngle.Criar(const valor : real;
TipoAng : TAngulo);
(* Construtor *)
begin
inherited Create; // Invoca construtor da classe
ascendente TObject.
case TipoAng of
// Conversores, consoante o tipo de ângulo
                                 //introduzido.
Rad : begin self.ValorRAD := valor;
self.ValorDEG := Rad2Deg(valor);
 end; Deg : begin self.ValorDEG := valor;
self.ValorRAD := Deg2Rad(valor);
end;
end;
end;
procedure TAngle.DefinirValorRAD(const valor :
real);(* Atribui valor à propriedade em radianos *)
begin self.VRAD := valor;
self.VDEG := self.Rad2Deg(valor);
end;
procedure TAngle.DefinirValorDEG(const valor :
real); (* Atribui valor à propriedade em graus *)
```

```
begin self.VDEG := valor;
self.VRAD := self.Deg2Rad(valor);
end; function TAngle.Deg2Rad(const deg : real) :
real;
begin result := (pi * deg) / 180;
end;
function TAngle.Rad2Deg(const rad : real) : real;
begin result := (rad * 180) / pi;
end;
var Angulo : TAngle; BEGIN
Angulo := TAngle.Criar(90, deg); // cria ângulo de
                                 //90º
writeln('Em radianos: ', Angulo.Deg2Rad
(Angulo.ValorDEG):0:3); // Converte em radianos
Angulo.ValorRAD := pi; // atribui 180º em radianos
writeln('pi(rad) em graus: ', Angulo.Rad2Deg
(Angulo.ValorRAD):0:3);
readln; // pausa
Angulo.Free();
END.
```

Apesar de os valores convertidos serem facilmente obtidos pela propriedade correspondente ao tipo de ângulo pretendido, a forma como realizámos a conversão nos procedimentos **writeln** demonstram como se podem utilizar vários métodos de uma vez só.

Note-se que se pode utilizar o bloco **with** para se evitar escrever continuamente o nome da instância seguido do método ou propriedade que se pretende. Contudo é necessária precaução no caso de haver várias instâncias e classes cujos métodos e propriedades partilham os mesmos nomes. Em caso de dúvida sobre o que vai o compilador assumir, considere sempre escrever na forma completa: ***nome_classe.método()***.

Fica então assim uma classe útil que o leitor poderá utilizar livremente nas suas programações em Pascal. Sinta-se à vontade para a expandir com novos métodos e propriedades, já que a melhor forma de aprender é a tentar.

Conclusão

Após este artigo, percebemos que o Pascal, já há quase 30 anos atrás, suporta o paradigma da Programação Orientada aos Objectos sem que a sua sintaxe original tivesse de ser alterada: apenas houve ligeiras adaptações e a entrada de novas palavras reservadas.

A entrada deste paradigma reforçou ainda mais o objectivo desta linguagem de programação: a programação estruturada, com “cabeça, tronco e membros” como se diz na gíria popular.

Conclui-se, sendo assim, que, apesar de muitos lhe chamarem uma linguagem ultrapassada, o Pascal está longe de ser uma linguagem do passado. Pascal é uma linguagem do século XXI.

Links úteis

Uma lista de documentos úteis da Wiki P@P, relacionados com o presente artigo.

Tutorial de Pascal (2011)

Parte I – Procedimentos e funções

Parte VI – Sucessor e predecessor

Parte VI – Lista padrão do Pascal

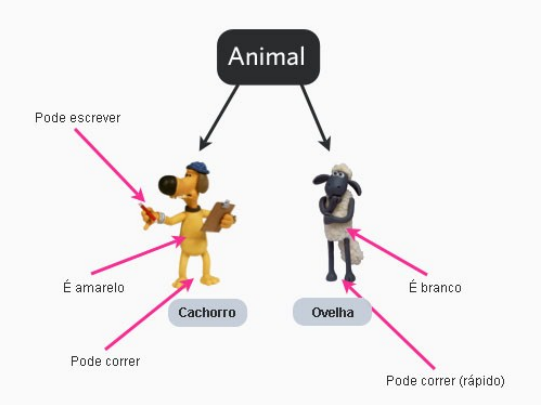
Tutorial de Delphi – noções básicas

Procedimentos e Funções – Passagem por Parâmetro / Passagem por Referência

Indentação

Fontes bibliográficas de apoio

http://pt.wikipedia.org/wiki/Ficheiro:UML_heranca.GIF



AUTOR

Escrito por Igor Nunes

Estudante de Ciências Farmacêuticas na Universidade da Beira Interior, entrou no mundo da programação aos 14 anos com TI-Basic. Dois anos depois descobriu o Pascal, que ainda hoje é a sua LP de eleição. Mais recentemente introduziu-se ao VB.NET e ao Delphi.

Membro do P@P desde Abril de 2010 (@thoga31), é actualmente membro da **Wiki Team** e **Moderador Local** dos quadros de Pascal e Delphi/Lazarus. Escreveu o novo Tutorial de Pascal da Wiki P@P, bem como os Tutoriais de TI-Basiz Z80 e de Introdução à Lógica e Algoritmia.

COLUNAS

VISUAL (NOT) BASIC – Iterators

CoreDump – Ambientes Produtivos

Enigmas de C#: Disposable Structs

Iterators

Os *Iterators* ou iteradores foram introduzidos no C# 2.0 (Visual Studio 2005) e agora, com o Visual Studio 2011 (*Developer Preview*), está também disponível no Visual Basic.

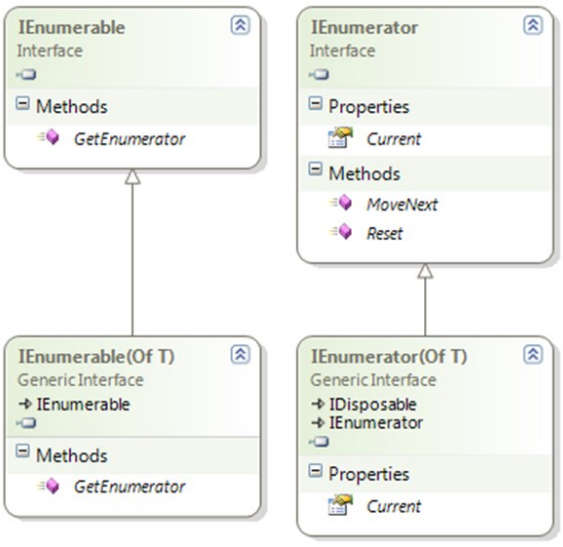
Para quem ainda não tem o Visual Studio 2011 *Developer Preview*, pode descarrega-lo através desta hiperligação http://tinyurl.com/6qq8558

Basicamente um *Iterator* é um método que realiza uma iteração sobre uma colecção, utilizando o Yield para retornar os elementos da colecção um-a-um. Os *Iterators* utilizam um mecanismo assíncrono que consegue retornar os itens logo que estejam disponíveis sem ter de esperar que a colecção esteja completamente construída.

Quando estamos a falar de uma colecção grande pode trazer vantagens significativas, quer em tempo, quer em recursos, uma vez que o tempo de espera diminui e também porque o seu mecanismo pode evitar a alocação em memória que seria utilizada caso a colecção estive-se completamente carregada. São a base da programação genérica. Para obter a informação basta utiliza uma simples instrução *For Each Next* directamente numa instância de um objecto que utilize *Iterators* ou então numa função ou propriedade.

Para criar um objecto que utiliza um Iterator basta implementar uma das seguintes interfaces:

- IEnumerable
- IEnumerable(Of T)
- IEnumerator
- IEnumerator(Of T)



Estas interfaces são utilizadas por exemplo, numa colecção List(Of T), Dictionary(Of TKey, TValue), ArrayList, etc.

Vantagens

- Tratar um possível erro directamente no Iterator;
- A cada chamada, o Iterator recomeça a partir do retorno anterior.

Limitações

- Um Iterator não pode receber parâmetros por referência;
- Um Iterator não pode ser utilizado num evento, numa instância de um construtor, num construtor/destrutor estático;
- Não suporta o método Reset.

Diferenças entre os *Iterators* em C# e *Iterators* em Visual Basic.NET

- Em VB é possível retornar um item dentro de um bloco *Try Catch Finally*;
- É possível sair de um *Iterator* em VB através de um *Exit Function* ou de um *Return*, em C# só é possível utilizando o *yield return*;
- Em VB é possível ter um *Iterator* anónimo, em C# não.

Onde usar

Os *Iterators* podem ser utilizados apenas em funções ou propriedades. Eis alguns exemplos:

Iterator Simples

Este exemplo utiliza uma instrução *For Each Next* que chama o *Iterator* que por sua vez retorna vários itens utilizando a instrução *Yield*. Vamos então criar um *Iterator* simples.

Um exemplo de utilização:

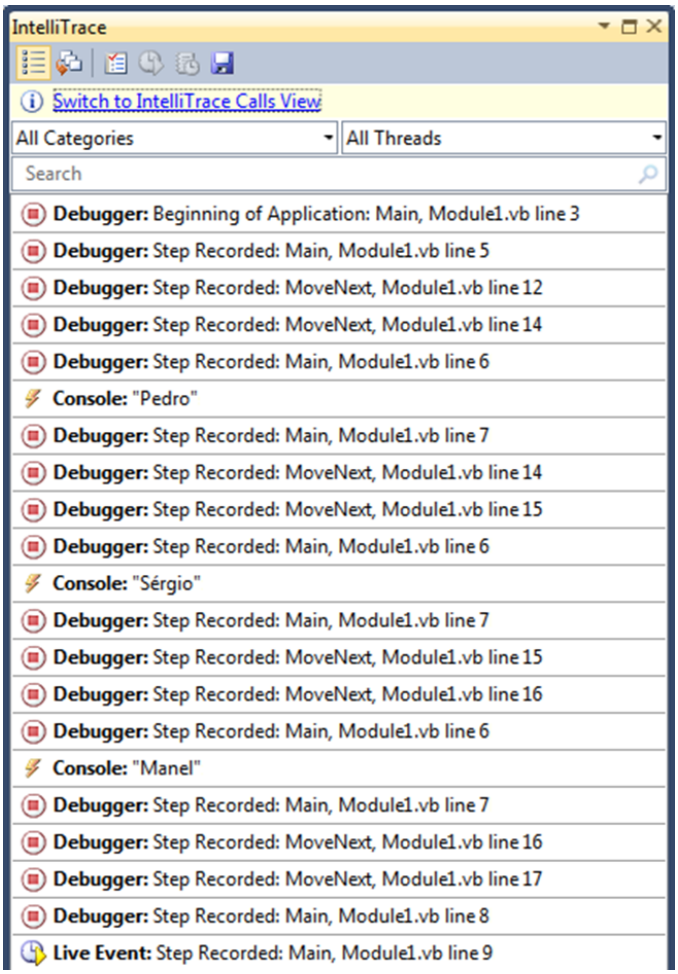
```
'Iterator com 3 retornos
Private Iterator Function Nomes() As _
    System.Collections.IEnumerable
    Yield "Pedro"
    Yield "Sérgio"
    Yield "Manuel"
End Function
```

```
Sub Main()
        'Inicia um ciclo For Each Next
        sobre a função Nomes
        For Each S In Nomes()
            Console.WriteLine(S)
        Next
        Console.Read()
    End Sub
```

Output: "Pedro", "Sérgio", "Manuel"

Para entender melhor o comportamento, basta fazer *Step Debug* ao código, ao faze-lo poderão ver como ele a cada iteração vai recomeçar no retorno anterior, evitando assim que toda a colecção seja novamente percorrida.

O IntelliTrace dá numa pequena noção do que se passou:



Analisando os métodos MoveNext, podemos verificar que no primeiro retorno foram executadas as linhas 12 e 14 (a linha 13 faz parte da linha 12, apenas esta dividida em duas), no segundo retorno foram executadas as linhas 14 e 15 e no terceiro as linhas 15 e 16.

No final ele ainda tentou ir para a linha 17 mas como já não haviam mais retornos a iteração foi dada como finalizada.

Tempo de Espera

Este exemplo mostra como o retorno é efectuado de uma maneira assíncrona, para isso vamos utilizar duas funções, ambas devolvem uma colecção do tipo List(Of Integer), mas uma com *Iterators* e outra com o método normal

```
Imports System.Threading
    Public Iterator Function _
        AlgunsNumerosComIterator() _
        As IEnumerable(Of Integer)
        Yield 1
        Thread.Sleep(1000)
        Yield 2
        Thread.Sleep(1000)
        Yield 3
        Thread.Sleep(1000)
        Yield 4
        Thread.Sleep(1000)
        Yield 5
    End Function

    Public Function _
        AlgunsNumerosSemIterator() _
        As List(Of Integer)
        Dim L As New List(Of Integer)
        L.Add(1)
        Thread.Sleep(1000)
        L.Add(2)
        Thread.Sleep(1000)
        L.Add(3)
        Thread.Sleep(1000)
        L.Add(4)
        Thread.Sleep(1000)
        L.Add(5)
        Return L
    End Function
```

Utilização

```
'Chama a nossa função com Iterator
For Each N In AlgunsNumerosComIterator()
      Console.WriteLine(N)
Next

Console.WriteLine()

'Chama a nossa função sem Iterator
For Each N In AlgunsNumerosSemIterator()
      Console.WriteLine(N)
Next

Console.ReadKey()
```

Output::

1, 2, 3, 4, 5
1, 2, 3, 4, 5

Iterators

As duas funções retornam exactamente a mesma coisa e no mesmo intervalo de tempo, no entanto a primeira, a cada segundo escreve na consola um número enquanto a segunda escreve tudo de uma só vez.

Para compreenderem melhor não há nada como experimentar.

Iterator parametrizável

Este exemplo mostra como utilizar um *Iterator* em que o retorno é gerado mediante informações passadas por parâmetro.

```
Imports System.Collections.Generic
  Private Iterator Function _
    SequenciaDeNumeros(
     ByVal NumeroInicial As Integer,
     ByVal NumeroFinal As Integer) _
      As IEnumerable(Of Integer)
      For NumeroActual As Integer = NumeroInicial _
          To NumeroFinal
          Yield NumeroActual
     Next
  End Function
```

Utilização:

```
Sub Main()
   'Inicia um ciclo For Each Next sobre a função
   'SequenciaDeNumeros onde é indicado também o
   'valor inicial e o valor final
    For Each I In SequenciaDeNumeros(1, 10)
            Console.WriteLine(I)
    Next
        Console.Read()
End Sub
```

Output: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Tipos Anónimos (Object)

```
Sub Main()
  'Esta função retorna um Iterator de tipo anonimo
  Dim Coleccao = Iterator Function() As IEnumerable
                           Yield "Pedro"
                           Yield "Sérgio"
                       End Function

        For Each S In Coleccao()
            Console.WriteLine(S)
        Next
        Console.Read()

End Sub
```

Output: “Pedro”, “Sérgio”

Com listas genéricas

```
Public Class ListaGenerica(Of T)
    'Implementa a interface IEnumerable(Of T)
    Implements IEnumerable(Of T)

    Private Array As T() = New T(24) {}
    Private PosicaoActual As Integer = 0

    'Metodo para preencher a nossa lista
    Public Sub Adicionar(ByVal Valor As T)
        Array(PosicaoActual) = Valor
        PosicaoActual += 1
    End Sub

    'Iterator implementado na interface
    'IEnumerable(Of T)
    'Este Iterator é utilizado por defeito
    'quando percorremos directamente
    'a instância do objecto.
    Public Iterator Function _
        GetEnumerator() As IEnumerator(Of T) _
        Implements IEnumerable(Of T).GetEnumerator
        For i = 0 To PosicaoActual - 1
            Yield Array(i)
        Next
    End Function

    'Iterator implementado na interface
    ' IEnumerable(Of T)
    Public Iterator Function _
        GetEnumerator1() As IEnumerator _
        Implements IEnumerable.GetEnumerator
        Yield GetEnumerator()
    End Function

    'Este Iterator retorna os primeiros x
    'números da colecção
    Public Iterator Function _
        Primeiros(ByVal Total As Integer) _
        As IEnumerable(Of T)
        If Array.Length < Total Then
            Throw New Exception(
                 "Este valor não é valido")
        End If
        For i = 0 To Total - 1
            Yield Array(i)
        Next
    End Function

End Class
```

Utilização:

```
Sub Main()
  'Declara uma nova instância da lista
   Dim L As New ListaGenerica(Of Integer)
   'Adiciona alguns itens
    For i = 1 To 25
     L.Adicionar(i)
    Next
   'Chama o Iterator directamente na instância
   'quando fazemos isto ele pega no Iterator
   'por defeito, chamado GetEnumerator()
    For Each I In L
     Console.WriteLine(I)
    Next
     Console.WriteLine()

    'Aqui como esta função já não é implementada
    'com a interface, já é necessário indicar o
    'nome da função a utilizar
    For Each I In L.Primeiros(5)
     Console.WriteLine(I)
    Next

     Console.Read()

End Sub
```

Output:

1, 2, 3, (…), 23, 24, 25.
1, 2, 3, 4, 5

Conclusão

Com este artigo dei a conhecer uma nova funcionalidade do VB e também uma nova forma de programar de forma assíncrona. Ao utilizar Iterators utilizamos uma maneira simples para diminuir o tempo resposta ao percorrer colecções, contribuindo assim para uma melhor programação.

Referencias

MSDN - http://tinyurl.com/79zowdd
Wikipedia - http://tinyurl.com/3yv487
All About Iterators - http://tinyurl.com/78uadzu

AUTOR

Escrito por Pedro Martins
É técnico Nível III em Informática/Gestão pela Escola Profissional Cisave É moderador do quadro Visual Basic.NET e ASP.NET na comunidade Portugal-a-Programar e também faz parte do staff da WikiTeam

Ambientes Produtivos

A área de informática é um dos casos mais prementes onde a gestão comete erros infantis no que toca à disponibilização de um ambiente produtivo. Estes erros são particularmente visíveis na área da consultoria, onde aos consultores externos não são dadas as melhores condições de trabalho (e a bem da verdade, aos colaboradores internos também não).

Um dos erros mais comuns é a utilização de *open spaces,* uma ideia que ganhou particular força a partir dos anos 50 sob a ideia de que colocar múltiplas pessoas num espaço aberto proporciona mais oportunidades para observar e aprender com aqueles que são mais experientes e possuem competências diferentes. Na verdade, há até estudos que provam que a produtividade de curto termo pode aumentar. Mas há também estudos que mostram que este tipo de ambiente pode ter quebras de produtividade até 15% e reduzir o bem estar das pessoas até 32%.[1]

Mas qualquer pessoa com bom senso não necessita de estudos científicos para observar que é totalmente contra-produtivo ter num mesmo espaço dezenas de pessoas com competências e funções distintas sentadas lado-a-lado. Rapidamente nos apercebemos que é impossível manter a concentração para desenhar uma arquitectura quando ao nosso lado está um comercial cujo telemóvel toca constantemente e que passa o dia a falar ao telefone e a gritar com a produção por não conseguir cumprir o prazo acordado com o cliente. Situação semelhante se passa quanto estamos a pensar numa solução para um problema complexo e na secretária à nossa frente se desenrola uma reunião com três pessoas onde por vezes os ânimos se exaltam na tentativa de fazer ver que determinada opinião é a melhor de todas as que se encontram em discussão.

Só quem nunca trabalhou num local fechado e sossegado não consegue avaliar a extensão dos danos que este tipo de ambiente faz. Mas o que mais me espanta é que este tipo de ambiente continua a ser patrocinado pelos próprios clientes, sendo a prática comum. Sendo os clientes que pagam estas falhas de produtividade dos seus colaboradores e as horas dos consultores externos, deviam ser os próprios a zelarem pelos seus interesses e proporcionar um ambiente produtivo que permitisse a execução das tarefas de forma expedita e com qualidade. No entanto, **os clientes, que tantas vezes se queixam do preço, parecem gostar de pagar esta factura**, uma vez que são os próprios que

atribuem estes horrendos lugares.

Mas há outros erros igualmente infantis que se repetem um pouco por todo o lado. A falta de tomadas eléctricas e pontos de rede é um deles. É comum um consultor externo levar o seu portátil e ser-lhe atribuído um desktop quando chega ao cliente, ou ser-lhe atribuída uma secretária onde se encontra um desktop, mesmo que não seja para seu uso. Quando se tenta ligar o portátil acontece sempre o mesmo filme: não há nenhuma tomada eléctrica livre e se por um feliz acaso houver um ponto de rede livre, este encontra-se desligado no bastidor.

É certo que o wifi veio minimizar este problema, mas os hubs de 8 portas e as extensões triplas ligadas a extinções quíntuplas continuam a existir aos molhos em cima e em baixo das secretárias, sempre acompanhadas por um emaranhado de fios que gostam de se enrolar nas nossas cadeiras ou nos nossos pés. Tudo isto em edifícios (ditos) modernos que têm pré-instalação de rede informática, como tanta vez ouço os comerciais da área imobiliária apregoarem. Quem se depara com esta triste realidade sabe que por vezes esta situação se torna tão ridícula que quando se quer carregar um telemóvel é necessário ligá-lo a uma tomada na casa de banho ou na copa. Pensar em ter um segundo monitor ligado ao computador então, nem pensar...

Os erros não terminam aqui, e há situações mais ou menos extremas que se tornam tão caricatas como ridículas, como seja a situação em que uma impressora tem direito a um lugar à janela e plantas verdes à sua volta e os profissionais se encontram ao fundo da sala, 12 por fila em 3 filas seguidas e que nem o estore podem abrir porque o ar condicionado não funciona e como tal têm de manter as janelas fechadas. Ou a situação em que a máquina do café fica mesmo ao lado da equipa de desenvolvimento, vendo-se forçada a suportar o irritante ruído do processo de moagem e compactação do café e extracção da água, vezes e vezes e vezes sem conta durante todo dia... Como se não bastasse, toda a gente se reúne à volta da máquina para conversar ou fazer mini-reuniões...

Os exemplos são inúmeros, basta olharem à vossa volta e vão encontrá-los sem dificuldade. Estes erros infantis podem ser fácil e rapidamente suprimidos. **Quem fizer as contas ao custo de uma hora de trabalho de todos quantos são afectados por estes erros, rapidamente se apercebe que o valor obtido até à hora de almoço já cobria o investimento na melhoria do ambiente para o tornar produtivo.**

Um bom gestor sabe que este tipo de ambiente é contra-produtivo mas, no entanto, estes ambientes prevalecem…

[1] Working in an office is bad for your brain.

AUTOR

Escrito por Fernando Martins
Faz parte da geração que se iniciou nos ZX Spectrum 48K. Tem um Mestrado em Informática e mais de uma década de experiência profissional nas áreas de Tecnologias e Sistemas de Informação. Criou a sua própria consultora sendo a sua especialidade a migração de dados.

SINFO

XIX SEMANA INFORMATICA

27 de Fevereiro a 2 de Março

Enigmas do C#: Disposable Structs

(continuação da página 21)

Resultado

```
Disposing...
Already disposed!!!
Disposing...
Disposing...
```

Explicação

Segundo a especificação do C# (§8.13) a instrução using é definida como:

using-statement:
    using (resource-acquisition)
        embedded-statement
    resource-acquisition:
        local-variable-declaration
        expression

Quando a *resource-acquisition* é da forma *local-variable-declaration*, como é o primeiro caso, o código é expandido para:

```
DisposableStruct v;
v = new DisposableStruct();
try
{
    v.Dispose();
}
finally
{
    v.Dispose();
}
```

Como se pode constatar, o método Dispose é invocado duas vezes para a variável v.

Como esperado, quando se invoca o método *Dispose* mais que uma vez na mesma instância é emitida a mensagem "*Already Disposed*!!!".

No entanto, quando a resource-acquisition é da forma expression, como é o segundo caso, o código é expandido para:

```
DisposableStruct s;
DisposableStruct _s;
s = new DisposableStruct();
_s = s;
try
{
    s.Dispose();
}
finally
{
    _s.Dispose();
}
```

Como se pode constatar, o método *Dispose* é invocado uma vez para a variável s e outra para a variável _s.

Tratando-se de um tipo por valor (value type – §1.3) como são as *struct* (§1.7), cada variável tem a sua cópia dos dados e não é possível efectuar operações sobre uma variável afectarem outra (excepto no caso de parâmetros ref e out).

Assim sendo, o método *Dispose* ao ser invocado uma vez para a variável s e outra para a variável _s significa que nunca é invocado mais que uma vez sobre os mesmos "dados".

Conclusão

O simples fato de que quando se copia uma instância de um tipo por valor se está a criar uma nova cópia dos dados, ao contrário dos tipos por referência onde se copia apenas uma referência para os dados, faz com que seja necessária especial atenção quando se lida com estes tipos.

Ligações Úteis

C# Reference

AUTOR

Escrito por Paulo Morgado
É licenciado em Engenharia Electrónica e Telecomunicações (Sistemas Digitais) pelo Instituto Superior de Engenharia de Lisboa e Licenciado em Engenharia Informática pela Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Pelo seu contributo para a comunidade de desenvolvimento em .NET em língua Portuguesa, a Microsoft premeia-o com o prémio MVP (C#) desde 2003. É ainda co-autor do livro "LINQ Com C#" da FCA.

COMUNIDADES

SharepointPT - Desenvolvimento em SharePoint 2010 - Parte 2

NetPonto - Biztalk Server - Princípios Básicos dos Mapas (2)

SQL Azure Federations na prática

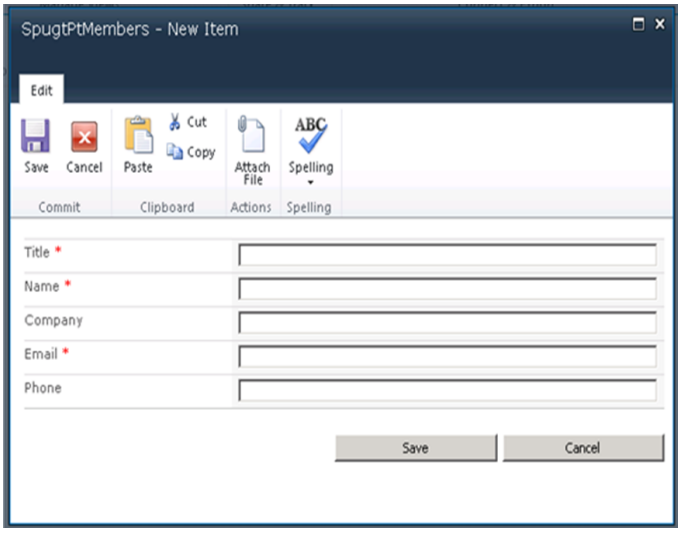
Desenvolvimento em SharePoint 2010 - Parte 2

Refactoring, Feature Activation Dependencies, Event Receivers e um sistema de Logging

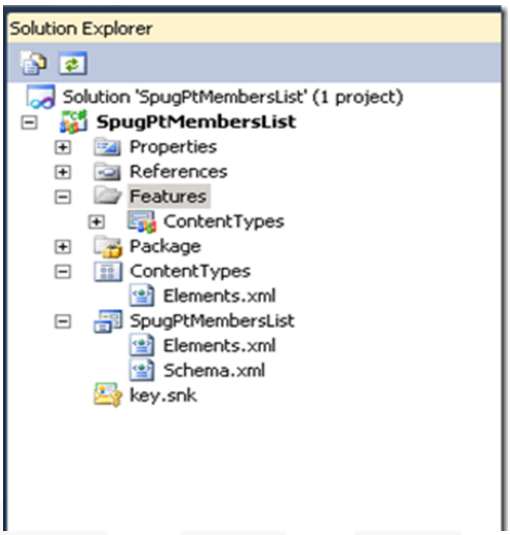
Dando continuidade ao artigo anterior de Desenvolvimento em SharePoint 2010 em que foi criada uma funcionalidade de registo de contactos dos actuais membros da Comunidade Portuguesa de SharePoint, neste artigo vamos potenciar a nossa solução com mais uma funcionalidade: um sistema básico de *logging*.

Apesar de ser, mais uma vez, um exemplo simples, acaba por demonstrar as potencialidades que a plataforma disponibiliza.

Se bem se recordam, a nossa lista de membros da Comunidade Portuguesa de SharePoint era qualquer coisa como isto:



e em termos do projecto Visual Studio, a estrutura era esta :



Restruturação

A 1ª coisa que reparo, quando revisito o projecto , é que tenho 1 *feature ContentTypes* a instalar 2 funcionalidades distintas , os *content types* e a definição da lista .

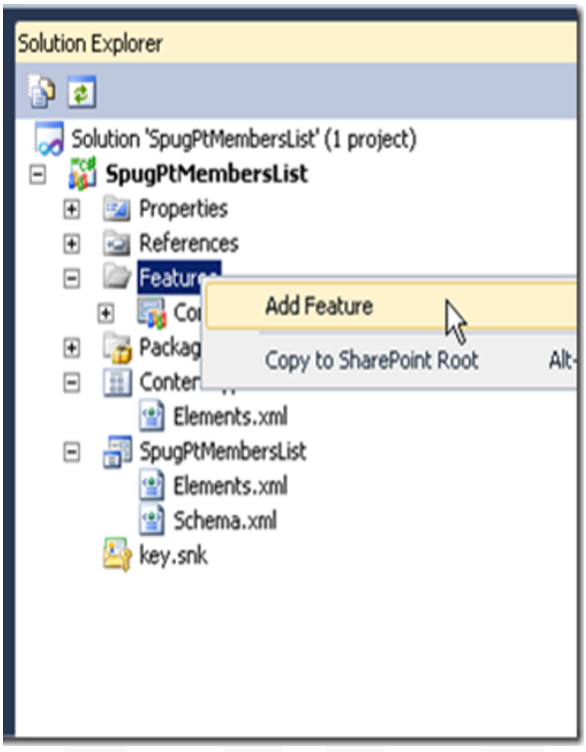
Humm….

Não me parece bem, pois em termos de boas práticas de implementação é importante separar contextos, ou seja, deveria ter uma feature que instalasse o content type e outra que instala-se a definição da lista.

Boa práctica #1: Esta questão de restruturação ou *refactoring* em projectos de desenvolvimento SharePoint ou qualquer outro tipo, é um factor a considerar, **SEMPRE .** Sejam críticos nas vossas implementações e nas das vossas equipas, e experimentem novas formas de desenvolvimento ;).

1. De volta ao nosso projecto, vamos criar uma nova feature **SpugPtMembersList**

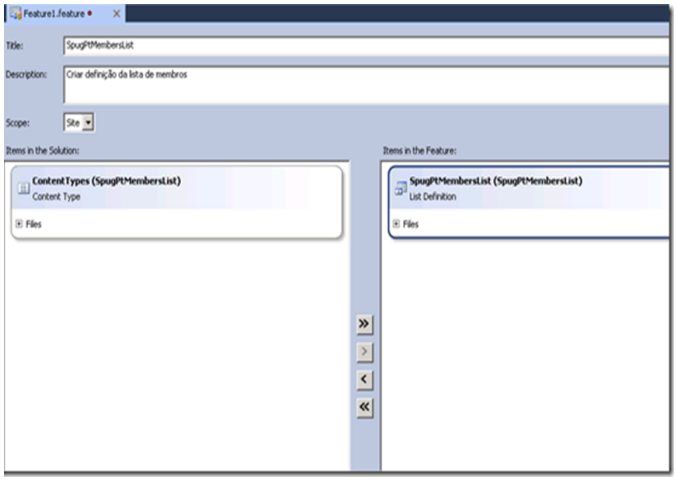
- Seleccionar "**Features" ,** clicar com o botão direito do rato para abrir o menu de contexto e seleccionar **"Add Feature"**



2. Caracterizar a feature :

- ◊ Title**: SpugPtMembersList**
- ◊ Description**: Criar definição da lista de membro**s
- ◊ Scope**: Site**
- ◊ Seleccionar Item **SpugPtMembersList** (1) da caixa **"Items in Solutions "** e passá-lo para **"Items in the Feature",** seleccionando a opção (seta) destinada a esse efeito (2)

3. Após esta operação, teremos algo deste género:



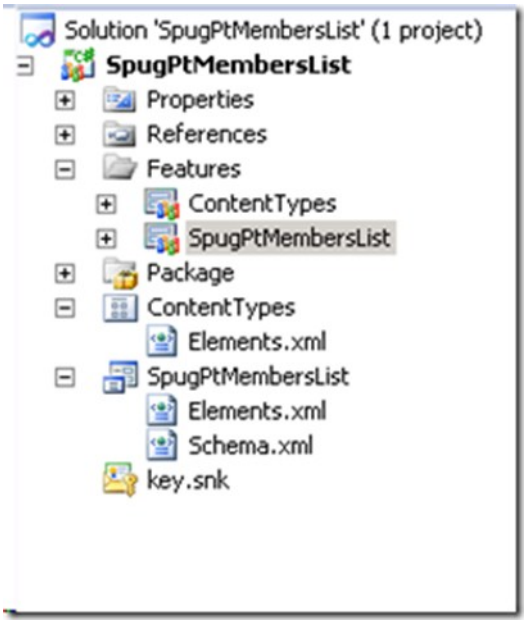
Editar na window "**Solution Explorer**" o nome da "**Feature1**" para "**SpugPtMembersList**"

4. Finalmente, remover o elemento de "List Definition" da feature antiga "**ContentTypes**"

Seleccionar a feature "**ContentTypes**" , e com o botão do lado direito, seleccionar a opção "**View Designer**"

Seleccionar Item SpugPtMembersList da caixa **"Items in the Feature"** e passá-lo para **"Items in Solutions ",**seleccionando a opção (seta) destinada a esse efeito **(<)**

No final teremos algo deste género …

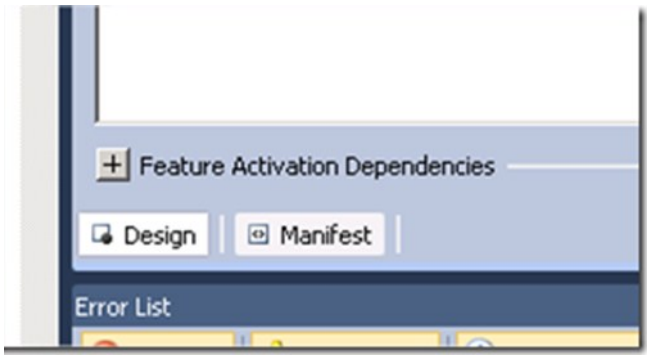


Feature Activation Dependencies

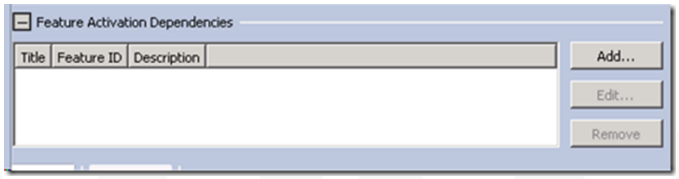
Para garantir essa operacionalidade, vou utilizar o conceito de dependência entre features.

1. Seleccionar a feature "**SpugPtMembersList**" , e com o botão do lado direito, seleccionar a opção "**View Designer**"

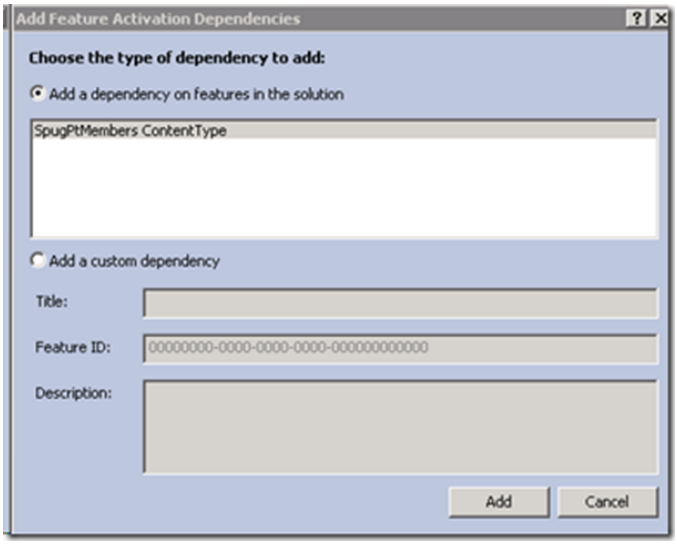
2. Seleccionar a opção "**Feature Activation Dependencies**", existente no canto inferior esquerda feature "**SpugPtMembersList**"



3. Ao seleccionar essa opção, esta expande-se e devemos, seleccionar a opção "**Add…**"



4. Seleccionar **"SpugPtMembers ContentType"** e novamente a opção **"Add"**



Neste momento temos a garantia que a feature que instala a definição de lista não é despoletada antes da feature de content types. :)

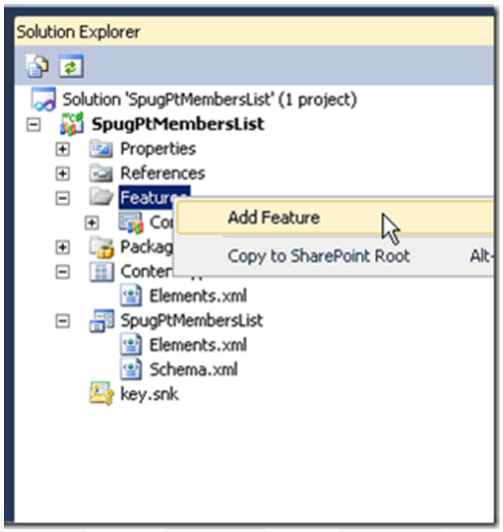
Lista de Logs

Este artigo objectiva a criação de um mecanismo de logging de operações na lista de membros da SpugPt de forma a que possamos auditar de alguma forma o provisionamento de informação.

Assim , precisamos de uma lista complementar para provisionar os logs propriamente disto:

1. Vamos criar uma nova feature **SpugPtLogs**

Seleccionar "**Features" ,** clicar com o botão direito do rato para abrir o menu de contexto e seleccionar **"Add Feature"**



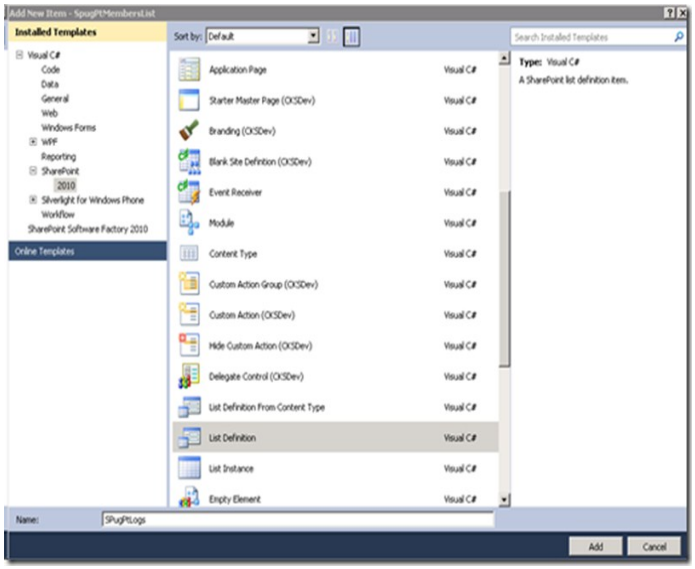
2. Caracterizar a feature :

- Title**: SpugPtLogs**
- Description**: Criar lista de provisionamento de logs**
- Scope**: Site**

1. Seleccionar o projecto com o botão direito do rato e seleccionar a opção **Add>NewItem**

2. Seleccionar o item **"List Definition"** e o digitar o nome **"SpugPtLogs"** para este item.

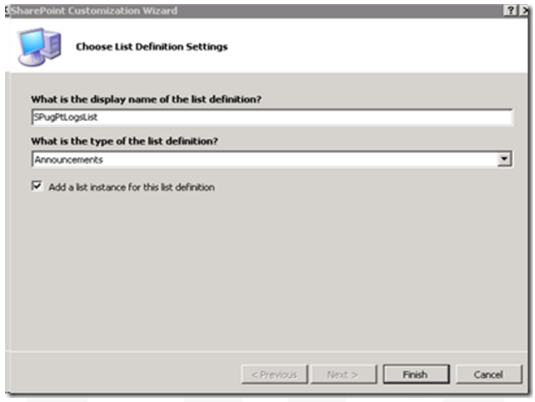
3. Seleccionar o botão "**Add**"



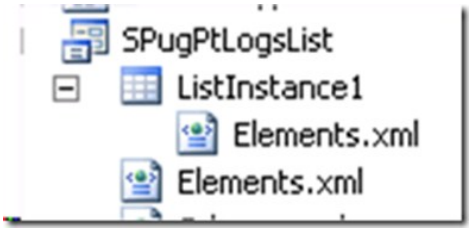
4. No ecrã seguinte , digitar em "**What is the display name…",** o seguinte texto "**SPugPtLogsList**"

5. Seleccionar em "**What is the type…",** a opção "**Custom List**"

6. Seleccionar o botão **"Finish"**



Em termos da estrutura da nova definição de lista temos:



1. Renomear ListInstance1 para Logs

```
<ListInstance Title="Logs"
    OnQuickLaunch="TRUE"
    TemplateType="10001"
    Url="Lists/Logs"
    Description="Lista de Logs (SpugPt)">
</ListInstance>
</Elements>
```

2. Dentro de Logs, editar o Elements.xml para que fique desta forma (alterar só os elementos a **bold**)

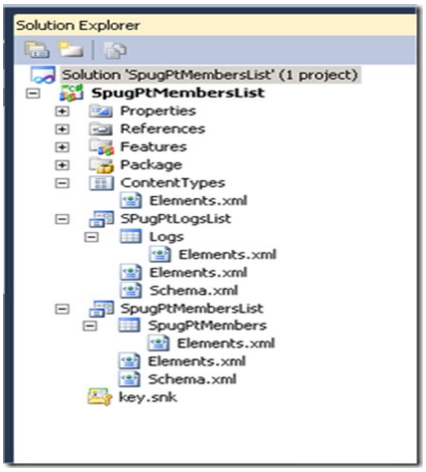
```
<ListTemplate
  ="" Name="SPugPtLogsList" =""
  ="" Type="10001" =""
  ="" BaseType="0" =""
  ="" OnQuickLaunch="TRUE" =""
  ="" SecurityBits="11" =""
  ="" Sequence="410" =""
  ="" DisplayName="SPugPtLogsList" =""
  ="" Description="SPugPtLogsList Definition" =""
  ="" Image="/_layouts/images/itgen.png"/>
</Elements>
```

3. Editar o outro Elements.xml visível para que fique desta forma (alterar só os elementos a **bold**)

Boa práctica #2: Criar e caracterizar a feature antes da implementação da funcionalidade em si.

Como já criamos a feature todos estes novos elementos são automaticamente associados a esta feature. Definir a estrutura **primeiro**, só depois as implementações.

Estrutura do projecto Visual Studio nesta fase:



Com a lista de logs criada, passemos ao sistema de logging em si.

Event Receivers

Uma das funcionalidades associadas às listas de SharePoint, que apresenta um grande potencial e acrescenta mais valor às nossas soluções na plataforma, é a de podemos criar e associar **Event Handlers** a objectos de provisionamento de informação do SharePoint.

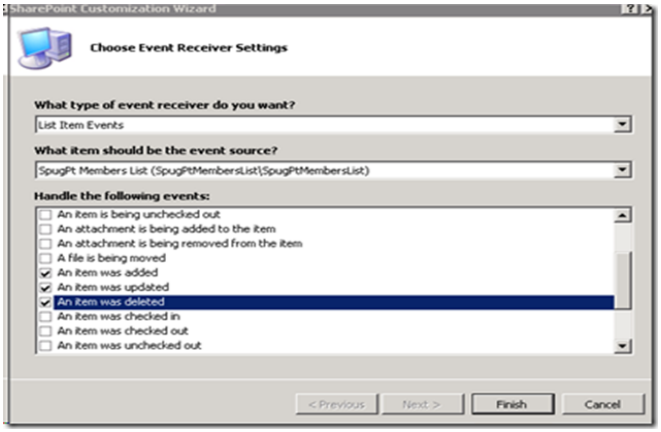
A ideia essencial deste mecanismo é despoletarmos algum tipo de funcionalidade quando alguma operação é inferida no conteúdo provisionado por estes objectos.

1. Seleccionar elemento **SpugPtMemberLists** e adicionar novo item

2. Seleccionar template "EventReceiver" e o digitar o nome **"MembersListEventReceiver"** para este item

3. A título de exemplo, só vamos fazer log de itens que são adicionados, actualizados ou eliminados da lista de membros.

Desta forma, seleccionar os eventos respectivos como demonstra a figura abaixo, e seleccionar "**Finish**".



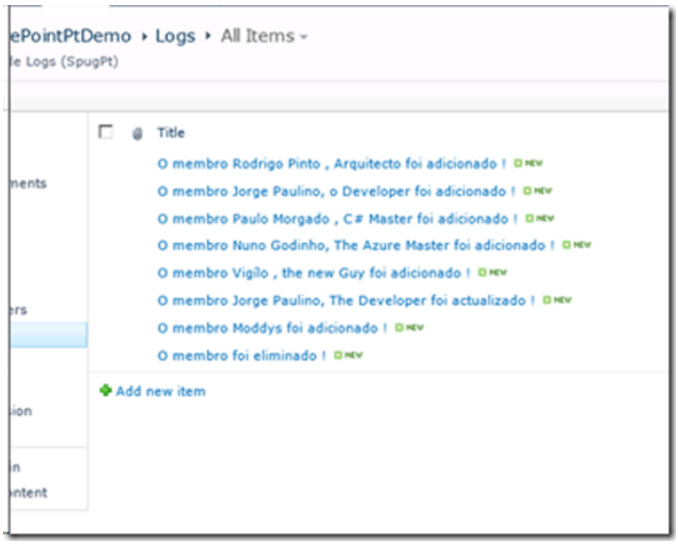
4. Teremos algo deste género:

```
/// <summary>
/// List Item Events
/// </summary>
public class MembersListEventReceiver :
             SPItemEventReceiver
{
    /// <summary>
    /// An item was added.
    /// </summary>
    public override void ItemAdded
    (SPItemEventProperties properties)
    {
        base.ItemAdded(properties);
    }
    /// <summary>
    /// An item was updated.
    /// </summary>
    public override void ItemUpdated
    (SPItemEventProperties properties)
    {
        base.ItemUpdated(properties);
    }
    /// <summary>
    /// An item was deleted.
    /// </summary>
    public override void ItemDeleted
     (SPItemEventProperties properties)
    {
        base.ItemDeleted(properties);
    }
}
```

5. Substituir a classe **MembersListEventReceiver** na sua totalidade, pelo código abaixo

```
/// <summary>
    /// List Item Events
    /// </summary>
    public class MembersListEventReceiver :
      SPItemEventReceiver
    {
        SPListItem itm = null;

        /// <summary>
        /// An item was added.
        /// </summary>
       public override void ItemAdded
        (SPItemEventProperties properties)
        {
            base.ItemAdded(properties);
            itm = properties.ListItem;
            LogOperation(properties, itm.Title, "
                                 foi adicionado !");
        }
        /// <summary>
        /// An item was updated.
        /// </summary>
        public override void ItemUpdated
         (SPItemEventProperties properties)
        {
            base.ItemUpdated(properties);
            itm = properties.ListItem;
            LogOperation(properties, itm.Title, "
                                foi actualizado !");
        }
        /// <summary>
        /// An item was deleted.
        /// </summary>
        public override void ItemDeleted
        (SPItemEventProperties properties)
        {
            base.ItemDeleted(properties);
            LogOperation(properties, "", " foi apa
                                gado !");
        }
 public void LogOperation(SPItemEventProperties
    properties, string memberName, string message)
     {
       //Valida se a lista Logs existe
       SPList lst = properties.Web.Lists.TryGetList
                    ("Logs");
       if (lst == null)
        throw new Exception("Lista de Logs não pro
                            visionada");
      //Se existir regista a operação
      SPListItem itm = lst.AddItem();
      itm["Title"] = String.Concat("O membro ",
                           memberName, message);
       itm.Update();
      }
// [BEST PRACTICES]>>>
// Desactiva o trigger dos eventos , para que
// possamos fazer as nossas operações
// Após as nossas operações, activa de novo o
```

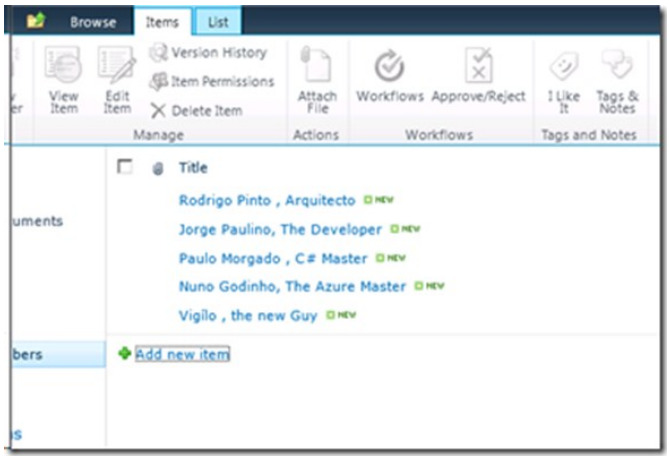
```
// trigger

 bool oldValue;
  public MembersListEventReceiver()
   {
    this.oldValue = base.EventFiringEnabled;
    base.EventFiringEnabled = false;
   }
  #region IDisposable Members
  public void Dispose()
   {
    base.EventFiringEnabled = oldValue;
   }
  // <<<[BEST PRACTICES]
}
```



Por cada operação de inserção, actualização e eliminação de um item de uma lista, a plataforma despoleta várias vezes a actualização desse item, levando a que os registo de logging sejam inseridos mais do que uma vez aquando cada operação inferida nos registos da lista de membros.

Como tal, o código indicado, como **[BEST PRACTICES]** complementa esta funcionalidade na perfeição

Assim, ao inserirmos registos na lista de membros….



temos os respectivo log de operações…. :)

Conclusão

Embora tenha sido mais uma vez um exemplo muito simples, espero que tenha atingido o objectivo.

Próximo artigo: Webparts funcionalidades e afins

NOTA: Neste artigo está espelhada numa das imagens uma funcionalidade que não foi detalhada aqui.

Quem for o mais atento e mais rápido a descobrir e tiver interesse, terei todo o prazer em divulgar um roadmap de sucesso para iniciação de implementações desta natureza. ;) Enviem os vossos palpites para spugpt@gmail.com.

Resta-me fazer-vos um convite: todos os meses aos 2ª Sábados, a Comunidade Portuguesa de SharePoint apresenta 2 sessões e chalk talks com membros activos na tecnologia onde podemos trocar ideias, esclarecer dúvidas e acima de tudo trocar experiências.

Apareçam e espero que tenha sido útil. Aquele abraço com muitos Dispose à mistura

AUTOR

Escrito por Rodrigo Pinto

SharePoint MVP , Architect, Evangelist, SharePoint Portuguese User Group (SPUGPT) Founder

Com 12 anos de experiência em Software Engineering & Architecture, é SharePoint Specialist, Evangelist na Indra. Tem experiência nas diversas áreas do SharePoint ,destacando-se as áreas de object model, implementação de soluções, e Web Content Management com conteúdos media Evangelista acérrimo de um roadmap de best pratices na plataforma SharePoint assente em rigor e criatividade , procura disponibilizar estas provas de conceito pela comunidade, clientes e parceiros. É o fundador da Comunidade Portuguesa de SharePoint.(www.sharepointpt.org)

CICLO
Formação e Consultoria

A encaminhar e direccionar carreiras no caminho da excelência na informática

CURSOS EM DESTAQUE

O QUE JÁ FEZ ESTE ANO PARA AUMENTAR A SUA EMPREGABILIDADE?
Invista em si e no seu CV, faça cursos de formação e recicle os seus conhecimentos!

Descontos especiais para membros da comunidade Portugal-a-Programar!

http://ciclo.pt/protocolos/portugal-a-programar

Edifício Bela Vista Office - Estrada de Paço de Arcos, 66 – Salas 1-16A e 1-16B - 2735-336 Cacém - Portugal • http://ciclo.pt • Tel.: +351 214 096 930 • e-mail: contacto@ciclo.pt

Biztalk Server - Princípios Básicos dos Mapas (2)

Os mapas, ou transformações, são um dos componentes mais comuns nos processos de integração. Funcionam como tradutores essenciais no desacoplamento entre os diferentes sistemas a interligar. Este artigo tem como objectivo explicar como os mapas são processados internamente pelo motor do produto à medida que exploramos o editor de mapas do BizTalk Server.

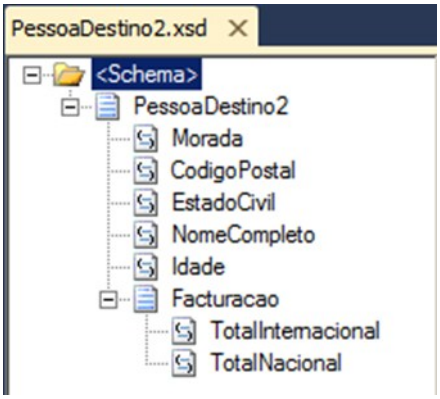
Este artigo tem como base o exemplo do artigo publicado na 32ª edição da Revista PROGRAMAR onde é explicado em detalhe as funcionalidades básicas dos mapas e como podem ser implementadas. Pretende ser uma nota introdutória e destinada a quem está a dar os primeiros passos nesta tecnologia.

Conforme explicado no artigo anterior, quando estamos a efectuar uma transformação de mensagens são 5 as funcionalidades comuns que normalmente nos surgem:

- Mapeamento simples de um determinado valor (cópia directa)
- Concatenação de valores
- Selecções condicionadas
- Scripts customizados
- Adicionar novos dados

Tendo por base estas funcionalidades vamos explicar como o motor de mapas do BizTalk traduz e processa estas ligações internamente.

Para melhor compreendermos o seu funcionamento efectuamos algumas alterações à estrutura do esquema (schema) do documento final: acrescentamos um elemento opcional ("EstadoCivil") e desorganizamos intencionalmente a estrutura do documento.



Modelo de processamento dos Mapas

Embora tradicionalmente a informação seja extraída da origem à medida que vai sendo processada, na realidade nos modelos baseado em XSLT o que acontece é exactamente o contrário: Cada elemento no destino provoca a procura pelo correspondente na origem. Vejamos um exemplo tradicional:

- A origem é percorrida de inicio ao fim do ficheiro;
- A informação é extraida da origem na ordem exacta que é encontrada;
- As regras de mapeamento são construidas à medida que a origem é percorrida.

O BizTalk também utiliza esta técnica nas conversões dos ficheiros texto (Flat Files) para formato XML (transformações de sintaxe, também explicado no artigo anterior), no entanto as transformações nos mapas utilizam uma abordagem diferente, como iremos verificar mais à frente neste artigo.

Um dos factores importantes quando utilizamos ferramentas de integração é também termos em atenção as tecnologias standards existentes, e foi isso o que os criadores do produto fizeram. O BizTalk trabalha internamente, quase exclusivamente, com documentos XML, sendo que fazia sentido utilizarem uma tecnologia standard para efectuarem este tipo de transformações, para isso o W3C definiu o XSLT (Extensible Stylesheet Language Transformation) como o formato padrão para representar as transformações entre documentos XML.

Desta forma todas as ligações e functoids que são visiveis graficamente na grelha de mapeamentos não são mais do que uma representação gráfica de um documento XSLT que permite transformar o documento de origem num determinado formato especificado pelo esquema de destino.

Biztalk Server - Princípios Básicos dos Mapas (2)

Podemos dizer que os mapas de BizTalk têm sempre o seu foco no documento final, fazendo assim sentido que as regras de transformação sejam processadas na sequencia requerida para o criar. Quando o mapa é compilado, essas regras são traduzidas em queries XPATH e funções XSLT por forma a transformar a informação pretendida, ou seja, as regras de mapeamento são construidas a partir da estrutura de destino e não da origem como algumas ferramentas tradicionais.

Sendo assim, os mapas de BizTalk seguem o seguinte modelo:

- O motor de mapeamento do BizTalk percorre a estrutura de destino do inicio ao fim;
- As regras de mapeamento são construidas e executadas conforme os links são encontrados na estrutura de destino;
- A informação é extraida da origem quando um link é encontrado na estrutura de destino.

Nota: Podemos utilizar um XSLT criado por uma aplicação externa e incluí-lo no mapa através de XSLT custom script ou importando um ficheiro XSLT que efectua a transformação total do mapa (obviamente o editor gráfico do mapa de BizTalk não representará as ligações visualmente).

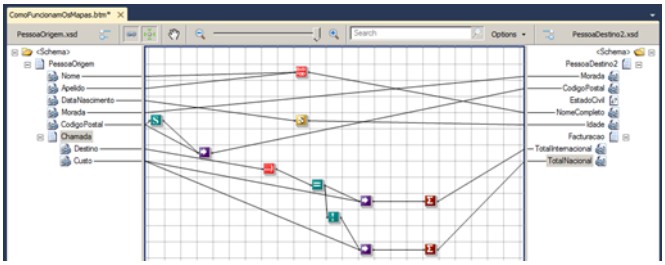
Desconstruindo um mapa

Neste artigo vamos utilizar as operações básicas de mapeamento descritas anteriormente, e analisar as decisões tomadas pelo "compilador" de mapas do BizTalk Mapper Designer. Basicamente neste mapeamento existem dois esquemas similares, no qual pretendemos mapear os elementos da origem no seu correcto destino e ao qual implementamos os seguintes desafios:

- Concatenar o Nome e Apelido por forma a obtermos o nome completo (concatenação de valores);
- Mapear a Morada no seu elemento de destino (mapeamento simples de um determinado valor)
- Transformar a data de nascimento em idade (scripts customizados);
- O Código Postal só deverá ser mapeado se tiver uma string valida, caso contrário não será mapeado (selecções condicionadas);
- O elemento Estado Civil é opcional e como não temos elementos para o mapear, o mesmo deverá ser ignorado (adicionar novos dados).
- Adicionalmente, iremos efectuar uma lógica de mapeamento mais avançado para calcular os totais de chamadas internacionais e nacionais utilizando ciclos, selecções condicionas e operações matemáticas.

Conforme podem verificar, intensionalmente trocamos a ordem dos elementos no esquema de origem, por forma a verificarmos com mais facilidade como os mapas de BizTalk funcionam.

Desta forma obtivemos o seguinte mapa final:



Nota: A ordem pelo qual efectuamos as ligações entre os elementos da origem para os elementos de destino, neste cenário, não tem qualquer importância para o compilador, no entanto o mesmo já não se pode dizer para as functoids. As functoids necessitam de determinados parâmetros de entrada que podem variar sendo a ordem de entrada importante.

A imagem anterior do mapa de BizTalk é na realidade a seguinte representação do documento XSLT: https://gist.github.com/1597897

Com base neste documento vamos seguir o comportamento do compilador. O que ele efectua é, traduzir as ligações existentes no mapa à medida que os encontra enquanto percorre o esquema de destino:

- O primeiro elemento encontrado é "Morada", como tem uma ligação associada, a mesma é traduzido por expressão XPath ("Morada/text()") que define o elemento a extrair da origem:

```
<Morada>
   <xsl:value-of select="Morada/text()" />
</Morada>
```

- O segundo elemento encontrado é o "CodigoPostal" que também ele tem uma ligação associada. Trata-se de uma selecção condicionada que será traduzida para uma condição XSLT (xsl:if):

```
        <xsl:variable name="var:v1" se-
        lect="userCSharp:LogicalIsString(string
        (CodigoPostal/text()))" />
<xsl:if test="string($var:v1)='true'">
```

```
        <xsl:variable name="var:v2"
                select="CodigoPostal/text()" />
          <CodigoPostal>
            <xsl:value-of select="$var:v2" />
          </CodigoPostal>
        </xsl:if>
```

- O terceiro elemento é "EstadoCivil", uma vez que não tem nenhuma ligação associada, o mesmo é ignorado no mapeamento.
- O quarto elemento processado é "NomeCompleto", este elemento tem uma ligação associada, que corresponde ao valor $var:v3 que é construído a partir da concatenação dos vários inputs e a execução da função userCSharp:StringConcat que visualmente era o ***String Concatenate Functoid***:

```
        <xsl:variable name="var:v3"
        select="userCSharp:StringConcat(string
        (Nome/text()) , &quot; &quot; , string
        (Apelido/text()))" />
        <NomeCompleto>
           <xsl:value-of select="$var:v3" />
        </NomeCompleto>
```

- O quinto elemento é o "Idade", uma ligação é encontrada o que significa que irá ser efectuado o mapeamento do script customizado que estava dentro do **CSharp Inline Scripting Functoid**:

```
        <xsl:variable name="var:v4" se-
        lect="userCSharp:CalcularIdade(string
        (DataNascimento/text()))" />
        <Idade>
         <xsl:value-of select="$var:v4" />
        </Idade>
```

- Por fim é encontrado o nó (record) "Facturacao" com elementos: "TotalInternacional" e "TotalNacional". De realçar que apesar de não termos definido nenhum ciclo através da functoid "Loop" o compilador e suficientemente inteligente para perceber que estamos a tratar de um ciclo e traduzi-lo correctamente, no entanto irá criar para cada um dos elementos um ciclo próprio. Para uma melhor optimização seria necessário utilizarmos um script customizado.

```
        <Facturacao>
          <xsl:variable name="var:v5"
      select="userCSharp:InitCumulativeSum(0)" />
          <xsl:for-each
               select="/s0:PessoaOrigem/Chamada">
            <xsl:variable name="var:v6"
select="userCSharp:StringLeft(string(@Destino) ,
&quot;4&quot;)" />
            <xsl:variable name="var:v7"
select="userCSharp:LogicalEq(string($var:v6) ,
&quot;+351&quot;)" />
            <xsl:if test="string($var:v7)='true'">
              <xsl:variable name="var:v8"
select="@Custo" />
              <xsl:variable name="var:v9"
select="userCSharp:AddToCumulativeSum(0,string
($var:v8),&quot;1000&quot;)" />
            </xsl:if>
          </xsl:for-each>
          <xsl:variable name="var:v10"
select="userCSharp:GetCumulativeSum(0)" />
          <TotalInternacional>
            <xsl:value-of select="$var:v10" />
          </TotalInternacional>
…
</Facturacao>
```

A sequência das ligações

"A ordem com que as ligações são associadas no destino tem um grande impacto no resultado final."… Esta afirmação é verdadeira e ao mesmo tempo falsa!

Na realidade a ordem com que associamos as ligações (Drag&Drop) dos elementos de origem para diferentes elementos de destino é irrelevante, uma vez que o compilador, conforme explicado anteriormente, irá processar pela ordem correcta… Excepto se tivermos de associar diversas ligações para o mesmo elemento de destino ou functoid. Nestes dois últimos casos a ordem com que se efectua a associação é extremamente importante, podendo originar resultados inesperados.

Impacto da ordem das ligações nas functoids

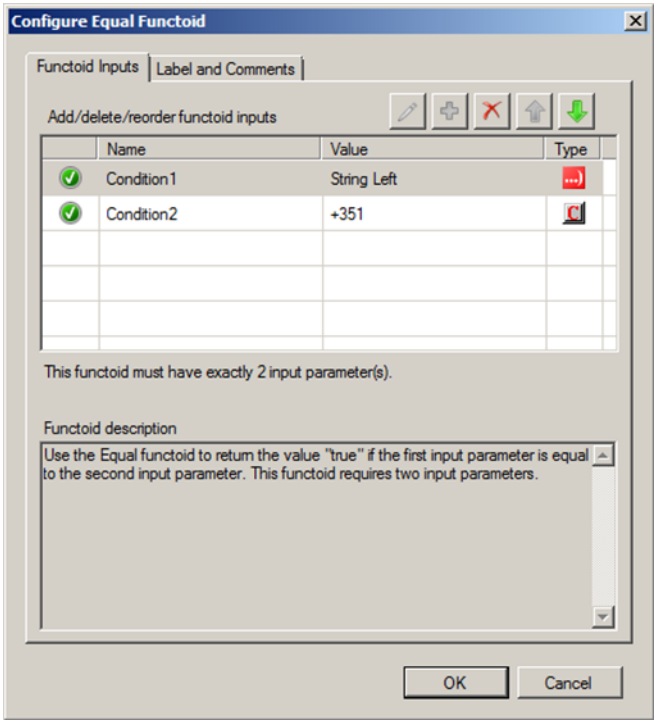
Uma grande parte das functoids existentes na Toolbox espera vários parâmetros de entrada, um exemplo prático é a functoid "Value Mapping Functoid".

Esta functoid retorna o valor do segundo parâmetro se o valor do primeiro for igual a "True", caso contrário não é retornado nada. Desta forma é necessário respeitar a ordem com que associamos as ligações:

- A primeira ligação a associar a esta functoid terá de enviar um valor booleano (true/false)
- O segundo será o valor que queremos retornar na functoid.

A troca na associação das ligações irá originar erros de mapeamento ou em resultados inesperados, conforme a functoid utilizada.

Reorganização das ligações (links) nas functoids é muito fácil, para isso basta abrir o detalhe (duplo clique) e usar os botões de ordenação.

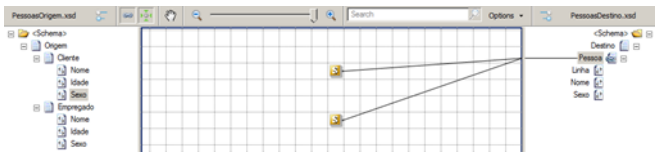


Impacto da ordem das ligações nos elementos no esquema de destino

Alterar a ordem da sequência na associação das ligações no mesmo elemento do esquema de destino poderá também ter impacto no resultado final pretendido.

Infelizmente, quando associamos diferentes ligações no mesmo elemento, não existe nenhuma forma ou local no editor gráfico onde podemos verificar a ordem da associação, à semelhança do que acontece com as functoids. A única forma de verificarmos a ordem nestes casos é inspeccionar o código XSLT gerado ou testando o mapa.

Um bom exemplo deste cenário é quando associamos duas Scripting functoid com diferentes inline XSLT scripts ao mesmo destino, uma vez mais a troca na associação poderá ter resultados inesperados.



Neste exemplo a 1ª "Scripting functoid" contêm o seguinte código XSLT:

```
<xsl:for-each select="Cliente">
  <Pessoa>
    <Nome><xsl:value-of select="Nome/text()" /></
Nome>
    <Sexo><xsl:value-of select="Sexo/text()" /></
Sexo>
  </Pessoa>
</xsl:for-each>
```

Este código efectua o mapeamento dos todos os elementos existentes no nó "Clientes" no esquema de origem, para os elementos no nó "Pessoa" no esquema de destino.

A segunda "Scripting functoid" contêm um código XSLT idêntico (https://gist.github.com/1598161) mas desta vez irá mapear todos os elementos existentes no nó "Empregado" no esquema de origem, para os elementos no nó "Pessoa" no esquema de destino.

O resultado expectável no documento final é aparecerem todos os clientes no nó "Pessoa" e de seguida todos os empregados. Se trocarmos a ordem da associação das ligações no esquema final, iremos verificar que o resultado também ele irá aparecer trocado.

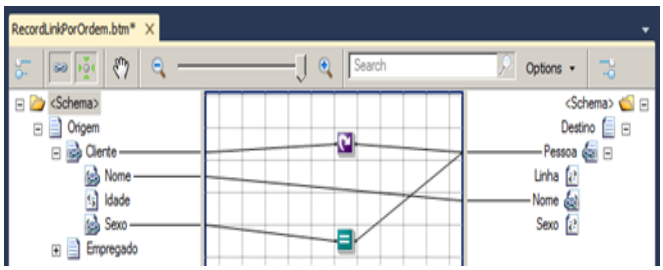
Podemos validar o resultado deste cenário nos seguintes links:

- Mensagem original:
  https://gist.github.com/1598182
- Resultado expectado:
  https://gist.github.com/1598172
- Resultado se trocarmos a ordem da associação:
  https://gist.github.com/1598188

A excepção à regra da sequência das ligações

Em resumo, o motor de mapas processa as "regras" percorrendo o esquema de destino do início ao fim, processando as ligações pela ordem que os encontra e em caso de múltiplas ligações num determinado elemento ou functoid, as mesmas são processados pela ordem de associação. Isto significa que as ligações associadas aos nós pais são processadas antes das ligações associadas aos filhos.

Um exemplo deste cenário é o uso de condições no nó pai quando pretendemos condicionar o resultado final segundo uma determinada condição. Vamos então procurar todos os nomes dos clientes do sexo feminino. Para isso iremos criar um mapa com as seguintes configurações:

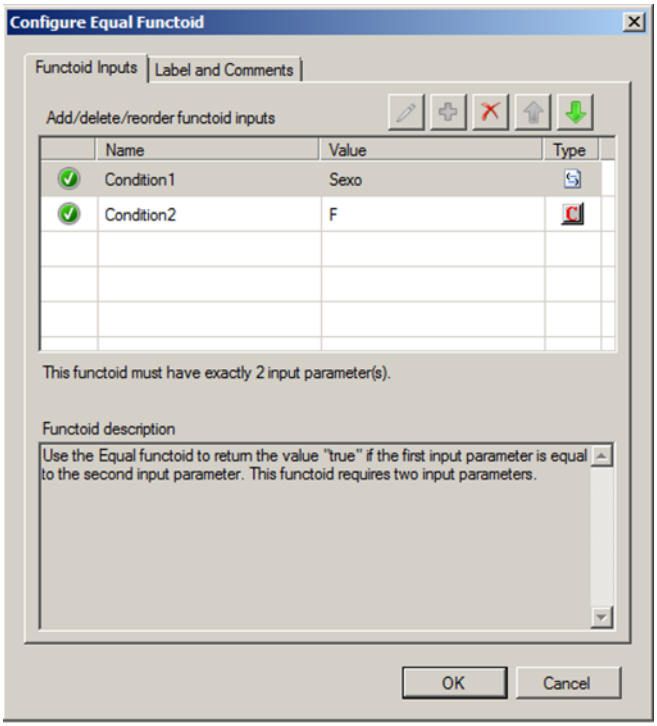


Adicionar a "Looping Functoid", associando o nó de origem "Cliente" ao nó de destino "Pessoa"

Adicionar uma "Equal Functoid" e associar o elemento de origem "Sexo" do nó "Cliente" à functoid e de seguida a functoid ao nó de destino "Pessoa"

Editar a "Equal Functoid" e editar a segunda condição com o

valor “F” para construirmos uma equação equivalente a Sexo=”F”:



- Ligar o elemento de origem “Nome” do nó “Cliente” ao elemento de destino “Nome” do nó Pessoa

A “Equal Functoid” vai retornar o valor “True” se o elemento “Sexo” for igual a “F”, caso contrário será retornado o valor “False”. O que origina que o nó “Pessoa” só é mapeado se o valor retornado for igual a “True”, obtendo assim a condição que pretendíamos.

Se verificarmos o código gerado,

```
…
<xsl:template match="/s0:Origem">
   <ns0:Destino>
      <xsl:for-each select="Cliente">
         <xsl:variable name="var:v1" se-
lect="userCSharp:LogicalEq(string(Sexo/text()) ,
"F")" />
         <xsl:if test="$var:v1">
            <Pessoa>
               <Nome>
                  <xsl:value-of select="Nome/text
()" />
               </Nome>
            </Pessoa>
         </xsl:if>
      </xsl:for-each>
   </ns0:Destino>
</xsl:template>
```

Iremos verificar que a primeira acção do mapa, após o ciclo que percorre os vários elementos do nó é: obter o valor gerado na “Equal Functoid”, representada na variável “v1”;

Sendo a segunda operação validar a condição (IF) do valor da primeira operação (v1), ou seja, vai testar se o valor de “v1” é “True”. Se a condição for verdadeira o código dentro da condição é executado, caso contrário irá passar para o próximo elemento sem executar nada. Obtendo assim o output desejado:

```
<ns0:Destino xmlns:ns0="http://
ComoFuncinamOsMapas.Schema2">
        <Pessoa>
                <Nome>Elsa Ligia</Nome>
        </Pessoa>
</ns0:Destino>
```

Excepção: Processar ligações fora de ordem

No entanto existe uma importante excepção a esta regra de sequência, especialmente quando utilizamos scripts customizados nos elementos recursivos.

Um bom exemplo deste cenário é a utilização de scripts de incremento de contadores. Podemos ilustrar este cenário, adicionando duas “Scripting Functoids” ao mapa:

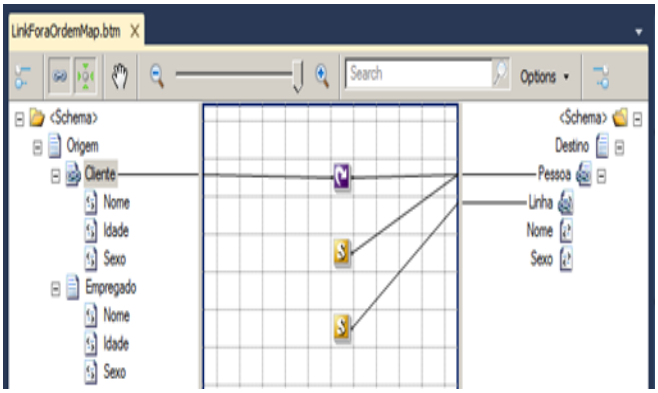
- A primeira contendo a inicialização e incremento do contador;

```
int contador = 0;
public void IncrementarContador()
{
   contador += 1;
}
```

- A segunda obtendo o valor do contador

```
public int RetornarContador()
{
     return contador;
}
```

Nota: Este exemplo estará associado a um ciclo (Loop), ou elemento recursivo.



Seria esperado que no primeiro ciclo o resultado do segundo script fosse o valor “1”, no segundo ciclo obtivéssemos o valor “2” e assim sucessivamente. No entanto, se testarmos

o mapa vamos verificar que a realidade é diferente:

```
<ns0:Destino xmlns:ns0="http://
ComoFuncinamOsMapas.Schema2">
     <Pessoa><Linha>0</Linha></Pessoa>
     <Pessoa><Linha>1</Linha></Pessoa>
     <Pessoa><Linha>2</Linha></Pessoa>
</ns0:Destino>
```

Conforme podemos verificar no resultado em cima, a sequência com que as ligações são executadas é:

- Criar o nó (elemento) “Pessoa”;
- Criar os elementos filhos e executar as ligações associadas aos mesmos:
    - Executar a função “RetornarContador” que irá retornar o valor “0” na primeira iteração.
- Executar as ligações associadas ao nó pai:
    - Executar a função “IncrementarContador”.

Como podemos atestar verificando o código XSL produzido pelo mapa:

```
…
<xsl:template match="/s0:Origem">
  <ns0:Destino>
    <xsl:for-each select="Cliente">
      <Pessoa>
        <xsl:variable name="var:v1"
         select="userCSharp:RetornarContador()" />
        <Linha>
          <xsl:value-of select="$var:v1" />
        </Linha>
        <xsl:variable name="var:v2"
       select="userCSharp:IncrementarContador()" />
        <xsl:value-of select="$var:v2" />
      </Pessoa>
    </xsl:for-each>
  </ns0:Destino>
</xsl:template>
```

Claro que podemos alterar o código existente nas “Scripting Functoids” por forma a contornamos este comportamento e obter assim o resultado pretendido. No entanto este exemplo serve para alertar que, em alguns cenários, especialmente no uso de scripts customizados nos nós recursivos, é necessário verificar e validar a sequência em que estes são executados.

Código Fonte

Todo o código fonte utilizado neste artigo encontra-se disponível no MSDN Code Gallery para download:

- Funcionalidades básicas dos mapas de BizTalk

Conclusão

Com este artigo exploramos alguns mapeamentos comuns associados aos mapas, tentando desmontar as opções que a máquina tomou para cumprir com a intenção original do mapa visual.

Quando começarem a explorar o mundo dos mapas, existem duas questões que devem avaliar com maior atenção:

Qual a melhor forma para resolver um problema: garantidamente existem várias abordagens para resolver um determinado problema. Muitas vezes decidir a melhor acaba por ser o mais difícil. Compilar e analisar o código gerado pode ser um bom princípio para começar a conhecer o impacto de determinadas opções.

Testes incrementais: muita das vezes caímos na tentação de tentar resolver um determinado problema de transformação de início ao fim e só depois de finalizarmos é que vamos testar a solução. Deixar para o final pode tornar extremamente difícil detectar problemas em mapeamentos complexos. Limitar o âmbito dos testes deverá ser um processo contínuo e incremental durante a criação de mapas devendo ser efectuados logo que seja completado um significativo bloco de trabalho.

Espero que este tipo de Hacking ajude a entender o comportamento e as técnicas de debugging elementares para este tipo de problemas. Como todas as áreas em constante maturação, acompanhar os diferentes autores online é provavelmente a forma mais natural para ir descobrindo novos padrões e respectivas soluções.

AUTOR

Escrito por Sandro Pereira
Actualmente Senior Software Developer na empresa DevScope. É Microsoft Most Valuable Professional (MVP) em Microsoft BizTalk. O seu principal foco de interesse são as tecnologias e plataformas de Integração (EAI): BizTalk e SOAP / XML / XSLT e Net, que utiliza desde 2002. É um participante bastante activo nos fóruns da Microsoft (MSDN BizTalk Server Forums), contribuidor no MSDN Code Gallery e autor do Blog: http://bit.ly/oFLwB4- Twitter: @sandro_asp – Membro da comunidade BizTalk Brasil: http://bit.ly/9NI7ie

Windows® Phone

Put people first.

Basta a tua ideia.

Nós ajudamos-te a torná-la realidade.

Com as ferramentas gratuitas de criação de apps para o Windows Phone vais poder, finalmente, dar vida à ideia que não te sai da cabeça, 'aquela' app que todos vão querer.

www.app-me-up.com

Microsoft

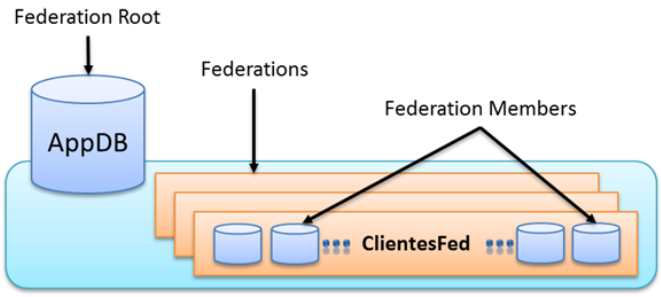
SQL Azure Federations na prática

Na edição passada introduzimos de forma teórica as SQL Azure Federations, vamos neste artigo realizar um pequeno exemplo prático e analisar algumas dicas de utilização das Federations nas nossas aplicações.

As SQL Azure Federations são a implementação do padrão de *sharding*. *Sharding* é um padrão que permite aumentar a escalabilidade e a performance de grandes bases de dados. Aplicar o padrão a uma base de dados significa "partir" essa base de dados em pedaços mais pequenos e distribui-los por vários servidores de modo a obter escalabilidade. A cada pedaço resultante chamamos de shard.

A base de dados raiz (root) poderá ter várias federações. Cada federação poderá ter vários membros (*shards*) e cada membro poderá ter várias unidades atómicas. A escalabilidade máxima é atingida quando todas as unidades atómicas estão contidas em apenas um membro da federação, ou seja, cada membro de uma federação tem apenas uma unidade atómica.

Cada membro da federação é suportado por uma instância de base de dados SQL Azure. Uma unidade atómica é a unidade mínima indivisível que poderemos ter dentro dos membros e representa uma instância da federation key (chave da federação).



Exemplo

Vamos assumir que o leitor já dispõe de uma instância de servidor SQL Azure. Caso não tenha pode muito facilmente aceder ao portal de Windows Azure em https://windows.azure.com/default.aspx e criar um novo servidor na sua subscrição. Não se esqueça de adicionar o seu endereço de IP actual nas regras da *firewall*.

Vamos agora estabelecer ligação à base de dados master do servidor para podemos criar a nossa base de dados de demonstração. Todos os servidores de SQL Azure dispõem de uma base de dados com o nome master. Esta base de dados é responsável, entre outras coisas, por armazenar os *logins* e permissões e é apenas sobre ela que podemos executar os comandos de CREATE, ALTER ou DROP de bases de dados e utilizadores. Como vamos criar uma nova base de dados para executar os exemplos deste artigo teremos que efectuar uma ligação a essa base de dados.

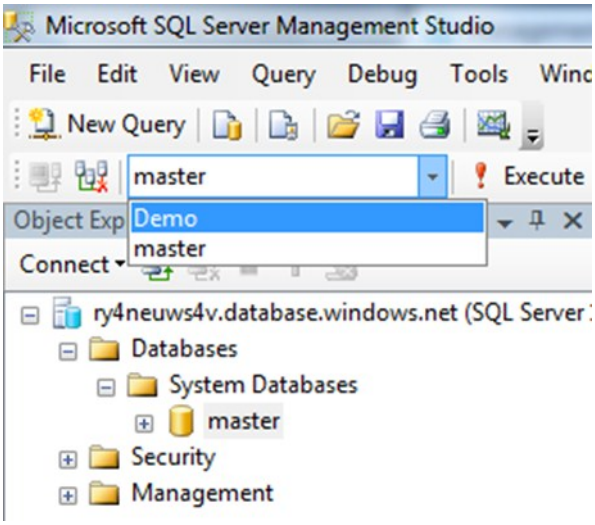
Neste exemplo estamos a utilizar um servidor de base de dados vazio pelo que a ligação irá ser estabelecida por omissão sobre a base de dados master. Caso o utilizador tenha já alguma base de dados criada no servidor deverá nas opções do SQL *Server Management Studio* (SSMS) indicar que pretende ligar à base de dados master sob a pena de a ligação ser criada sobre uma base de dados já existente e não conseguirá criar a nova.

Criar esta base de dados no portal seria muito mais fácil, bastava adicioná-la ao servidor pretendido mas, de forma a dar mais alguma experiência aos nossos leitores relativamente a ligações a base de dados em SQL Azure optou-se por explicar estes detalhes.

Para os menos experientes, alerta-se ainda para o facto de que o login terá que estar no formato *<login>@<server>* como pode verificar na imagem seguinte:

Feita a ligação vamos começar por a base de dados onde vamos criar as nossas *federations* através do comando CREATE DATABASE Demo. Após este passo, torna-se necessário mudar a ligação da base de dados master para a base de dados Demo que acabámos de criar.

Note que o comando USE DEMO irá produzir o erro *"USE statement is not supported to switch between databases. Use a new connection to connect to a different Database."* Sendo necessário desligar a ligação e voltar a ligar escolhendo como base de dados Demo ou simplesmente mudando a ligação através da lista de bases de dados disponíveis do SSMS.



Vamos agora criar o esquema.

No artigo de introdução falámos de dados centralizados. Dados centralizados são dados sobre os quais não existem muitas operações de leitura e escrita e que estão disponíveis e que apenas estão disponíveis numa única localização.

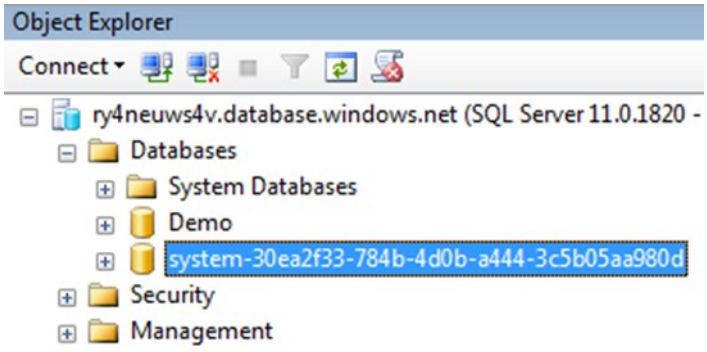
Neste exemplo vamos criar uma tabela Loja que irá ficar alojada apenas na nossa base de dados raiz sendo por isso considerado dados centralizados.

```
CREATE TABLE Loja(
LojaID INT PRIMARY KEY,
Nome NVARCHAR(100))
```

Até aqui nada de novo, vamos então dar inicio à criação da nossa primeira federação. Para tal iremos executar o seguinte comando:

```
CREATE FEDERATION ProdutosFed (ID BIGINT RANGE)
```

Como vimos no artigo de introdução, será criada uma nova base de dados para alojar esta federação e inicialmente todos os seus membros. Após a execução do comando anterior poderemos logo verificar que a nova base de dados já foi criada.



Antes de criar o *schema* da nossa federação teremos que mudar a ligação para a nova base de dados.

Essa mudança de ligação faz-se através do comando:

```
USE FEDERATION ProdutosFed(ID = 0) WITH RESET, FIL-
TERING=OFF
```

O comando USE FEDERATION é próprio do SQL Azure mas fornece ao utilizador uma experiência de utilização muito semelhante ao comando USE do SQL Server.

Caso o leitor se esteja a questionar o porquê do ID=0 , neste momento a federação tem apenas um membro pelo que qualquer ID será válido para realizar as operações pretendidas.

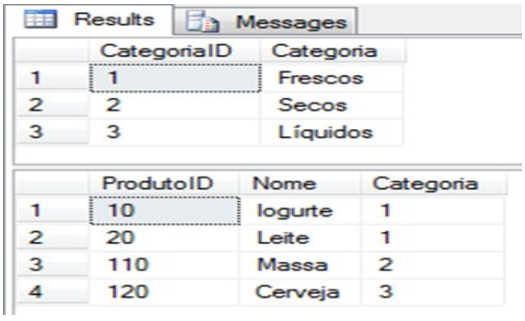
Vamos agora criar o esquema das tabelas da federação:

```
CREATE TABLE Categoria(
CategoriaID INT PRIMARY KEY,
Categoria NVARCHAR(100))

CREATE TABLE Produto(
      ProdutoID BIGINT PRIMARY KEY,
      Nome NVARCHAR(200),
      Categoria INT REFERENCES Categoria
(CategoriaID),
) FEDERATED ON (ID=ProdutoID)
```

A primeira tabela será uma tabela de dados de referência, que será replicada por todos os eventuais membros da federação, porque não tem o atributo FEDERATED ON. Em oposição, os dados da tabela Produto serão considerados dados particionados porque já dispõe do atributo FEDERATED ON.

Realizando algumas inserções nas tabelas poderemos verificar que neste momento ambas as tabelas contêm todos os dados.

```
INSERT INTO Categoria VALUES (1,'Frescos')
INSERT INTO Categoria VALUES (2,'Secos')
INSERT INTO Categoria VALUES (3,'Líquidos')

INSERT INTO Produto VALUES (10,'Iogurte',1)
INSERT INTO Produto VALUES (20,'Leite',1)
INSERT INTO Produto VALUES (110,'Massa',2)
INSERT INTO Produto VALUES (120,'Cerveja',3)

SELECT * FROM Categoria;
SELECT * FROM Produto;
```



	CategoriaID	Categoria
1	1	Frescos
2	2	Secos
3	3	Líquidos

	ProdutoID	Nome	Categoria
1	10	Iogurte	1
2	20	Leite	1
3	110	Massa	2
4	120	Cerveja	3

SQL Azure Federations na prática

Estamos agora em condições de realize o primeiro particionamento dos dados e iremos realizar essa operação através dos comandos:

```
USE FEDERATION Root WITH RESET
GO
ALTER FEDERATION ProdutosFed SPLIT AT(ID=100)
```

O primeiro comando serve para mudarmos a ligação para a base de dados raiz (Demo). Os comandos de SPLIT e DROP são sempre executados sobre a base de dados raiz porque é esta a base de dados que contém os metadados que suportam as operações.

O segundo comando é o comando que efectivamente ordena o particionamento dos dados.

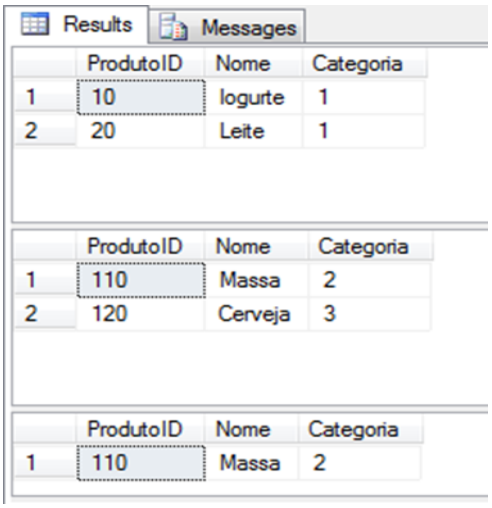
Com vimos no artigo de introdução, o divisão de uma *federation* em duas resulta na criação de duas novas bases de dados e na cópia filtrada dos dados para as respectivas bases de dados novas. O tempo que esta operação demora é proporcional à dimensão dos dados a copiar mas a informação mais importante nesta fase será a de que não é possível iniciar novo particionamento dos dados sem que o anterior tenha terminado.

Para monitorizar a existência de algum processo de SPLIT a ocorrer podemos fazer uso dos metadados presentes na base de dados raiz:

```
select * from sys.dm_federation_operations
```

Após a operação de SPLIT estar completa termos agora os dados particionados em duas bases de dados. Para verificar essa situação vamos realizar algumas consultas e observar os resultados:

```
-- #1

USE FEDERATION ProdutosFed(ID = 10) WITH RESET,
FILTERING=OFF
GO
SELECT * FROM Produto;
GO

-- #2

USE FEDERATION ProdutosFed(ID = 110) WITH RESET,
FILTERING=OFF
GO
SELECT * FROM Produto;
GO

-- #3

USE FEDERATION ProdutosFed(ID = 110) WITH RESET,
FILTERING=ON
GO
SELECT * FROM Produto;
```



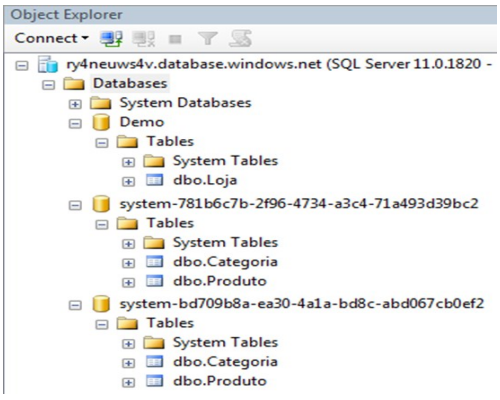
O primeiro conjunto de comandos efectua uma ligação à 1ª parte dos dados, ou seja, todos os produtos com ID inferior a 100.

O segundo comando efectua uma ligação à 2ª parte dos dados (produtos com ID igual ou superior a 100).

O terceiro comando usa a opção FILTERING = ON de modo a que a ligação seja filtrada e desta forma só se consiga aceder aos dados do ID que foi passado como parâmetro. As aplicações *multitenant* poderão tirar muito partido desta opção.

Cada uma das bases de dados particionadas terá uma cópia completa da tabela Categoria dado que se tratam de dados de referência. As consultas a esta tabela são ainda independentes do ID e do FILTERING. Caso se pretenda alterar os dados presentes nesta tabela terá que ter em atenção que terá que realizar essa alteração em todos os membros da federação (em todas as bases de dados).

Como seria de esperar, temos neste momento 3 bases de dados neste servidor, uma base de dados raiz que suporta os dados centralizados e duas base de dados que suportam, no nosso exemplo, metade dos dados federados cada uma.



Já falámos nos *metadados* que estão presentes na base de dados raiz. Através de vistas sobre esses *metadados* poderemos retirar algumas informações tais como as federações presentes, as distribuições de cada federação, os membros que cada federação tem e os respectivos limites, etc.

```
SELECT * FROM sys.federations
SELECT * FROM sys.federation_distributions
SELECT * FROM sys.federation_member_distributions
```

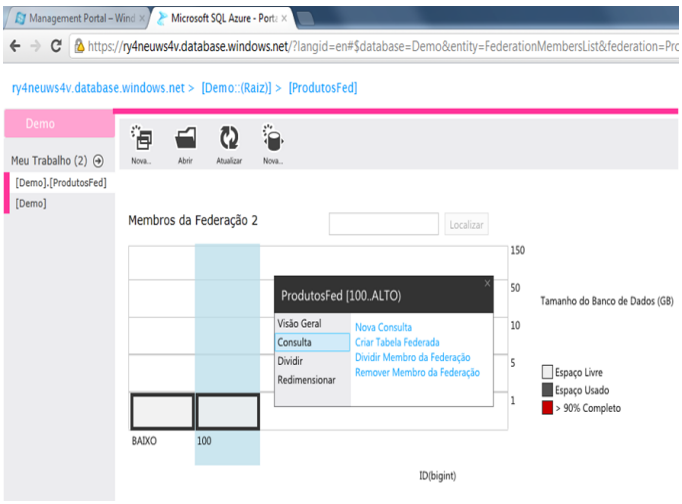
Results | Messages

	federation_id	name
1	65536	ProdutosFed

	federation_id	distribution_name	distribution_type	system_type_id	max_length	precision
1	65536	ID	RANGE	127	8	19

	federation_id	member_id	distribution_name	range_low	range_high
1	65536	65537	ID	-9223372036854775808	100
2	65536	65538	ID	100	NULL

Todas estas operações podem ser realizadas através do SQL Azure Management Portal mas, como já referimos, é também objectivo dar mais alguma sensibilidade ao utilizador sobre o que realmente se passa "lá dentro".



Regras

Vamos agora verificar quais as regras para a correcta construção de uma base de dados com federações.

Criação da Federação

```
CREATE FEDERATION nome_federacao (nome_distribuicao
<tipo> RANGE)
```

Para criar uma federação usamos o comando CREATE FEDERATION e além do nome da federação passamos 3 parâmetros, nome da distribuição, tipo de dados e tipo de distribuição.

O nome da federação e tem que ser único dentro de uma base de dados SQL Azure.

O nome da distribuição também é o identificados que iremos utilizar para referir a chave da federação quando, por exemplo, utilizarmos o comando CREATE TABLE … FEDERATED ON(...) ou o comando USE FEDERATION.

O tipo de dados terá que ser INT, BIGINT, UNIQUEIDENTIFIER ou VARBINARY(n) onde o n poderá ir até 900.

Neste momento RANGE é a única opção de particionamento disponível.

Federated Tables

As tabelas particionadas (*federated tables*) são as tabelas que têm os seus dados divididos por vários membros. Estas tabelas são criadas dentro das federações através da adição do atributo FEDERATED ON aquando da criação das mesmas.

```
CREATE TABLE Produto(…)FEDERATED ON (ID=ProdutoID)
```

É necessário passar como parâmetro no atributo qual a coluna desta tabela que representa a chave da federação. Relativamente e esta coluna temos as seguintes regras:

- O tipo de dados desta coluna tem que ser obrigatoriamente igual ao tipo de dados da chave da federação.
- Não pode ter valor nulo.
- Não pode ser actualizada para valores fora do range do membro actual.
- Não pode ser uma *computed column*.
- Tem que fazer parte de todas as *unique* e *clustered* indexes.
- Tem que fazer parte de todas as chaves estrangeiras para outras *federated tables*.

Sendo necessário indicar uma coluna como sendo a coluna que representa a chave da federação podemos concluir que todas as tabelas particionadas têm que conter a chave da federação. Esta regra implica que, em algumas tabelas, tenhamos que proceder a uma desnormalização do esquema.

Relativamente a esquema, temos ainda as seguintes regras:

- Todas as chaves estrangeiras entre *federated tables* têm que conter a chave da federação.
- Não podem existir *reference tables* com chaves estrangeiras para uma *federated table*.

- Uma *federated table* pode ter chaves estrangeiras para *reference tables* sem restrições.

Os membros da federação não suportam:

- Index Views
- Colunas Identity
- Tipo de dados *timestamp* e *rowversion*

Note ainda que não é obrigatório que os membros da federação tenham todos o mesmo esquema, cada membro pode ter o "seu" esquema desde que respeite as regras. Só poderão ser realizadas alteração ao esquema de um membro através de ligações sem filtro (FILTERING=OFF).

Reference Tables

São tabelas sobre os quais apenas existem operações de leitura (as escritas são possíveis mas raras) e que por questões de performance estão disponíveis em todos os membros da federação. Os seus dados são copiados integralmente durante as operações de SPLIT.

Estas tabelas são criadas dentro dos membros da federação através de CREATE TABLE sem o atributo FEDERATED ON. No exemplo anterior a tabela Categoria é uma *reference table.*

Em relação a restrições tenha em atenção que não podem existir chaves estrangeiras para tabelas particionadas (*federated tables*) e que as limitações relativamente a *identity* e *timestamp* se mantêm também para *reference tables*.

Só poderão ser realizadas alteração a dados ou ao esquema de uma referece table através de ligações sem filtro (FILTERING=OFF).

Central Tables

As tabelas centrais são tabelas que existem apenas na base de dados raiz. São tabelas sobre as quais não existem muitas operações de leitura ou escrita, normalmente são tabelas de metadados ou configurações.

Como não são tabelas que fazem parte das operações de SPLIT não se aplicam as restrições que vimos para os outros tipos de tabelas. Note apenas que os dados destas tabelas não estão visíveis quando estamos conectados a um membro de uma federação. Só poderemos aceder a estes dados através de uma ligação directa à base de dados raiz.

Utilização nas aplicações

Podemos dividir a utilização de SQL Azure Federations em dois casos.

O primeiro e mais simples, acontece quando a carga de trabalho incide sobre uma unidade atómica da federação. Neste caso basta apenas executar o comando USE FEDERATION para estabelecer a ligação ao membro correcto e a partir daí podemos fazer uma utilização como se de uma base de dados tradicional se tratasse.

```
USE FEDERATION ProdutosFed(ID = 1) WITH RESET,
FILTERING=ON
```

No caso em que é necessária a participação de dados de mais do que um membro da federação o comportamento já terá que ser diferente. Neste momento as *fan out queries* não são suportadas pelo que teremos que realizar estas consultas individualmente a todos os membros da federação e agregar os resultados na aplicação.

Mas mesmo neste caso a resolução do problema não é muito complicada. Podemos tirar partido da vista sys.federation_member_distributions para obter a lista de membros da federação e quais os seus limites inferiores e superiores.

Results | Messages

	federation_id	member_id	distribution_name	range_low	range_high
1	65536	65537	ID	-9223372036854775808	100
2	65536	65539	ID	100	110
3	65536	65540	ID	110	NULL

Note que o limite máximo superior é representado por NULL. Tenha em conta ainda que é apenas na tabela raiz que consegue obter todos os membros e as suas distribuições. Em cada membro da federação esta vista também está disponível mas tem apenas resultados para o próprio membro.

LINQ

Como vimos anteriormente terá que ser emitido um USE FEDERATION antes da primeira consulta à base de dados. Em LINQ para SQL existe um pormenor, temos que abrir a ligação antes de emitir o USE FEDERATION.

```
using (DataClasses1DataContext db = new
DataClasses1DataContext())
{
  db.Connection.Open();
  db.ExecuteCommand("USE FEDERATION ProdutosFed(ID
  = 110) WITH RESET, FILTERING=OFF");
  var lista = from p in db.Produtos select new
   { ID = p.ProdutoID, Produto = p.Nome };
dataGridView1.DataSource = lista;
}
```

Entity Framework

A utilização em Entity Framework tem o mesmo requisito, teremos que abrir a ligação antes de emitir o USE FEDERATION.

```
using (DemoEntities db = new DemoEntities())
{
 db.Connection.Open();
 string federationCmdText = @"USE FEDERATION
  ProdutosFed(ID = 110) WITH RESET, FILTERING=OFF";
  db.ExecuteStoreCommand(federationCmdText);
```

```
var lista = from p in db.Produto select new { ID =
  p.ProdutoID, Produto = p.Nome };
 dataGridView1.DataSource = lista;
}
```

Tem ainda outro requisito, as SQL Azure Federations não suportam MARS (*Multiple Active Result Sets*). Devido a esta situação teremos que modificar a *connection string* com multipleactiveresultsets=False;

```
<add name="DemoEntities" connectionString="(…)
multipleactiveresultsets=False;" (…) />
```

A inexistência de suporte a MARS (Multiple Active Result Sets) é um assunto importante que iremos ver mais à frente.

Entity Framework Code First

Em Entity Framework Code First a abertura da ligação é feita de forma ligeiramente diferente porque ObjectContext utiliza um tipo de ligação diferente.

```
((IObjectContextAdapter)
db).ObjectContext.Connection.Open();
```

Transacções

Relativamente a transacções temos que ter ainda em conta que o *TransactionScope* só deverá ser instanciado depois de emitido o USE FEDERATIONS de modo a que a transacção seja estabelecida sobre o membro da federação.

```
using (DemoEntities db = new DemoEntities())
{
db.Connection.Open();
string federationCmdText = @"USE FEDERATION
ProdutosFed(ID = 110) WITH RESET, FILTERING=OFF";
using (TransactionScope scope = new
TransactionScope
(TransactionScopeOption.RequiresNew))
{
//(...)
}
}
```

Multiple Active Result Sets (MARS)

Actualmente não existe suporte para Multiple Active Result Sets em SQL Azure Federations.

Isto significa que não poderemos usar algumas características que os ORMs (*Object-relational mappin*) nos fornecem actualmente e que são suportadas pelo MARS. São elas *lazy loading, LoadProperty()* e *Load()*.

Em alternativa deveremos fazer *eager loading*. *Lazy loading* trata-se de apenas inicializar os objectos no momento em que estes são necessários. *Eager loading* é o contrário de *lazy loading*, ou seja, iniciar os objectos no momento em que são criados.

Em LINQ podemos fazer *eager loading* através do método DataLoadOptions.LoadWith<T>() de modo a que todos os resultados sejam obtidos da base de dados no momento da instanciação.

Em Entity Framework, o *eager loading* é realizado através do método *Include*:

```
DataLoadOptions dataLoadOptions = new DataLoadOp-
tions();
dataLoadOptions.LoadWith<Produto>(c =>
c.Categoria);
db.LoadOptions = dataLoadOptions;
```

Outra opção é utilizar o método ToList() ou ToArray() antes de um ciclo de modo a que os resultados possam ser logo

```
var produtos = from p in db.Produto.Include
("Categoria1")  select p;
```

trazidos para memória no momento da construção. Neste caso teremos que ter a opção *LazyLoading* activa.

Conclusão

```
var produtos = (from p in db.Produto select
p).ToList();
foreach (Produto produto in produtos)
{
Categoria categoria = produto.Categoria1;
}
```

Esperamos que com este artigo o leitor tenha aprofundado mais os seus conhecimentos de SQL Azure Federations e que se sinta mais confiante na hora de abordar o uso desta tecnologia.

Referências

Your Data in the Cloud - http://bit.ly/fnxTNn
Windows Azure Customer Advisory Team - http://bit.ly/qV4G1n
MSDN Library - http://bit.ly/wlHmkP

AUTOR

Escrito por Vítor Tomaz
Consultor independente na área das tecnologias de informação. Tem especial interesse por cloud computing, programação concorrente e segurança informática. É membro de algumas comunidades tais como Portugal-a-Programar, NetPonto, AzurePT, HTML5PT e GASP.

Veja também as edições anteriores da Revista PROGRAMAR

32ª Edição - Dezembro 2011

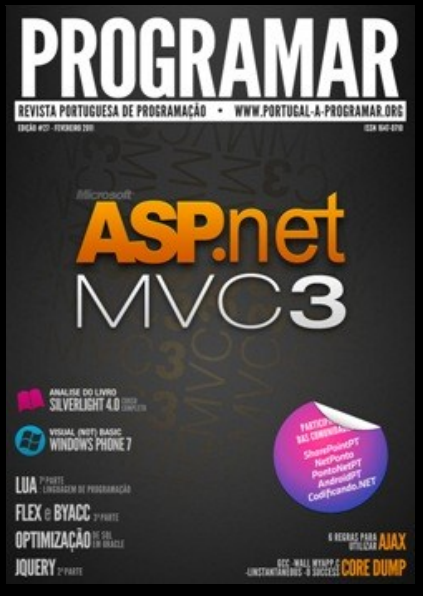
31ª Edição - Outubro 2011

30ª Edição - Agosto 2011

29ª Edição - Junho 2011

28ª Edição - Abril 2011

27ª Edição - Fevereiro 2011



e muito mais em ...

www.revista-programar.info

XIX Semana da Informática do IST

Para quem ainda não conhece, a Semana Informática do IST (SINFO), é um evento organizado desde 1993 pelos alunos dos cursos de licenciatura/mestrado em Engenharia Informática e Computadores do Instituto Superior Técnico, e tem como objetivo principal "aproximação dos alunos de Informática ao mundo tecnológico-empresarial que os rodeia, tanto no panorama nacional como internacional".

Esta iniciativa, que se realiza entre os dias **27 de Fevereiro e 2 de Março de 2012**, e que este ano atinge a sua XIX edição, conta com um elenco de luxo de personalidades ligadas à informática ou às tecnologias, onde se pode destacar **Richard Matthew Stallman,** fundador do movimento free software, do projeto GNU, e da Free Software Foundation ("Fundação para o Software Livre") – cuja vinda é inteiramente patrocinada pela Ordem dos Engenheiros; **João Damas**, um dos guardiões das 14 chaves do sistema de nomes de domínio (DNS); **Nuno Subtil Engenheiro de Software da NVIDIA**; **Zeinal Bava** CEO da Portugal Telecom; **Mirko Gozzo** European General Manager na Riot Games, empresa bastante conhecida pelo jogo League of Legends; **Paulo Taylor,** cofundador do Ebuddy, e vencedor do prémio de Empreendedorismo Inovador da Diáspora Portuguesa em 2009; **Celso Martinho**, Board member na PT Inovação, Director Tecnologia de Produto na Portugal Telecom, CTO e Co-Fundador na SAPO e **James Portnow**, CEO da Rainmaker Games e escritor da Webseries Extra-Credits.

Estarão também presentes nas SINFO instituições e empresas como a Ordem dos Engenheiros, a VILT, a Microsoft, a Altitude Software, a ANSOL, a Novabase, a Deloitte, a EDP, a Polícia Judiciária, o Millenium BCP, a Hitachi Consulting, a CapGemini, a Logica, a Indra, a Accenture, a Rumos - entre outras.

Segundo a organização do evento, a SINFO é primariamente direcionada aos alunos de Informática do Instituto Superior Técnico e outras faculdades. No entanto, dada a grande abrangência de temas tecnológicos e atividades, o público-alvo alarga-se aos restantes alunos da instituição e a qualquer indivíduo ou empresa que se interesse pelos temas abordados, incluindo professores, empresas ou simples curiosos. Sendo por isso um evento direcionado para todos os interessados nesta matéria.

O evento irá decorrer no campus da Alameda do Instituto Superior Técnico, e abrange dois edifícios em simultâneo, o Pavilhão Central e o Pavilhão de Civil, que alberga um anfiteatro com lotação para 300 pessoas e duas salas de showroom com capacidade para 50 e 70 pessoas, respetivamente.

Cada dia, terá um tema, assim:

- 2ª-feira 27 de Fev.: Jogos e Multimédia
- 3ª-feira 28 de Fev.: Inteligência Artificial e Robótica
- 4ª-feira 29 de Fev.: Software na Sociedade
- 5ª-feira 01 de Mar.: O Futuro da Web
- 6ª-feira 02 de Mar.: Leic++: Investigação e Empreendorismo Para o último dia ficou reservado o tema LEIC++: Investigação e Empreendedorismo.

A comunidade Portugal-a-Programar associa-se a esta iniciativa como media partner.

DUVIDAS?

IDEIAS?

AJUDAS?

PROJECTOS?

portugal-a-programar.org