PROGRAMAR

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

EDIÇÃO #26 -DEZEMBRO 2010

ISSN 1647-0710

MICROSOFT

LINQ

IMPLEMENTAÇÃO DOS OPERADORES TAKELAST, TAKELASTWHILE SKIPLAST E SKIPLASTWHILE

PROGRAMAÇÃO FUNCIONAL COM PERL

INTRODUÇÃO ENTITY FRAMEWORK 4.0 JQUERY

LUA 6ª PARTE : LINGUAGEM DE PROGRAMAÇÃO

FLEX e BYACC

OPTIMIZAÇÃO DE SQL EM ORACLE

ASPECTOS LOW-LEVEL DE 3D

PARTICIPAÇÃO DAS COMUNIDADES SharePointPT NetPonto

E SE AMANHA A APPLE DOMINAR? MAC ADDRESS

QUANDO SE É ! PRODUTIVO CORE DUMP

EDITORIAL

EQUIPA PROGRAMAR

Coordenadores

António Silva

Fernando Martins

Editor

António Silva

Design

Sérgio Alves (@scorpion_blood)

Redacção

André Lage, Augusto Manzano, Diogo Constantino, Fernando Martins, João Mares, João Matos, Jorge Paulino, Paulo Morgado, Pedro Aniceto, Ricardo Rodrigues, Ricardo Trindade, Sara Silva

Contacto

revistaprogramar@portugal-a-programar.info

Website

http://www.revista-programar.info

ISSN

1647-0710

INDICE

Sempre em movimento…

Porque sabemos que parar é morrer, nesta edição da Revista PROGRAMAR introduzimos várias alterações que achamos importantes para continuidade deste projecto. Das várias alterações introduzidas, a salta imediatamente à vista, mesmo ao leitor que ainda está somente a ler a página do editorial, é a **mudança de design**. Em qualquer tipo de publicação o design influencia o modo como vemos como lemos. Aqui temos que agradecer a todos quantos ajudaram a adaptar este novo design, mas especialmente ao Sérgio Alves por toda a parte da idealização e desenho do novo design.

Outra alteração, que todavia não será tão notada pelos leitores é a alteração do software de paginação. Chegamos à conclusão que o Scribus já evoluiu bastante desde a última vez que foi utilizado para paginar a Revista PROGRAMAR e está à altura das nossas necessidades. Assim decidimos que não havia necessidade de continuarmos a utilizar um software pago e que trazia vários problemas de leitura a alguns utilizadores (dos quais chegamos a receber comentários).

Também fizemos várias **parcerias com algumas comunidades portuguesas** que estão de algum modo relacionadas com programação, de modo a obter mais e melhores artigos para si. Todavia não estamos fechados a mais parcerias, continuamos abertos a futuros contactos para o nosso endereço de correio electrónico, que serão posteriormente analisados por toda a equipa.

Trazemos também até si, **colunas de opinião e colunas direccionadas** a um tema específico. Assim poderá contar que surgirão artigos desse assunto com uma periodicidade específica na sua Revista favorita.

Por fim, achamos que estamos prontos para voltar a trazer a **Revista PROGRAMAR até si de 2 em 2 meses**, e por isso a próxima edição deverá sair em Fevereiro ao invés de Março como estava programado.

Tudo isto para si, que está ler. Esperámos que as alterações sejam bem recebidas por todos, mas também esperámos opiniões sobre todas estas alterações, mesmo porque as novidades poderão não ficar por aqui, contámos ter mais novidades na próxima edição. Por isso não tenha medo de nos contactar para revistaprogramar@portugal-a-programar.org a dizer o que acha de todas estas alterações. Porque você é importante para nós, quer as suas opiniões, quer o seu trabalho a divulgar o nosso projecto aos seus amigos e colegas, e mais importante a sua preferência que nos permite crescer para lhe oferecer algo maior e melhor.

A equipa da Revista PROGRAMAR

Upload Lisboa e Upload Lisboa Pro 2010

O Upload Lisboa é um evento que por si só nomeia a era em que estamos. A era da nova comunicação, da troca de informação no momento, de forma prática e natural, e da partilha de conhecimentos em rede. O Upload Lisboa pretende, mais uma vez, passar a esfera da rede interior e global para a esfera da partilha conjunta, física.

Este evento nasce da necessidade crescente de debate e partilha de conhecimentos sobre a web 2.0, numa altura em que as novas plataformas de comunicação e informação assumem um lugar cada vez mais central na vida de pessoas e empresas. Hoje, mais que nunca, a partilha através das plataformas sociais na web ocupam um lugar de destaque na influência na opinião pública e na formação de critérios de análise ao que se passa no país e no mundo.

São dois dias, 11 e 15 de Dezembro, que certamente não irá perder!

Mais informações: http://uploadlisboa.com/

Seminário PocketPT sobre tecnologias Windows Phone

Encontram-se abertas as inscrições para o 6º Seminário PocketPT sobre tecnologias móveis Windows Phone. Este ano o seminário será dedicado à nova plataforma móvel da Microsoft, e irá incluir várias sessões que irão dar a conhecer as funcionalidades do mesmo.

Será ainda uma excelente oportunidade para os participantes testarem e verem em primeira mão os novos equipamentos.

Como tem vindo a acontecer nos anos anteriores a inscrição este ano é gratuita e irá ter lugar no auditório da Microsoft Portugal no TagusPark em Porto Salvo no próximo dia 18 de Dezembro pelas 10h.

Mais informações e inscrições:
http://www.mtechseminar.com/2010/index.htm

SAPO CodeBits 2010

De 11 a 13 de Novembro de 2010 decorreu o Sapo Codebits 2010. A comunidade Portugal-a-Programar esteve presente, tendo alguns membros efectuado alguns projectos. Este ano, com oferta da comunidade, todos os que participaram e solicitaram, tiveram direito a uma t-shirt para usar durante o evento.

Esta foi a foto de grupo de alguns dos elementos que estiveram presentes (infelizmente não foi possível juntar todos):

16a Reunião Presencial da Comunidade NetPonto

No dia 11/12/2010 será realizada a 16ª reunião presencial da comunidade NetPonto, em Lisboa. Nesta sessão, poderá ver duas apresentações presenciais sobre Domain Driven Design e WP7 & XNA – Let's play?

Para participar, basta efectuar a inscrição através do site http://netponto-lisboa-dezembro-2010.eventbrite.com/ A entrada é gratuita e são realizadas reuniões todos os meses.

Para mais informações, consulta de próximas sessões e inscrição: http://netponto.org/

LINQ: Implementação dos operadores TakeLast, TakeLastWhile, SkipLast e SkipLastWhile

Há algum tempo necessitei de obter os últimos itens de uma sequência que satisfaziam um determinado critério e, olhando para os operadores disponíveis na classe Enumerable, apercebi-me de que tal operador não existia.

A única forma de obter o resultado pretendido era inverter a ordem dos itens na sequência, obter os itens que satisfaziam o critério e inverter os itens da sequência resultante.

Algo como isto:

```
Sequence
  .Reverse()
      .TakeWhile(criteria)
           .Reverse();
```

Parece simples, certo? Primeiro chamamos o método Reverse para produzir uma nova sequência com os mesmos itens mas na ordem inversa da sequência original, depois chamamos o método TakeWhile para obter os primeiros itens que satisfazem o critério e, finalmente, chamamos novamente o método Reverse para recuperar a ordem original dos itens.

O problema desta aproximação é que o método rev memoriza toda a sequência antes de iterar sobre os seus itens na ordem inversa – e o código acima usa-o duas vezes. Isto significa iterar sobre todos os itens da sequência original e memoriza-los, iterar sobre os primeiros itens que satisfazem o critério e memoriza-los e, finalmente, iterar sobre estes itens para produzir a sequência final.

Se estiveram a contar, chegaram à conclusão de que se iterou sobre todos os itens da sequência original uma vez e mais três vezes sobre os itens que satisfazem o critério. Se a sequência original for grande, isto pode consumir muita memória e tempo.

Existe ainda um problema adicional quando se usa a variante que usa o índice de item na sequência original na avaliação do critério de selecção (>). Quando se inverte a ordem dos itens, os índices destes serão invertidos e o predicado de avaliação do critério de selecção terá de ter isso em consideração, o que poderá não ser possível se não se souber o número de itens na sequência original.

Tinha de haver uma solução melhor, e é por isso que eu implementei os Operadores Take Last:

TakeLast<TSource>(IEnumerable<TSource>)

Retorna o número especificado de elementos contínuos no fim da sequência.

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85
};

   var topThreeGrades = grades
                .OrderBy(grade => grade)
                .TakeLast(3);

Console.WriteLine("As    notas    mais    altas
são:");

foreach (int grade in topThreeGrades)
{
        Console.WriteLine(grade);
}
```

```
/*
Este código produz o seguinte resultado:
As notas mais altas são:

98
92
85
*/
```

TakeLastWhile<TSource>(IEnumerable<TSource>, Func<TSource, Boolean>)

Retorna os elementos do fim da sequência para os quais a condição especificada é verdadeira.

```
string[] fruits =
{ "maçã", "maracujá","banana", "manga",
  "laranja", "mirtilo", "uva", "morango"
};

var query = fruits
       .TakeLastWhile(fruit =>
       string.Compare("laranja",
       fruit, true) != 0);

foreach (string fruit in query)
{
       Console.WriteLine(fruit);
}

/*
Este código produz o seguinte resultado:
maracujá
uva
morango
*/
```

TakeLastWhile<TSource>(IEnumerable<TSource>, Func<TSource, Int32, Boolean>)

Retorna os elementos do fim da sequência para os quais a condição especificada é verdadeira.

```
string[] fruits =
{ "maçã", "maracujá","banana", "manga",
  "laranja", "mirtilo", "uva", "morango" };

var query = fruits
       .TakeLastWhile((fruit, index) =>
       fruit.Length >= index);

foreach (string fruit in query)
{
       Console.WriteLine(fruit);
}

/*
Este código produz o seguinte resultado:
morango
*/
```

Tendo introduzido os operadores TakeLast, faz todo o sentido introduzido os seus duais: os operadores SkipLast:

SkipLast<TSource>(IEnumerable<TSource>)

Retorna todos os elementos da sequência à excepção do número especificado de elementos contínuos no fim da sequência.

```
int[] grades =
       { 59, 82, 70, 56, 92, 98, 85 };
var lowerGrades = grades
       .OrderBy(g => g)
       .SkipLast(3);

Console.WriteLine("Todas as notas excepto as
top 3:");

foreach (int grade in lowerGrades)
{
       Console.WriteLine(grade);
}
```

```
/*
Este código produz o seguinte resultado:
As notas mais altas, excepto:
56
59
70
82
*/
```

SkipLastWhile<TSource>(IEnumerable<TSource>, Func<TSource, Boolean>)

Retorna todos os elementos da sequência à excepção dos elementos do fim da sequência para os quais a condição especificada é verdadeira.

```
int[] grades =
       { 59, 82, 70, 56, 92, 98, 85 };
var lowerGrades = grades
       .OrderBy(grade => grade)
       .SkipLastWhile(grade =>
       grade >= 80);

Console.WriteLine("Todas as notas menores que
80:");

foreach (int grade in lowerGrades)
{
       Console.WriteLine(grade);
}

/*
Este código produz o seguinte resultado:
Todas as notas menores que 80:
56
59
70
*/
```

SkipLastWhile<TSource>(IEnumerable<TSource>, Func<TSource, Int32, Boolean>)

Retorna todos os elementos da sequência à excepção dos elementos do fim da sequência para os quais a condição especificada é verdadeira.

```
int[] amounts =
{
       5000, 2500, 9000, 8000,
       6500, 4000, 1500, 5500
};

var query = amounts
       .SkipWhile((amount, index) => amount >
index * 1000);

foreach (int amount in query)
{
       Console.WriteLine(amount);
}

/*
Este código produz o seguinte resultado:
5000
2500
*/
```

IMPLEMENTADO OS OPERADORES TAKELAST

Implementando O Operador TakeLast

O operador TakeLast retorna o número especificado de elementos contínuos do fim de uma sequência e é implementado como o método de extensão TakeLast.

Para implementar este operador, primeiro começamos por memorizar, pelo menos, o número requerido (count) de itens da sequência

fonte (source) num array que funciona como uma memória intermédia (buffer) circular:

```
var sourceEnumerator =
source.GetEnumerator();
var buffer = new TSource[count];
var numOfItems = 0;
int idx;

for (idx = 0; (idx < count) &&
sourceEnumerator.MoveNext(); idx++,
numOfItems++)
{
buffer[idx] = sourceEnumerator.Current;
```

Se o número de itens memorizados (numOfItems) for inferior ao número requerido (count), produzimos os itens memorizados:

```
if (numOfItems < count)
{
       for (idx = 0; idx < numOfItems; idx++)
       {
              yield return buffer[idx];
       }
       yield break;
 }
```

Em seguida, iteramos pelos restantes itens e vamos os armazenando, de forma circular, na memória intermédia:

```
for  (idx  =  0;  sourceEnumerator.MoveNext();
idx = (idx + 1) % count)
{
                                buffer[idx]     =
sourceEnumerator.Current;
}
```

E finalmente, iteramos sobre os itens memorizados para produzir a sequência final:

```
for (; numOfItems > 0; idx = (idx + 1) %
count, numOfItems--)
{
       yield return buffer[idx];
}
```

Há duas optimizações que ainda podemos fazer.

A primeira é óbvia, se o número de itens requerido for 0 (zero), retornamos uma sequência vazia:

```
if (count <= 0)
{
  return
    System.Linq.Enumerable.Empty<TSource>();
}
```

A segunda é se a sequência fonte (source) implementar a interface IList<T>. Os objectos que implementam esta interface têm uma propriedade Count e acesso indexado aos seus itens que nos permite optimizar a produção da sequência final.

A produção da sequência final passa por produzir o número de itens requeridos do fim da lista (ou todos se a lista contiver menos que os itens requeridos):

```
int listCount = list.Count;
for (int idx = listCount - ((count <
listCount) ? count : listCount); idx <
listCount; idx++)
{
       yield return list[idx];
}
```

Implementando os Operadores TakeLastWhile

O operador TakeLastWhile retorna os últimos

elementos contíguos que satisfazem o critério especificado e é implementado como os métodos de extensão TakeLastWhile.
Para implementar este operador, primeiro começamos com uma memória intermédia (buffer) vazia e, para cada item que satisfaça o critério implementado pelo predicado (predicate), memorizamos esse item. Sempre que ocorrer um item que não satisfaça o critério de selecção, a memória intermédia é limpa:

```
var buffer = new List<TSource>();
foreach (var item in source)
{
    if (predicate(item))
    {
        buffer.Add(item);
    }
    else
    {
        buffer.Clear();
    }
}
```

Depois de percorrer toda a sequência fonte (source), produzimos todos os itens, se existirem, da memória intermédia:

```
foreach (var item in buffer)
{
    yield return item;
}
```

A diferença para o método que tem em conta o índice do item no predicado (predicate) de selecção é apenas na invocação deste:

```
var buffer = new List<TSource>();
var idx = 0;

foreach (var item in source)
{
    if (predicate(item, idx++))
    {
        buffer.Add(item);
    }
    else
    {
        buffer.Clear();
    }
}

foreach (var item in buffer)
{
    yield return item;
}
```

IMPLEMENTANDO OS OPERADORES SKIPLAST

Implementando o Operador SkipLast

O operador SkipLast retorna o número especificado de elementos contínuos do fim de uma sequência e é implementado como o método de extensão SkipLast.

Para implementar este operador, começamos por memorizar, pelo menos, o número requerido (count) de itens da sequência fonte (source) num array que funciona como uma memória intermédia (buffer) circular:

```
var sourceEnumerator =
source.GetEnumerator();
var buffer = new TSource[count];
int idx;

for (idx = 0; (idx < count) &&
sourceEnumerator.MoveNext(); idx++)
{
    buffer[idx] = sourceEnumerator.Current;
}
```

Em seguida, iteramos pelos restantes itens da sequência fonte (source) memorizando

circularmente cada item e produzindo o item armazenado anteriormente na mesma posição:

```
idx = 0;

while (sourceEnumerator.MoveNext())
{
    var item = buffer[idx];
    buffer[idx] = sourceEnumerator.Current;
    idx = (idx + 1) % count;

    yield return item;
}
```

Se o número de itens a excluir for igual ou superior ao número de itens existentes na sequência fonte (source), o método sourceEnumerator.MoveNext() retornará false na primeira iteração do ciclo while e será produzida uma sequência vazia.

Tal como acontecia com o operador TakeLast, podem-se fazer algumas optimizações.

A primeira é óbvia, se o número de itens requeridos for 0 (zero) ou inferior, retornamos uma sequência equivalente à fonte (source):

```
if (count <= 0)
{
    return source.Select(i => i);
}
```

A segunda é se a sequência fonte (source) implementar a interface IList<T>. Os objectos que implementam esta interface têm uma propriedade Count e acesso indexado aos seus itens que nos permite optimizar a produção da sequência final.

Se o número de itens a excluir for igual ou superior ao número de itens da sequência fonte (source), retornamos uma sequência vazia:

```
var list = source as IList<TSource>;
if (list != null)
{
    if (count >= list.Count)
    {
     return
     System.Linq.Enumerable.Empty<TSource>();
    }
    // ...
}
```

Se o número de itens da sequência fonte (source) for superior ao número de itens a excluir, produzir a sequência final consiste em produzir todos os itens da sequência fonte (source) excepto os últimos count itens:

```
int returnCount = list.Count - count;

for (int idx = 0; idx < returnCount; idx++)
{
    yield return list[idx];
}
```

Implementando os Operadores SkipLastWhile

O operador SkipLastWhile retorna os últimos elementos contíguos que satisfazem o critério especificado e é implementado como os métodos de extensão SkipLastWhile.

Para implementar este operador, primeiro começamos com uma memória intermédia (buffer) vazia e, para cada item que satisfaça o critério implementado pelo predicado (predicate), memorizamos esse item.

Sempre que ocorrer um item que não satisfaça o critério de selecção, a memória intermédia é limpa e o item que não satisfaz o critério é produzido:

LINQ: Implementação dos operadores TakeLast, TakeLastWhile, SkipLast e SkipLastWhile

```
var buffer = new List<TSource>();
foreach (var item in source)
{
    if (predicate(item))
    {
        buffer.Add(item);
    }
    else
    {
        if (buffer.Count > 0)
        {
         foreach (var bufferedItem in buffer)
            {
              yield return bufferedItem;
            }
            buffer.Clear();
        }
        yield return item;
    }
}
```

A diferença para o método que tem em conta o índice do item no predicado (predicate) de selecção é apenas na invocação deste:

```
var buffer = new List<TSource>();
var idx = 0;

foreach (var item in source)
{
    if (predicate(item, idx++))
    {
        buffer.Add(item);
    }
    else
    {
        if (buffer.Count > 0)
        {
         foreach (var bufferedItem in buffer)
            {
                yield return bufferedItem;
            }
            buffer.Clear();
        }
        yield return item;
    }
}
```

Conclusão

Como podem ver, implementar este tipo de operadores é muito fácil, pelo que, não se deve evitar a sua implementação sempre que se justifique. Implementar um específico para a nossa aplicação pode significar uma melhor legibilidade do código e até um aumento de performance.
Podem encontrar a implementação completa destes operadores (e outros) no meu projecto de utilitários e operadores para LINQ no CodePlex: PauloMorgado.Linq.

Recursos

O meu website: http://PauloMorgado.NET/
Livro "LINQ Com C#"
PauloMorgado.Linq
Classe Enumerable
Métodos de extensão

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.

BYACC

No artigo anterior adquirimos algumas noções sobre flex e vimos como implementar um scanner simples. Relembrando, o flex é um gerador de analisadores lexicais ou scanners, e o byacc é um gerador de parsers. Usados em conjunto permitem escrever aplicações bastante sofisticadas, mas no nosso artigo limitamo-nos a considerar uma calculadora simples, servindo como exemplo introdutório. Agora que temos o scanner para a nossa calculadora escrito podemos começar a considerar o parser.

Um ficheiro byacc tem a mesma estrutura que o flex, já que o analisador lexical em que o flex se baseia, o lex, copiou a sua estrutura do yacc, o gerador de parsers em que o byacc se baseia. Como tal o nosso ficheiro byacc vai ser algo deste género:

```
definições

%%

gramática

%%

int main()
{
        yyparse();
}
```

Comecemos por considerar como é que o parser e o scanner funcionam em conjunto. Um parser recebe tokens e verifica se esses tokens estão a seguir uma gramática. O que nós estávamos a retornar no scanner, o NUMBER e o NL são esses tokens. Os tokens podem ter um valor associado ou não. O que o parser faz é ver se eles seguem a gramática que ele define. Se o objectivo do parser for só verificar que um certo ficheiro, por exemplo um ficheiro xml, segue a especificação da linguagem, podemos não ter interesse nenhum em saber o valor de qualquer token. A nossa calculadora no entanto, precisa de produzir resultados, portanto necessita de saber o valor de cada token NUMBER que recebe. O parser define os tokens e o tipo de valores que lhes podem estar associados na zona das definições, sempre que precisa de um token novo ele chama o scanner com o resto do input que falta analisar. Vejamos a definição dos tokens para a nossa calculadora:

```
%union {
  double d;
};

%token<d> NUMBER
%token NL

%left '+' '-'
%left '*' '/'
%right '^'
%nonassoc UMINUS


%%

...

%%

int main()
{
        yyparse();
}
```

BYACC

A primeira coisa que encontramos é uma %union, seguida da declaração de dois tokens, NUMBER e NL. A %union serve para indicar tipos que queiramos associar a tokens e eventualmente a elementos da gramática. Ao escrevermos %token<d> NUMBER estamos a indicar que para além de NUMBER ser um token, tem um valor associado do tipo d, neste caso double. Isto vai ser importante quando passarmos para a gramática e é por esta razão que no scanner passávamos o número que tínhamos lido ao yylval.d antes de retornarmos o token. A informação da union e dos tokens que escrevermos nesta secção vai servir para o byacc gerar o header ao qual o nosso scanner fez #include e que permite ao scanner e ao parser comunicarem entre si. O '+', '-', '*', '/', '^' e UMINUS também são tokens mas aqui para além de indicarmos que são tokens estamos a indicar a associatividade dos operadores, um conceito que o leitor deverá conhecer, mas que abordaremos com maior profundidade na gramática.

```
%{
#include <iostream>
#include <cmath>
%}


%union {
  double d;
};

%token<d> NUMBER
%token NL

%left '+' '-'
%left '*' '/'
%right '^'
%nonassoc UMINUS

%type<d> expr
```

```
%%
expr_list    : expr NL              { std::cout
<< $1 << std::endl; }
             |   expr_list   expr   NL      {
std::cout << $2 << std::endl; }
             ;

expr  : NUMBER          { $$ = $1; }
      | '-' expr %prec UMINUS       { $$ =
-$2; }
      | expr '+' expr                { $$ =
$1+$3; }
      | expr '-' expr             { $$ = $1-
$3; }
      | expr '*' expr                { $$ =
$1*$3; }
      | expr '/' expr                { $$ =
$1/$3; }
      | expr '^' expr                { $$ =
pow($1, $3); }
      | '(' expr ')'              { $$ = $2;
}
      ;


%%

int main()
{
      yyparse();
}
```

Esta versão do parser introduz a gramática, mas comecemos por observar as diferenças na secção das definições. Podemos ver primeiro que a introdução de código C/C++ se processa da mesma forma que no scanner. Em seguida temos a instrução, %type<d> expr, expr não é um token, é outro símbolo da gramática mas ainda assim tem um tipo, portanto temos que o indicar.

Uma gramática, tal como a gramática da língua

portuguesa, serve para nos indicar a estrutura da linguagem, quais as composições de elementos que são válidas ou não de acordo com a linguagem. É fácil de ver como isto é útil para um compilador. A nossa calculadora tem uma gramática muito simples, mas ainda assim tem-la, portanto temos que compreender alguns princípios sobre gramáticas. As gramáticas são um conjunto de regras, às quais chamamos produções, constituídas por uma cabeça e um corpo. Por exemplo,

```
stmt : WHILE expression stmt;
```

é uma produção, com cabeça stmt e corpo WHILE expression stmt. A cabeça indica-nos uma construção da gramática e o corpo como ela é composta. Os elementos duma gramática dividem-se em símbolos terminais e não terminais, sendo os terminais, para efeitos deste artigo, os tokens que o scanner envia e os não-terminais as cabeças de produções, ou seja símbolos que podem ser produzidos através doutros símbolos.

Por clareza, usamos maiúsculas para representar terminais e minúsculas para não terminais, logo, no exemplo acima, stmt e expression são símbolos não terminais enquanto que o WHILE é terminal. Um símbolo terminal não pode ser obtido através da junção de outros símbolos, portanto só pode aparecer em corpos de produções. Um não terminal por outro lado pode ser derivado de terminais e outros não terminais, logo pode aparecer tanto na cabeça como no corpo duma produção. Um símbolo não terminal pode, e habitualmente, tem várias definições. Pegando no exemplo anterior e expandindo, obtemos,

```
stmt : WHILE expression stmt;
stmt : IF expression stmt;
stmt : IF expression stmt ELSE stmt;
stmt : block;
...
```

faz sentido criar várias produções para um stmt, já que um stmt pode ser várias coisas. Neste caso, um while, um if, um bloco, e continuando teríamos um for, um switch, um do while, etc. Sendo usual a existência de várias produções com a mesma cabeça, o byacc permite usar a seguinte sintaxe, com significado equivalente, para aumentar a legibilidade:

```
stmt  : WHILE expression stmt
      | IF expression stmt
      | IF expression stmt ELSE stmt
      | block
    ...
      ;
```

Se quiséssemos escrever um parser para uma linguagem de programação, tínhamos de ter uma forma de relacionar as várias regras, por outras palavras uma gramática necessita de uma estrutura. A forma de criar essa estrutura é criando um símbolo não terminal inicial ao qual todos os outros estão subordinados. Pode ajudar pensar na estrutura da gramática como uma árvore na qual o símbolo não terminal inicial é a raiz, os outros não terminais são nós e os terminais são as folhas.

```
%%

program : list
        ;

list    : statement
        | list statement
        ;

statement : declaration
          | function
          ;

declaration : type identifier
            |    type    identifier    '='
expression
            ;
```

```
type : INT
     | DOUBLE
     | CHAR
     ;

function : type identifier '(' arg_list ')'
block
          ;

block : '{' stmt_list '}'
       ;

stmt_list : stmt
           | stmt_list stmt
           ;
...
%%
```

Acima temos um esqueleto de uma linguagem muito simplificada, ainda com vários não terminais sem produções associadas, mas basta para termos uma noção da estrutura duma gramática. Qualquer que seja o programa escrito para esta linguagem, se ele estiver sintacticamente correcto deve ser possível partir de program e derivar todos os elementos que o compõem. Neste caso program é o símbolo inicial da gramática, e indicamos isso ao byacc colocando-o como a primeira produção do ficheiro.

Já vimos nos exemplos anteriores que as produções podem ser recursivas, o corpo duma produção pode conter o mesmo não terminal que é a cabeça. A recursividade é necessária e algo que o byacc processa muito eficientemente, mas usada sem cuidado é uma das coisas que introduz conflictos na gramática. Um conflicto surge quando o parser não sabe que regra usar para uma determinada sequência de tokens. Um exemplo segue-se:

```
s : x
  | y
  ;
x : A B C ;
y : A B C ;
```

Este caso é óbvio, temos duas produções diferentes com o mesmo corpo, o parser não sabe qual deve usar quando recebe os três tokens. O mesmo se passa nos casos seguintes.

```
s : x
  | y
  ;
x : A B C ;
y : A z C ;
z : B ;

s : x
  | y
  ;
x : A B C ;
y : z C ;
z : A B ;
```

O byacc só tem um token de lookahead, ou seja, no caso abaixo ele não consegue decidir se o que recebeu foi o x ou o y porque no momento de escolher a produção ele só sabe que o se segue é o token C. Como existem dois casos com o token C, ele não sabe o que fazer. Isto não é um problema com a gramática, é uma limitação do byacc.

```
s : x C D
  | y C F
  ;
x : A B;
y : A B;
```

O mesmo não ocorre no caso seguinte. Nesta situação ele consegue saber que produção usar vendo se o lookahead é D ou F.

```
s : x D
  | y F
  ;
x : A B;
y : A B;
```

A existência de conflictos na gramática não impede o parser de ser gerado, já que o byacc tem comportamentos default quando não sabe que produção usar, mas claro que o comportamento default pode não ser o que o programador tinha em mente quando escreveu a gramática portanto convém eliminar os conflictos antes de gerar o parser.

Os conflictos são introduzidos na escrita da gramática e podem sempre ser eliminados reescrevendo-a, mas por vezes é preferível eliminá-los usando precedências. Tomemos o seguinte exemplo clássico, o “dangling else”:

```
stmt : IF expression stmt
     | IF expression stmt ELSE stmt
     ;
```

Para o seguinte input:

```
if(a)
   if(b) c = a + b; else c = a;
```

O problema que nos surge é que o parser não sabe se o que ele tem é um IF expression stmt com expression igual a (a) e o stmt igual a if(b) c = a + b; else c = a; ou se em vez disso tem um IF expression stmt ELSE stmt com expression também igual a (a), o primeiro stmt igual a if(b) c = a + b; e o segundo stmt igual a c = a;. O que queremos que o parser faça? Associe o else ao primeiro if ou ao segundo? O comportamento que esperamos é que associe ao segundo, portanto para o conseguir vamos usar precedências.

```
%nonassoc IFX
%token ELSE

...

%%

...

stmt : IF expression stmt  %prec IFX
     | IF expression stmt ELSE stmt
     ;

...
%%
```

Introduzimos o token IFX - o token ELSE já tem de estar definido, tal como o IF se o estamos a usar na gramática - que não é retornado pelo scanner. O IFX só existe dentro do parser e serve apenas para indicar o nível de precedência de uma produção. Indicamos ao byacc o nível de precedência dos tokens pela ordem em que os declaramos, tendo os últimos maior precedência que os primeiros. A precedência de um token determina a precedência da produção em que ele se encontra, numa situação em que duas produções são viáveis como no exemplo anterior a produção com maior precedência é usada. Se quisermos modificar a precedência de uma produção sem afectar a precedência de nenhum dos seus tokens, podemos, como mencionado previamente, criar um token “falso” como o IFX e escrever %prec IFX no fim da produção para o conseguir. Usando o token IFX no primeiro IF estamos a dizer que aquele tipo de if tem menos precedência que o segundo IF,já que o token IFX é declarado antes do token ELSE, portanto quando existem as duas hipóteses o parser escolhe sempre a segunda, o que é o comportamento que desejávamos.

```
expr : expr '*' expr
     | NUMBER
     ;
```

Neste caso para o input NUMBER * NUMBER * NUMBER por exemplo, o parser não sabe se há-de fazer (NUMBER * NUMBER) * NUMBER ou NUMBER * (NUMBER * NUMBER). Para resolver este problema precisamos de indicar a associatividade do operador binário '*' ao byacc. Um operador com associatividade à esquerda para o input NUMBER * NUMBER * NUMBER * NUMBER faz ((NUMBER * NUMBER) * NUMBER) * NUMBER, um operador com associatividade à direita faz NUMBER * (NUMBER * (NUMBER * NUMBER)) para o mesmo input. A forma de indicar a associatividade de um token ao byacc é escrever %left ou %right em vez de %token quando o declaramos. Para declarar um operador unário não associativo (ou seja um operador que não se pode encadear) escrevemos %nonassoc em vez de %token.

Entendendo associatividade e precedências podemos começar a compreender a gramática da nossa calculadora.

```
%token NUMBER

%left '+' '-'
%left '*' '/'
%right '^'
%nonassoc UMINUS

%%

expr : NUMBER
     | '-' expr %prec UMINUS
     | expr '+' expr
     | expr '-' expr
     | expr '*' expr
     | expr '/' expr
     | expr '^' expr
     | '(' expr ')'
     ;
```

Dizemos que os operadores binários '+' e '-' são associativos à esquerda e têm a mesma precedência. Os operadores '*' e '/' funcionam de forma idêntica mas têm maior precedência. O operador de potência '^' é associativo à direita e tem a maior precedência de todos os operadores binários. O UMINUS indica que uma expressão negada tem a maior precedência e que não se pode negar uma negação. Os parênteses forçam a avaliação de qualquer que seja a expressão que estiver dentro deles, contornando as precedências caso seja necessário. Para o input, -1^2 + 5 + 3 - 2 * 3 ^ 4 ^ 5 / -6 / 2 - 5 a nossa gramática vai fazer (((((-1)^2) + 5) + 3) - (((2 * (3 ^ (4 ^ 5))) / (-6)) / 2)) - 5 como seria de esperar.

Podemos associar acções a cada produção, para serem executadas quando o parser acaba de processar o corpo de uma produção e o substitui pela cabeça. As acções indicam-se da mesma forma que no scanner, e é possível usar elementos da produção dentro delas desde que eles tenham valores associados. Para indicar o valor da cabeça usamos $$, para os membros do corpo usamos $1, $2, … $n de acordo com a ordem em que eles aparecem. Um não terminal tem por default associado um int. Se quisermos associar um tipo diferente declaramos %type<tipo-do-valor-associado> nome-do-símbolo-não-terminal na primeira secção do ficheiro. Vejamos então o ficheiro inteiro:

```
%%
%{
#include <iostream>
#include <cmath>
%}
```

```
%union {
  double d;
};

%token<d> NUMBER
%token NL
%left '+' '-'
%left '*' '/'
%right '^'
%nonassoc UMINUS
%type<d> expr

%%
expr_list   : expr NL            { std::cout << $1 << std::endl; }
            | expr_list expr NL  { std::cout << $2 << std::endl; }
            ;

expr : NUMBER                    { $$ = $1; }
     | '-' expr %prec UMINUS     { $$ = -$2; }
     | expr '+' expr             { $$ = $1+$3; }
     | expr '-' expr             { $$ = $1-$3; }
     | expr '*' expr             { $$ = $1*$3; }
     | expr '/' expr             { $$ = $1/$3; }
     | expr '^' expr             { $$ = pow($1, $3); }
     | '(' expr ')'              { $$ = $2; }
     ;
%%
int main()
{
       yyparse();
}
```

O ficheiro começa com os #includes de C++ que precisamos, seguidos da %union com o único tipo que a nossa calculadora necessita, um double. Lemos a declaração dos tokens, com os tipos associados, associatividade e precedência. Concluímos a primeira secção com a declaração que o símbolo não terminal expr tem um valor associado do tipo double. Isto é essencial porque as nossas operações não estão a receber tokens NUMBER como argumentos mas sim não terminais expr. O símbolo inicial da nossa gramática é expr_list e está definido como uma sequência de um ou mais não terminais expr, separados por mudanças de linha. Associamos a cada produção de expr_list a acção da impressão do valor de expr. As produções com cabeça expr tem associadas as acções que realizam os cálculos, guardando os valores na cabeça. Na terceira secção temos um main que chama o yyparse().

Neste momento temos um parser e um scanner para a nossa calculadora, no próximo artigo veremos como ligá-los e como modificar o scanner e o parser em conjunto de forma a adicionar mais funcionalidades à calculadora.

AUTOR

Escrito por **João Mares**
Estando actualmente no terceiro ano de LEIC no IST, interssa-se preferencialmente por arquitectura de computadores e computação gráfica.

Introdução ao Entity Framework

O ADO.NET Entity Framework permite criar aplicações em que o acesso a dados é feito com base em um modelo conceitual e não utilizando comandos directos à base de dados. Isto permite que o programador se abstraia totalmente da base de dados (criação de ligações de acesso, comandos, parâmetros, etc.) e utilize apenas objectos durante o desenvolvimento.

A versão 4.0 do Entity Framework (actual), que está disponível na .NET Framework 4.0, tem um conjunto de novas funcionalidades e melhorias, como é o caso de suporte a POCO - Plain Old CLR Objects (permite criar classes que não herdam, nem implementam nenhuma outra classes ou interface), abordagem Model-First (permite criar primeiro o modelo conceptual e, com base nele, criar a base de dados), suporte para o uso de funções em LINQ-to-Entities, Complex Types (criação de tipos de dados complexos), Deferred Loading ou Lazy Loading (capacidade de carregar as propriedades de associação das entidades no momento em são chamadas, se forem chamadas), etc. Não é especifica apenas para o SQL Server, pois existem *providers* que permitem usar outras bases de dados, como Oracle, MySql, PostgreSQL, DB2, SQL Anywhere, Ingres, Progress, Firebird, Synergy, etc.

Existem várias vantagens na utilização de um ORM (Object-relational mapping), que tornam a sua adopção quase inevitável, mas vejamos o seguinte exemplo:

*"SELECT * FROM utilizadores WHERE nome = 'Jorge' AND morada = 'Moita'"*

Este exemplo irá listar todos os utilizadores que tenham como nome "Jorge" e que tenham como morada a "Moita".

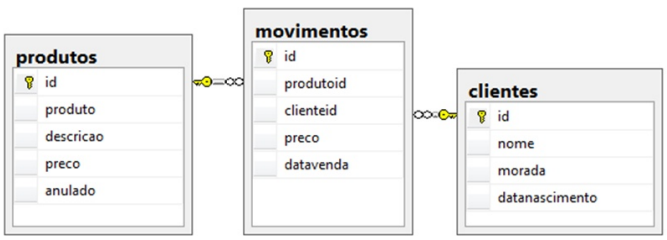
Imaginemos que o programador ou o DBA (database administrator) altera o campo "nome" para "nomecompleto", pois quer que se registe o nome completo do utilizador e, desta forma, o nome do campo fica mais coerente. Se aplicação for compilada não é detectado qualquer problema e só no momento de execução irá ocorrer um erro. Situações como estas são muito frequentes e podem representam vários erros na aplicação.

Este é apenas um exemplo de como se pode beneficiar, e muito, da utilização do Entity Framework, mas existem muito mais vantagens como a utilização de LINQ to Entities, intellisense no momento em que se está a escrever o código, código mais fácil de manter e actualizar, etc, etc.

Neste artigo, que será uma breve introdução ao Entity Framework, irá apenas mostrar como construir um modelo relacional de uma base de dados SQL, uma abordagem database-first, e como executar algumas operações CRUD.

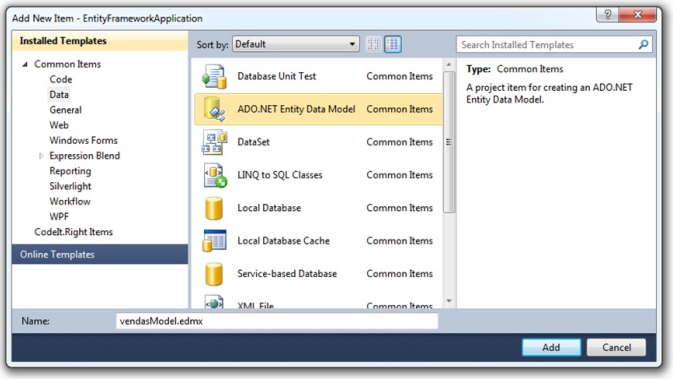
CRIAÇÃO DO MODELO RELACIONAL (database-first)

Para criação de um modelo relacional iremos usar a seguinte base de dados SQL:



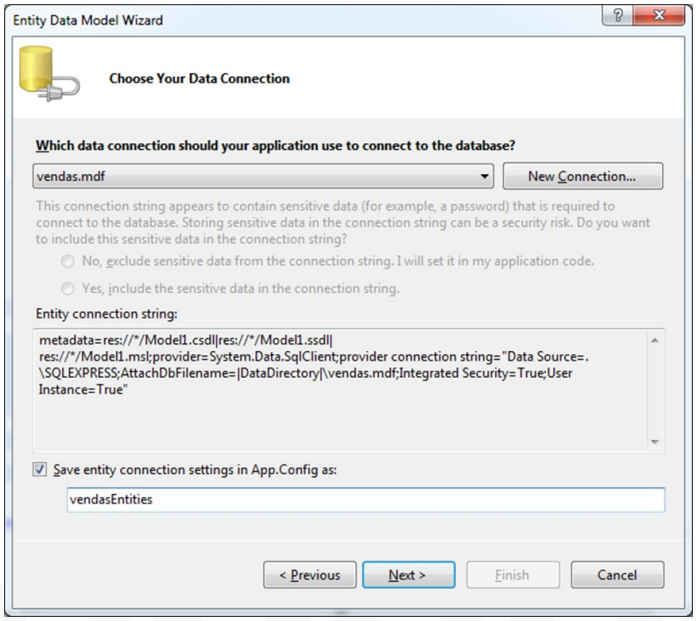
Como é possível ver nesta imagem, existem apenas 3 tabelas, com as respectivas chaves primárias/estrangeiras: produtos, clientes e uma terceira tabela onde se registam os movimentos.

O primeiro passo é adicionar um novo item ao projecto (menu Project – Add New Item), e seleccionar no grupo de templates "Data" o item "ADO.NET Entity Data Model". Isto irá iniciar um Wizard que permitirá criar o modelo final.



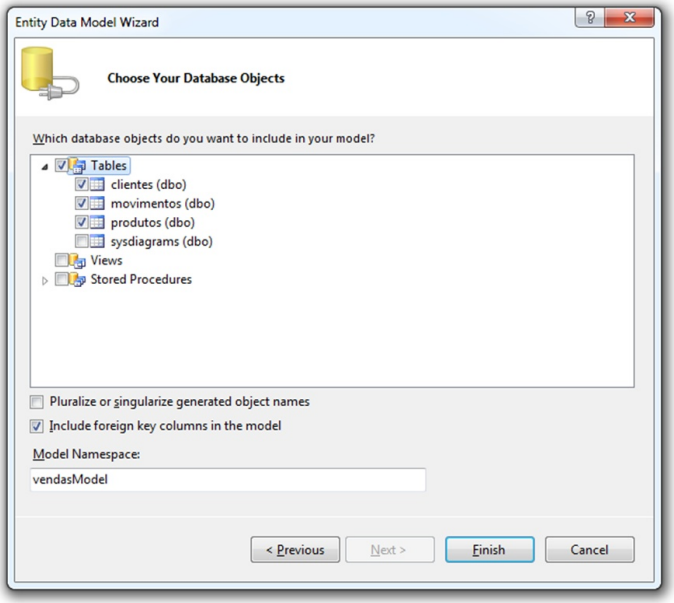
De seguida seleccionar "Generate from database" e carregar em "Next".

A próxima opção permite fazer a ligação à base de dados, escolhendo uma já existente ou criando uma nova ligação.



Para o nome da ligação, e para este exemplo, utilize "vendasEntities".

A última opção deste *Wizard* é onde são seleccionados os objectos a utilizar no modelo. Neste exemplo são escolhidas as três tabelas já referidas.



Após este processo está criado o ficheiro EDMX, que não é mais do que um conjunto de classes e métodos que nos permitirá aceder à base de dados de uma forma simples, como iremos ver de seguida.

OPERAÇÕES CRUD

As operações CRUD (acrónimo de Create, Read, Update e Delete em Inglês) são quatro operações básicas e fundamentais para a manipulação de bases de dados. São por isso

bastante importantes e requerem, sem a utilização de um ORM como o Entity Framework, algum trabalho que é normalmente complexo.

Com o Entity Framework este trabalho está bastante simplificado para o programador, necessitando apenas de algumas linhas de código, com recurso a IntelliSense, para as executar.

O funcionamento para todas as operações é semelhante: é criada uma nova instância da nossa entidade (neste exemplo "vendasEntities") que herda de uma classe da EntityFramework chamada ObjectContext, que a classe primária e responsável de gerir a informação como objectos e a ligação dos objectos aos dados.

Inserir registos

Para inserir novos registos, neste caso um novo cliente, criamos uma nova instância da respectiva classe, definindo as suas propriedades, que correspondem aos respectivos campos. Depois adicionamos o objecto através do método AddObject() e, finalmente, executamos o método SaveChanges() que irá efectuar as respectivas alterações, neste caso inserir um registo, na base de dados.

```
Using context As New vendasEntities

    ' Cria um novo cliente
    Dim cliente As New clientes With
        {
            .nome = "Rui Paulo",
            .morada = "Lisboa",
            .datanascimento =
             New DateTime(1980, 10, 31)
        }
    context.clientes.AddObject(cliente)
    context.SaveChanges()

    ' Podemos verificar logo o número
    '  (ID) do registo inserido
   Debug.WriteLine("Número de registo: " &
                cliente.id.ToString())

End Using
```

Apagar registos

Para apagar um registo é necessário seleccionar o registo correcto, neste caso usando uma Lambda Expression, e caso o resultado seja válido, executamos o método DeleteObject(), indicando o registo a apagar.

Finalmente gravamos as alterações na base de dados.

```
Using context As New vendasEntities

  ' Procura o registo com o número de
  ' cliente (ID) igual a 1
 Dim cliente = context.clientes.Where(
               Function(c) c.id = 1).
               FirstOrDefault()

    If cliente IsNot Nothing Then
       context.clientes.DeleteObject(cliente)
       context.SaveChanges()
    Else
        MessageBox.Show(
           "Número de cliente inválido")
    End If

End Using
```

Modificar Registos

Para modificar um registo, e à semelhança da operação anterior, é necessário seleccionar o registo. Neste exemplo utilizamos LINQ to Entities para fazer essa selecção. Depois, modificamos o que for para modificar e finalmente gravamos as alterações.

```
Using context As New vendasEntities

    Dim cliente =
        (From c In context.clientes
         Where c.id = 1
         Select c).FirstOrDefault()

    If cliente IsNot Nothing Then
        With cliente
            .nome = "Rui Paulo"
            .morada = "Setúbal"
            .datanascimento =
             New DateTime(1979, 10, 31)
        End With
        context.SaveChanges()
    Else
        MessageBox.Show(
        "Número de cliente inválido")
    End If

End Using
```

Listar Registos

A listagem de registo é também muito semelhante às operações anteriores. Neste exemplo, usando LINQ to Entities, vamos seleccionar todos os registo com base num critério, para uma variável do tipo IQueryable(Of T), e depois efectuamos um ciclo para mostrar os resultados obtidos.

```
Using context As New vendasEntities

    Dim listaClientes =
        From c In context.clientes
        Where c.morada.Contains("Lisboa") And
        c.datanascimento <
                  New DateTime(1985, 1, 1)
        Select c

    For Each c As clientes In listaClientes
        Debug.WriteLine(c.nome)
    Next

End Using
```

Entre entidades existir normalmente associações, e essas associações (também representadas na última imagem), têm algo que se designa por Navigation Properties. As Navigation Properties permitem uma navegação bidireccional entre entidades, de acordo com as suas associações. Isto é algo bastante prático pois permite-nos criar queries entre várias entidades usando LINQ to Entities.

O seguinte exemplo mostra esta abordagem, definindo depois o resultado como DataSource num controlo DataGridView.

NOTA: Se utilizar uma base de dados anexa ao

```
Using context As New vendasEntities

    Dim listaClientes =
        From c In context.clientes
        Join m In context.movimentos
            On m.clienteid Equals c.id
        Join p In context.produtos
            On m.produtoid Equals p.id
        Select m.datavenda,
```

```
        Select m.datavenda,
               c.nome,
               p.produto,
               m.preco

  Me.DataGridView1.DataSource = listaClientes

End Using
```

projecto (AttachDb), a base de dados é copiada para as pastas Debug/Release. Pode seleccionar no menu Project a opção Show All Files e verificar as alterações nessa localização.

CONCLUSÃO

Como foi possível ver ao longo do artigo, que abordou de uma forma muito superficial algumas das funcionalidades deste ORM, o Entity Framework é uma excelente opção para se trabalhar com bases de dados.

É simples de utilizar, permite-nos trabalhar apenas com objectos e com intellisense, resolve inúmeros problemas que são habituais quando trabalhamos directamente com a base de dados (formatos de datas, números, caracteres especiais), etc

Estas são apenas algumas, das muitas vantagens, que nos devem fazer de uma vez por todas, deixar de usar SqlCommands, SqlConnections, DataSets, etc.

Alguns recursos interessantes:

Data Developer Center
http://msdn.microsoft.com/en-us/data/ef.aspx

Data Access Hands-on Labs
http://msdn.microsoft.com/en-us/data/gg427655.aspx

ADO.NET team blog
http://blogs.msdn.com/b/adonet/

Julie Lerman's Blog
http://thedatafarm.com/blog/

AUTOR

Escrito por **Jorge Paulino**
Exerce funções de analista-programador numa multinacional sediada em Portugal. É formador e ministra cursos de formação em tecnologias Microsoft .NET e VBA. É Microsoft Most Valuable Professional (MVP), em Visual Basic, pela sua participação nas comunidades técnicas . É administrador da Comunidade Portugal-a-Programar e membro de várias comunidades (PontoNetPT, NetPonto, MSDN, Experts-Exchange, CodeProject, etc). É autor do blog http://vbtuga.blogspot.com - twitter @vbtuga

Programação Funcional com Perl (parte I)

Começo com este artigo uma série de artigos relacionados com a Programação Funcional em Perl.

Nesta série de artigos pretendo demonstrar algumas das capacidades do Perl para utilização com o paradigma de Programação Funcional.

Não pretendo ensinar nem as bases da Programação Funcional, nem do Perl. Vou abrir excepções quando se tratarem de aspectos mais avançados e/ou muito relacionados com a própria Programação Funcional. A compreensão elementar do Perl e da Programação Funcional é um requisito para a compreensão destes artigos contudo tudo será explicado de forma a permitir que um principiante possa compreender tudo e quem tiver dúvidas poderá contactar-me para as esclarecer.

Umas das principais características do Perl, em seguimento do mantra da sua comunidade («There's more than one way to do it»), é não obrigar o programador a utilizar, ou escolher um paradigma de programação, mas sim permitir ao programador utilizar o que quiser quando quiser, ficando ao cuidado dele ter os cuidados necessários para que o código seja útil e siga boas práticas de programação. Há quem concorde com esta filosofia, há quem não concorde, mas a filosofia do Perl e da sua comunidade não é o âmbito deste artigo.

Um dos muitos paradigmas de programação que são possíveis utilizar em Perl é a Programação Funcional (PF). Contudo não cabe a este artigo explicar a PF. Por isso sugiro que antes de prosseguirem na leitura deste artigo leiam pelo menos uma curta explicação do que é PF (os primeiros parágrafos do artigo da Wikipedia sobre PF por exemplo). Como já há vários artigos sobre PF em diversas linguagens, como Python, Scheme, Haskell (recomendo que os leiam), deixo aqui a sugestão a alguém, para redigir um artigo teórico e agnóstico (em relação a linguagens de programação) sobre Programação Funcional.

> “Umas das principais características do Perl (...) é não obrigar o programador a utilizar (...) mas sim permitir ao programador utilizar o que quiser quando quiser”

Algumas das funcionalidades elementares das linguagens funcionais são as funções anónimas e as closures. O Perl tem ambas as features.

Uma das alterações que o Perl 5.12 trouxe foi uma função chamada say. O say é basicamente um print que adiciona um newline à string que queremos imprimir (poupa-nos pelo menos 4 caracteres).

Qualquer uma das seguintes linhas de código vai imprimir isolado numa linha a string: «Hello world!»:

```
print "Hello world!\n";
print "Hello world!"."\n";
print "Hello world!", "\n";
say "Hello world!";
```

Para demonstrar a utilização de funções anónimas decidi criar uma implementação da função say que permite definir antes da sua utilização como queremos que a string seja formatada:

```
sub say {   #definição da função say
    my ($string, $format) = @_;
    my $str = $format->($string);
    print $str;
}
my $format = sub {    #função anónima de formatação da string
        my $str = shift;
        return $str."\n";
    };
say("Hello world!", $format);
```

Explicação do código do exemplo anterior:

Primeiro começamos por definir a função say, como definiríamos qualquer outra função em Perl. Depois definimos a função anónima de formatação, e atribuímos essa função a uma variável (que guarda uma referência para código) para que possa ser passada de forma visualmente mais limpa à função say (poderíamos também ter definido a função anónima, na invocação de say mas faria com que o código fosse muito menos legível).

Qual a utilidade das funções anónimas?

As utilidades são diversas. E vão desde a utilização das funções anónimas para poder modificar o comportamento de uma função que tenha sido criada para poder permitir essa alteração. Há quem utilize isto para poder por exemplo poder voltar a correr código que tinha sido serializado. E passam pela utilização em "dispach tables". Uma outra utilidade é a utilização nas "dispach tables":

```
my %lingua = (  #dispach table
    "pt" => sub { return "Olá mundo!" },
    "es" => sub { return "Hola mundo!" },
    "en" => sub { return "Hello world!" },
    "fr" => sub { return "Bonjour monde!" },
   );

sub dispach {
    my $l = shift;
        if(defined $lingua{$l} && exists $lingua{$l}) {
        my $str = $lingua{$l}->();
        print $str."\n";
    }
    else {
        print "Erro: lingua desconhecida!\n";
    }
}
dispach("fr");
```

Explicação do código do exemplo anterior:

Começou por ser definido uma hash table que, como chaves contém, as opções válidas de línguas que queremos utilizar na impressão de uma saudação e como valores, tem funções anónimas que imprimem uma saudação na língua representada pela sua chave na hash table.

Isto poderia ter sido feito definindo individualmente funções "regulares", criando variáveis que eram referências para cada uma dessas funções e utilizando essas variáveis como valores a utilizar na hash table.

Mas isso traria imediatamente dois problemas: aumentaria a quantidade de código necessário e este artigo tem um limite de caracteres; não

utilizaria funções anónimas, que é o objectivo deste exemplo ;).

Para além de isso, o que é pretendido neste exemplo, é algo tremendamente simples que pode perfeitamente ser feito utilizando funções anónimas, sem se aumentar a dificuldade relevante da sua compreensão como um todo e em parte.

A opção por esta solução, em vez de a utilização de um encadeamento maior de if-elsif, ou de um switch-case (também maior) permite que a solução escale para mais opções de línguas, sem qualquer alteração e atinge melhor vários dos objectivos da utilização de funções na programação: conter/isolar os problemas de forma a simplificar a sua resolução, facilitar a compreensão do código.

No próximo, artigo vou falar da segunda das principais capacidades do Perl para a Programação Funcional de que falei antes, as closures.

“(...) permite que a solução escale para mais opções de línguas, sem qualquer alteração e atinge melhor vários dos objectivos da utilização de funções na programação (...)”

AUTOR

Escrito por **Diogo Constantino**
é Consultor na área as tecnologias licenciadas como Software Livre, com aproximadamente vários anos de experiência profissional, especializado em desenvolvimento com Perl.
Larga experiência na área de sistemas, engenharia e de operações de redes de telecoms, tendo desenvolvido, mantido e utilizado ferramentas para diversas telecoms nestas áreas.
Conhecimentos em tecnologias como Perl, Python, PHP, MySQL, GNU/Linux, C, HTML, CSS, Javascript, GNU/Linux. Tendo tido ainda contacto profissional com tecnologias como Switchs, equipamentos de routing, firewalling e switching de diversos fabricantes, diversos sistemas *nix e diversas outras ferramentas e tecnologias comuns em sistemas *nix e/ou utilizadas em empresas de telecomunicações, redes e web. Relativamente conhecido advogado da causa do Software Livre em diversas comunidades.
Sócio da Associação Nacional para o Software Livre e da Associação Portuguesa de Programadores de Perl.

BackgroundWorkers - Implementação prática em Windows Presentation Foundation (WPF)

Neste artigo pretendo mostrar o que é o BackgroundWorker e vou exemplificar como se deve proceder à sua implementação usando a tecnologia WPF na versão .Net Framework 4.

Suponhamos:

"Tenho um algoritmo complexo de Optimização Combinatória que irá ter como parâmetro de entrada um objecto do tipo World. Classe que define toda a estrutura de dados da minha aplicação e no final retorna o objecto World com as alterações realizadas pelo algoritmo."

Esta geração vai implicar que tenhamos pelo menos três passos:
1. A partir do objecto World vamos criar a estrutura de dados do algoritmo;
2. Gera-se o algoritmo a partir dos dados recebidos;
3. Depois de gerar o algoritmo é preciso converter o resultado de forma a reflectir no World o resultado do algoritmo;

Enquanto estes três passos estão a decorrer, eu quero ser informada sobre o progresso da geração do algoritmo e pretendo ter a capacidade de a cancelar.

Este cenário é um dos muitos cenários onde o BackgroundWorker pode ser implementado. Outros exemplos reais são a leitura de ficheiros extensos, aceder a base de dados para efectuar o carregamento de dados, fazer o load de imagens e gerar relatórios.

Antes de passarmos à implementação vou dar uma breve apresentação teórica:

Um BackgroundWorker é uma classe contida no System.ComponentModel. Esta classe permite-nos ajudar a gerir uma tarefa numa thread separada da thread principal sem termos de nos preocupar com a inicialização e gestão da thread onde vamos executar a tarefa.

As propriedades a ter em conta são:

CancellationPending
Permite obter se a tarefa foi cancelada.

WorkerReportsProgress
Permite obter e definir se o BackgroundWorker vai reportar o seu progresso da tarefa.

WorkerSupportsCancellation
Permitir obter e definir se o BackgroundWorker vai permitir cancelar a tarefa.

Os métodos a ter em conta são:

RunWorkerAsync
Permite iniciar a tarefa.
Existem duas assinaturas deste método. Uma delas não tem argumento, a outra que o tem utiliza-o na execução da tarefa.

CancelAsync
Permite cancelar a tarefa e consequentemente o BackgroundWorker irá terminar.
Para que seja possível cancelar a tarefa é necessário que a propriedade WorkerSupportsCancellation tenha o valor de verdade true.

ReportProgress
Permite reportar o progresso da tarefa em dado momento.
Para que o progresso seja reportado é necessário que o WorkerReportsProgress tenha

o valor de verdade true.
Este método apresenta duas assinaturas, ReportProgress(int percentProgress) e ReportProgress(int percentProgress, object userState). Ambas permitem reportar a percentagem da tarefa que já foi realizada. Na última assinatura é possível enviar mais informação sobre o estado da tarefa.

Os eventos a ter em conta:

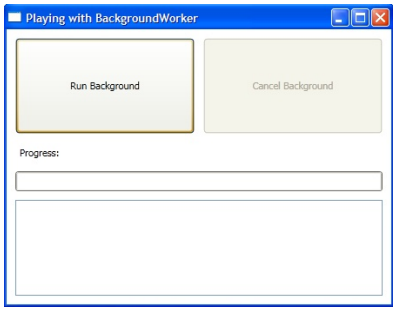
DoWork
Ocorre quando o método RunWorkerAsync é chamado.
Ao subscrever este evento é definido um handler que recebe dois parâmetros:
- Um parâmetro do tipo object que representa o objecto que subscreveu o evento.
- Um parâmetro do tipo DoWorkEventArgs que contém informação respeitante ao parâmetro de entrada e permite devolver o resultado da tarefa. De referir ainda que este parâmetro possui duas propriedades relevantes. Sendo elas, a propriedade Argument - que representa o objecto que é enviado como parâmetro do método RunWorkerAsync e a propriedade Result - que representa o resultado do DoWork.

ProgressChanged
Ocorre quando o método ReportProgress é chamado.
Um dos parâmetros do handler deste evento é um objecto do tipo ProgressChangedEventArgs, que contém duas propriedades relevantes, que são o ProgressPercentage e UserState, cujos valores foram enviados pelo método ReportProgress;

RunWorkerCompleted
Ocorre quando a tarefa é terminada, cancelada ou é lançada uma excepção.
Um dos parâmetros do handler deste evento é um objecto do tipo RunWorkerCompletedEventArgs, que contêm três propriedades relevantes.
São a propriedade Cancelled - que nos informa se a tarefa foi cancelada, a propriedade Error - que nos reporta a excepção que fez terminar a tarefa e a propriedade Result - que representa o resultado da execução da tarefa.

Vamos agora passar à implementação prática, para isso vou usar um projecto do tipo WPF Application.

Na interface da janela vamos ter:
- Um Button para executar a tarefa;
- Um Button para cancelar a tarefa;
- Um Progressbar para mostrar o progresso;
- Uma Listbox que indica mais detalhes do progresso.



Toda a interface é definida no ficheiro .xaml.
Em Code Behind iremos proceder à implementação de toda a funcionalidade para resolver o problema em causa.

1. Comecemos então por definir uma variável global:

```
private BackgroundWorker _backgroundWorker;
```

2. De seguida vamos criar um método onde vamos fazer a inicialização desta variável e definir as suas propriedades e subscrever os eventos:

```
private void InitializeBackgroundWorker()
{
   _backgroundWorker =
         new BackgroundWorker();

   _backgroundWorker.DoWork +=
         new DoWorkEventHandler(DoWork);

   _backgroundWorker.ProgressChanged +=
         new ProgressChangedEventHandler(
                  ProgressChanged);

   _backgroundWorker.RunWorkerCompleted +=
         new RunWorkerCompletedEventHandler(
                  RunWorkerCompleted);

  _backgroundWorker.WorkerReportsProgress =
true;

_backgroundWorker.WorkerSupportsCancellation
= true;
}
```

De notar que já estou a definir que o meu objecto _backgroundWorker vai permitir que a tarefa seja cancelada a meio e que o progresso da tarefa possa ser reportado.

3. Agora vamos definir os handler dos eventos:

```
private void DoWork
(object sender, DoWorkEventArgs e)
{}

private void ProgressChanged
(object sender, ProgressChangedEventArgs
args)
{}


private void RunWorkerCompleted
(object sender,
RunWorkerCompletedEventArgs e)
{}
```

4. No handler do evento do Click do btnRunBackground vamos mandar executar a tarefa e como parâmetro vamos enviar o objecto _world.

```
private void RunBackground_Click(
                        object sender,
                        RoutedEventArgs e)
{
   ClearProgress();

   _backgroundWorker.RunWorkerAsync(
                       (object)_world);
   btnCancel.IsEnabled = true;
   btnRunBackground.IsEnabled=false;
}
```

Caso não pretendêssemos enviar parâmetros usávamos a assinatura do RunWorkerAsync sem parâmetros.

5. No handler do evento do Click do btnCancel vamos mandar cancelar a tarefa, chamando o método CancelAsync.

```
private void btnCancel_Click(
       object sender, RoutedEventArgs e)
{
   _backgroundWorker.CancelAsync();
   btnCancel.IsEnabled = false;
   btnRunBackground.IsEnabled = true;
}
```

6. Neste momento estamos prontos para definir a tarefa (handler):

private void DoWork(object sender, DoWorkEventArgs e){}

É executado quando o método RunWorkerAsync é chamado. Consideremos a seguinte porção de código:

```
private void DoWork(object sender,
DoWorkEventArgs e)

{
   Algorithm algorithm = new Algorithm();
   World world = (World)e.Argument;

   object stepValue = world;

   foreach
   (IAlgorithmStep step in algorithm.Steps)
   {stepValue =  step.DoWork(stepValue);}
   e.Result = stepValue;
}
```

Note-se que vou obter o objecto do tipo World que foi enviado por parâmetro e em seguida vou iterar pelos vários passos do algoritmo.

Por fim vou devolver o resultado da tarefa.

Aparentemente posso pensar que já defini a minha tarefa, no entanto, ainda não estou a ter em conta que poderei cancelar a execução do algoritmo nem estou a reportar o progresso.

```
private void DoWork( object sender,
                          DoWorkEventArgs e)
{
   Algorithm algorithm = new Algorithm();
   World world = (World)e.Argument;

   object stepValue = world;

   foreach
   (IAlgorithmStep step in algorithm.Steps)
   {
      if
(_backgroundWorker.CancellationPending)
      {
         e.Cancel = true;
         return;
      }
      else
      {
         stepValue =
         step.DoWork(stepValue);
      }
   }
   e.Result = stepValue;
}
```

A cada passo do algoritmo vou verificar se a tarefa foi cancelada através da propriedade CancellationPending e consoante o valor de verdade desta propriedade a tarefa continua ou não.

Por fim temos que reportar o progresso de cada passo e para isso vamos recorrer ao método ReportProgress.

```
private void DoWork(
object sender, DoWorkEventArgs e)
{
   Algorithm algorithm = new Algorithm();

   World world = (World)e.Argument;

   _backgroundWorker.ReportProgress(
               5, "Is starting...");

   object stepValue = world;

   foreach
   (IAlgorithmStep step in algorithm.Steps)
   {
      if
(_backgroundWorker.CancellationPending)
      {
         _backgroundWorker.ReportProgress(10,
           string.Format(
           "{0} is canceled.", step.Name));
         e.Cancel = true;
         return;
      }
      else
      {
         _backgroundWorker.ReportProgress(10,
                 string.Format(
           "{0} is starting.", step.Name));

         stepValue =
         step.DoWork(stepValue);

         _backgroundWorker.ReportProgress(20,
                       string.Format(
           "{0} is finished.", step.Name));
      }
   }

   e.Result = stepValue;

   _backgroundWorker.ReportProgress(5,
                     "Finished.");

}
```

Neste momento já temos a nossa tarefa definida tendo em conta as operações que pretendíamos.

7. Para que o progresso seja apresentado na interface com o utilizador é preciso definir o seguinte handler:

private void ProgressChanged(object sender, ProgressChangedEventArgs args){}

Que é executado quando o método ReportProgress é chamado.

```
private void ProgressChanged(
              object sender,
              ProgressChangedEventArgs args)

{
   progressBar.Value +=
   args.ProgressPercentage;
   lbxProgress.Items.Add(
   args.UserState.ToString());
}
```

Desta forma actualizamos o progresso na interface com o utilizador.

8. Para terminar, vejamos o handler:

private void RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e){}

Que é executado depois do DoWork ter terminado, ter ocorrido uma excepção ou a tarefa ter sido cancelada.

```
private void RunWorkerCompleted(
             object sender,
             RunWorkerCompletedEventArgs e)
{
   if (!e.Cancelled)
   {
      try
      {
         _world = (World)e.Result;
      }
      catch
      {
         lbxProgress.Items.Add(
                        e.Error.Message);
         lbxProgress.Items.Add(
                       "Aborted.");
         progressBar.Value =
         progressBar.Maximum;
      }
   }
   else
   {
       lbxProgress.Items.Add("Finished.");
      progressBar.Value =
      progressBar.Maximum;
   }

   btnRunBackground.IsEnabled = true;
   btnCancel.IsEnabled = false;
}
```

Vou verificar se a tarefa foi cancelada ou não.

Em caso de sucesso posso obter o resultado da tarefa. Caso tenha ocorrido um erro, este será apresentado ao utilizador.

Em conclusão, a implementação de um BackgroundWorker permite-nos correr uma tarefa numa thread separada sem que esta interfira com a interface com o utilizador. Permite-nos ainda ter a capacidade de:
- Cancelar a tarefa;
- Reportar o progresso da tarefa;
- Actualizar os controlos de WPF enquanto a tarefa decorre.

A comunidade NetPonto é uma iniciativa independente e sem fins lucrativos, que tem como simples e único objectivo a partilha de conhecimento na área de arquitectura e desenvolvimento de software na plataforma .NET, na tentativa de disseminar o conhecimento diferenciado de cada um de seus membros.

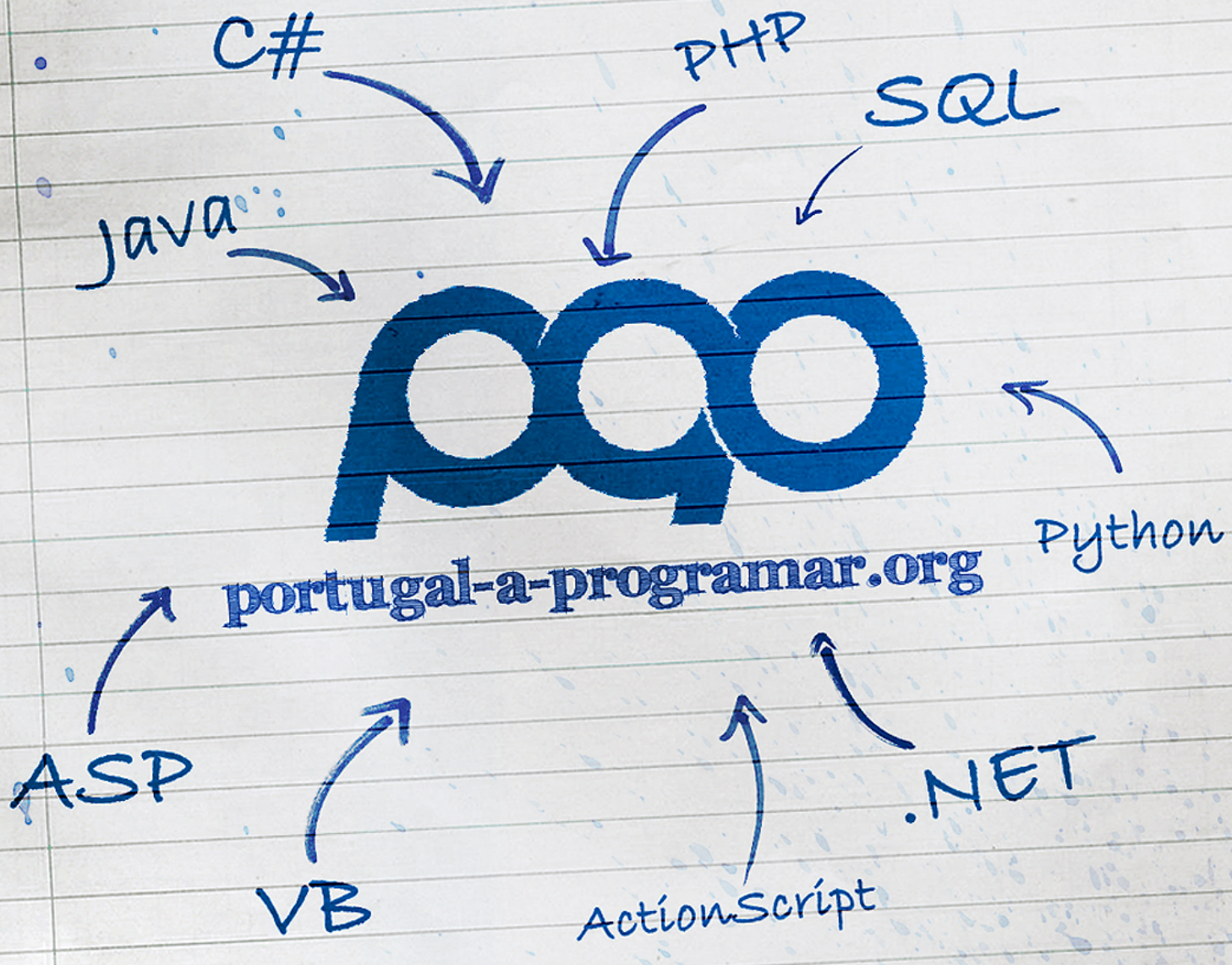
Cada um de nós tem diferentes talentos, e com as dezenas de tecnologias que são lançadas todos os dias, é muito difícil (para não dizer impossível) estar por dentro de tudo, e essa é uma das principais vantagens em fazer parte de uma comunidade de pessoas ligadas à nossa área. Podemos aprender mais, e mais rápido, com as experiências de cada um.

Visite-nos em http://netponto.org

AUTOR

Escrito por **Sara Silva**
É licenciada em Matemática – Especialidade em Computação, pela Universidade de Coimbra, actualmente é Software Developer no Porto.
O entusiasmo pela área resultou na obtenção dos títulos de Microsoft Certified Profissional Developer – Windows 3.5, Microsoft Certified Tecnology Specialist – WPF 3.5, WPF 4 e Windows Forms. Faz parte de várias comunidades, tendo uma participação activa na Comunidade NetPonto e no P@P.

A maior comunidade portuguesa de programação!



portugal-a-programar.org

Visite-nos!

jQuery - O quê, como e porquê?

O objectivo deste artigo é dar a conhecer as principais funcionalidades do jQuery, um pouco da sua história e motivações e fornecer exemplos da sua utilização.

O jQuery, existente desde 26 de Agosto de 2006, criado por John Resig, é uma cross-browser JavaScript library (embora considerado uma verdadeira framework), licenciado ao abrigo do MIT e GNU, fazendo dele software livre, e é o mais usado mundialmente face aos seus concorrentes (Dojo, MooTools, Prototype).

Os seus objectivos são simplificar a interacção com o DOM (Document Object Model), handling de eventos, criação de animações, interacção AJAX (Asynchronous JavaScript and XML), e, last but not least, ser cross-browser, que foi o factor decisivo para John Resig o criar.

O jQuery tem uma sub-library de nome jQuery UI, que é um subset do jQuery específico para animações e interface visual, fornecendo muitas funcionalidades out-of-the-box, como vários controlos, um motor de estilos visuais para estes (theme roller), várias funcionalidades utilizadas em interfaces ricos como drag & drop, dialogs, tabs, sliders, entre muitos outros.

O projecto jQuery gerou um spin-off de nome Sizzle, que é um motor de selectores CSS em JavaScript, sendo naturalmente utilizado no jQuery, a partir do qual foi gerado.

Actualmente a versão do jQuery é a 1.4.3 e do jQuery UI 1.8.5.
Para o futuro, a grande novidade é o jQuery Mobile (do qual já foi feita uma release 1.0 alpha 1 em 16 de Outubro de 2010), que irá revolucionar o desenvolvimento a nível de dispositivos móveis, pois o seu grande trunfo é ser cross-browser entre as várias plataformas móveis e além disso as próprias themes serem adaptadas ao aspecto do dispositivo em uso, fazendo com que o aspecto e usabilidade (look & feel) das aplicações desenvolvidas com este plugin não se distingam das aplicações nativas do dispositivo, o que é um avanço enorme no desenvolvimento de aplicações web para dispositivos móveis.

Pode-se acrescentar que tanto o jQuery como o jQuery UI têm um mecanismo de extensibilidade que permite aos programadores criarem plugins completamente integrados na API, fazendo com que actualmente existam cerca de 4000 plugins na página oficial de plugins do jQuery.
De seguida irei apresentar um tutorial muito básico de como desenvolver com jQuery que irei aprofundar em futuras edições.

Para iniciar o desenvolvimento, deve-se adicionar uma referência ao ficheiro js que contém o jQuery (minificado) na head do nosso html document: (img.1)

```
<html>
  <head>
          <script   type="text/javascript"
src="jquery-1.4.3.min.js"></script>
    <script type="text/javascript">

    </script>
  </head>
  <body>
<a            href="http://www.portugal-a-
programar.org/">PaP</a>
  </body>
</html>
```

img.1

Para código assim que o nosso documento estiver no estado ready (on document ready), antes do jQuery, o natural seria ter algum código:

```
window.onload = function() { alert("on doc ready"); }
```

O problema de executar código neste evento é que este só é executado quando o DOM está completamente carregado e todas as referências externas já foram descarregadas do servidor (imagens, js, css), o que não é nada agradável para o programador.

Por este motivo, o jQuery tem um evento que é despoletado logo que o DOM está carregado, não esperando pelos recursos externos:

```
$(document).ready(function() {
  //código aqui
});
```

Podemos adicionar código para ao clicarmos no link, mostrar um alert:

```
$(document).ready(function() {
      $("a").bind("click",
function(event) {
alert("Aceder ao site da comunidade
Portugal a Programar");
   });
});
```

Existem mais 2 modos de assignar um handler ao evento ready do nosso document:

```
$(document).bind("ready",
function() {});
```

e o modo mais abreviado e mais utilizado:

```
$(function() {
});
```

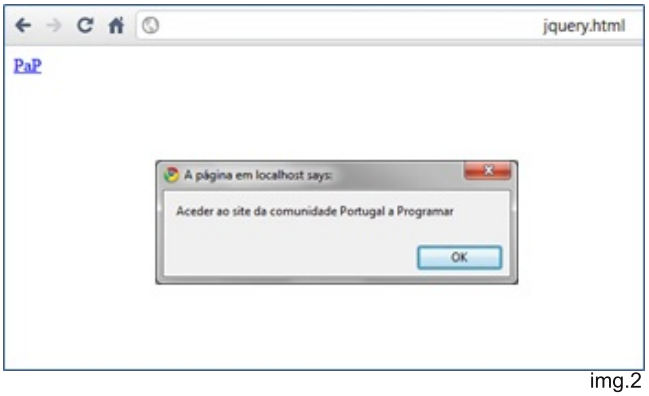
Neste momento podemos gravar o nosso ficheiro HTML com todo o código e testar:
Ao testarmos clicar no link, a popup aparece, e de seguida entramos na página Portugal a Programar.

```
<html>
  <head>
     <script type="text/javascript"
src="jquery-1.4.3.min.js">
     </script>
    <script type="text/javascript">
$(document).ready(function() {
$("a").bind("click",
function(event) {
</html>
      alert("Aceder ao site da
comunidade Portugal a Programar");
       });
```

img.2

```
        });
      </script>
    </head>
    <body>
<ahref="http://www.portugal-a-
programar.org/">PaP</a>
    </body>
</html>
```

Se quisermos prevenir o comportamento por omissão (default) do nosso clique sobre a âncora, podemos fazê-lo, chamando a função preventDefault do objecto event, que é recebido no handler de qualquer evento:
img. 4



img.2

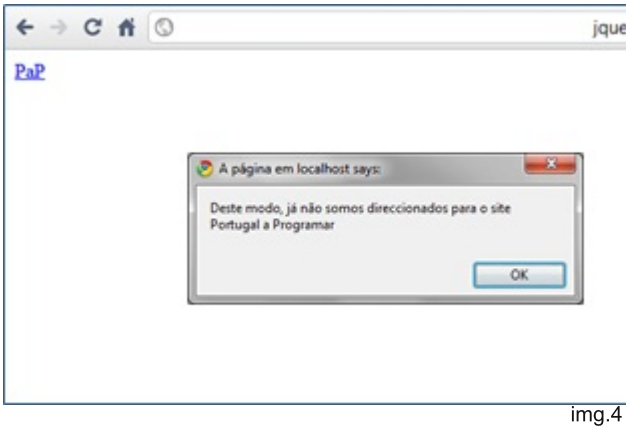
img.3

```
$(document).ready(function() {
$("a").bind("click",
function(event) {
alert("Deste modo, já não somos
direccionados para o site Portugal
a Programar");
     event.preventDefault();
   });
})
```

Para carregarmos o jQuery, a melhor opção é utilizar um CDN (Content Delivery Network, que fornece conteúdos na área geográfica do utilizador, de alta disponibilidade e a alta velocidade) da Microsoft ou da Google. Quanto ao script incluído inline, o ideal é ser colocado num ficheiro externo, de modo a não poluir o html.

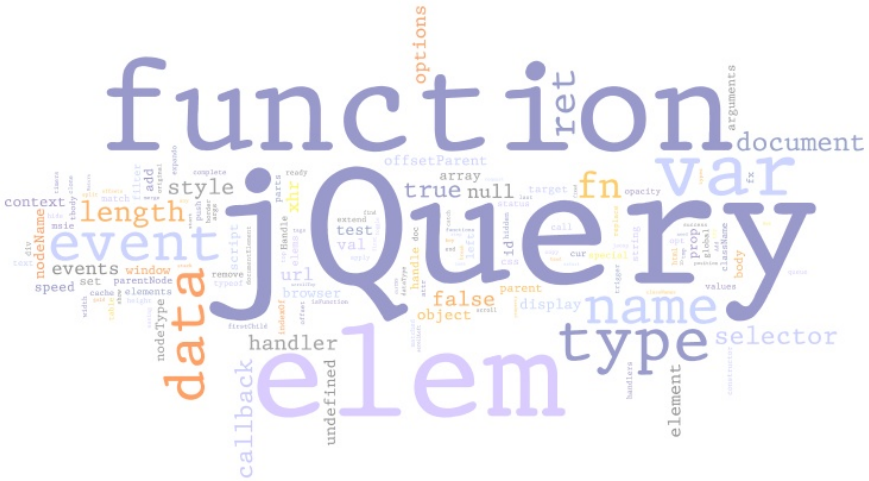
Como pude demonstrar neste pequeno tutorial básico, de iniciação ao jQuery, ele é uma framework tão acessível quanto poderosa e com um potencial cada vez maior e com cada vez mais funcionalidade, por isso, irei continuar em artigos subsequentes, a aprofundar cada vez mais as suas funcionalidades, truques e dicas, assim como partilhar boas prácticas de desenvolvimento, por isso não percam os próximos artigos!

> "Actualmente existem cerca de 4000 plugins na página oficial !"



img.4

> "Uma framework tão acessível quanto poderosa e com um potencial cada vez maior"



AUTOR

Escrito por **Ricardo Rodrigues**
É técnico Nível III em Informática/Gestão pela Fundação Escola Profissional de Setúbal, tendo ingressado após na FCT da Universidade Nova de Lisboa.
Posteriormente frequentou vários cursos da Microsoft em diversas áreas como Windows Forms,ASP.NET, Securing .NET Applications, WCF, WWF, Web Services e COM+ tendo obtido as certificações MCP .NET 2.0, MCAD .NET 1.1, MCSD .NET 1.1, MCPD Windows, Web e Distributed Applications e MCPD - Enterprise Applications Developer.
(MCP Profile)
Contribui activamente em comunidades como StackOverflow e também possui um blog/twitter como temática relacionada: Blog / @ricmrodrigues

Lua – Linguagem de Programação (Parte 6)

No artigo anterior (quinto artigo desta série) fora apresentado o tema relacionado ao uso e criação de funções. Como prometido este artigo tratará do tema relacionado ao uso de arquivos. Além deste será tratado também alguns aspectos ainda não mencionado sobre o ambiente interactivo da linguagem e a obtenção de tamanho de strings.

Um pouco mais sobre Lua

Na primeira parte desta série de artigos foi comentado um breve histórico sobre o surgimento da linguagem Lua. Cabe ainda acrescentar mais alguns detalhes.
Antes do surgimento da linguagem Lua o departamento de informática da PUC-Rio fazia uso interno das linguagens DEL (Data-Entry Language) e SOL (Simple Object Language)

Cada uma das linguagens fora desenvolvida entre 1992 e 1993 no sentido de adicionar maior flexibilidade de interactividade a dois projectos de aplicação em computação gráfica para o departamento de engenharia da empresa PETROBRAS. Apesar da ideia arrojada, ambas as linguagens apresentavam problemas muito comuns e foi a partir desses problemas que ocorreu a junção das linguagens DEL e SOL, dando vez a linguagem Lua.

Segundo reza a lenda o nome Lua foi sugerido devido ao uso da linguagem SOL. Mas cabe aqui um pouco de devaneio, pois me parece o nome Lua bem mais sugestivo que que uma ocorrência a partir da linguagem SOL. Lua é o satélite natural do planeta Terra.

A linguagem Lua é uma linguagem que apesar de sua independência operacional pode ser usada em conjunto com outras linguagens como se fosse um satélite natural para essas outras linguagens.

Fica aqui esta questão para ser pensada.

Ambiente Interactivo

A linguagem de programação Lua possui um ambiente de trabalho que pode ser usado na forma interactiva.
Neste modo de operação pode-se realizar algumas tarefas operacionais. Para tanto, na linha de comando prompt (modo terminal) do sistema operacional em uso faça a chamada do programa **lua** e accione <Enter>. Neste sentido será apresentada uma mensagem de identificação semelhante a:

```
Lua 5.1.4  Copyright (C) 1994-2008 … Rio
>
```

Para sair do prompt do ambiente Lua e retornar ao terminal do sistema operacional basta executar o comando os.exit() e em seguida accionar a tecla <Enter> ou <Return> ou ainda usar as teclas de atalho <Ctrl> + <c> ou então as teclas de atalho <Ctrl> + <z> + <Enter> dependendo do sistema operacional em uso.

Dentro do ambiente interactivo é possível passar comandos com o sentido de obter respostas. Por exemplo, desejando-se obter o resultado da soma do valor 7.5 com o valor 4.8, basta executar a linha de comando:

```
print(7.5 + 4.8)
```

Após accionar a tecla <Enter> ou accionar a tecla <Return> obter-se-á o resultado 12.3.

Veja a seguir uma série de operações com cálculos matemáticos que podem ser efectivadas junto ao ambiente interactivo.

```
print(-2 + -3)
print(2 * 3)
print(10 / 4)
print(10 % 4)
print(2 ^ 3)
print(2 + 5 * 3)
print(2 + (5 * 3))
print((2 + 5) * 3)
```

Os exemplos apresentados anteriormente resultam respectivamente nos valores: -5, 6, 2.5, 2, 8, 17, 17 e 21.

O ambiente interactivo retorna mensagens de erro quando alguma coisa não é escrita dentro das regras sintácticas do ambiente. Por exemplo, execute o código.

```
print(2 + + 3)
```

Ocorrerá a apresentação da mensagem de erro *stdin:1: unexpected symbol near* '+' informando que um dos símbolos "+" é desconhecido.

Para a obtenção de resultados de cálculos matemáticos pode-se substituir o comando print() pelo símbolo "=" igual a. Desta forma, podem ser definidos os seguintes cálculos.

```
=2*3
=2/3
=2+3
=2-3
```

Serão apresentados como resultados os valores 6, 0.66666666666667, 5 e -1.
Os cálculos efectivados poderá trabalhar com outros formatos numéricos, como notação cientifica. Por exemplo, desejando-se obter o resultado decimal do valor 9.5e2, sendo:

```
=9.5e2
```

Que mostrará como resultado o valor 950, pois calcula-se o valor 9.5 multiplicando-se por 10 e elevando-se a potência 2, ou seja, 9.5 x 102. Experimente =9.5e-2 para obtenção do resultado 0.095.

Outra possibilidade de operações são os usos de valores hexadecimais convertidos em valores decimais. Por exemplo, observe os seguintes comandos.

```
=0xa
=0xA
=0Xa
=0XA
```

Ao serem executados os comandos anteriores todos resultam na apresentação do valor 10 (A hexadecimal igual a 10 decimal). Note que não importa como é definido o valor, seja em formato de carácter maiúsculo ou formato de carácter minúsculo o resultado será sempre apresentado.

O ambiente interactivo permite a definição e uso de variáveis. Por exemplo, informe as linhas de código seguintes accionando a tecla <Enter> ou <Return> para cada linha informada.

```
A = 2
B = 3
print(A + B)
```

Observe o surgimento do valor 5 como resultado da operação.

Um cuidado que se deve tomar na definição de variáveis é com relação ao nome atribuído. Por exemplo as seguintes variáveis não são as mesmas variáveis, pois a linguagem Lua é do tipo case sensitive.

```
COR = 1
COr = 2
Cor = 3
cor = 4
Cor = 5
cOr = 6
CoR = 7
coR = 8
print(COR)
print(COr)
print(Cor)
print(cor)
print(Cor)
print(cOr)
print(CoR)
print(coR)
```

Outro detalhe no uso e definição de variáveis é a possibilidade de se efectivar a criação de múltiplas variáveis. Por exemplo, observe o código seguinte:

```
CODIGO, NOME = 123, "Paulo"
print(CODIGO)
print(NOME)
```

Observe que serão apresentados os valores 123 e Paulo. O símbolo "=" separa os conteúdos das variáveis (123 e Paulo) das variáveis (CODIGO e NOME).

Detecção de Tamanho de String

Vagarosamente a cada artigo estão sendo apresentados diversos recursos operacionais que podem ser utilizados junto a linguagem de programação Lua. Entre os recursos explanados cabe neste instante abordar o operador "#" de obtenção de tamanho de um string.
Observe a seguir as linhas de código que podem ser executados no ambiente interactivo.

```
print(#" ")
print(#"Linguagem Lua")
```

Após executar as linhas de código anteriores serão apresentados os valores 1 e 13 que são respectivamente o tamanho das mensagens um espaço e **Linguagem Lua**.

Arquivos

Um dos recursos mais úteis e importantes a ser oferecido por uma linguagem de programação de computadores á a capacidade desta linguagem permitir a manipulação de arquivos.
A linguagem Lua oferta para o programador uma biblioteca de entrada e de saída para o controlo das operações de arquivos. A abordagem deste artigo se restringirá em aplicações básicas.
O programa seguinte apresenta um exemplo de programa que cria um arquivo contendo valores numéricos pares situados na faixa de valores entre 1 um) e 200 (duzentos).

```
-- inicio do programa ARQ01

  ARQ = io.open("pares.txt","w")

  for N = 1, 200 do
    R = N % 2
    if (R == 0) then
      ARQ:write(N.."\n")
    end
  end
```

```
  ARQ:close()

-- fim do programa ARQ01
```

Em seguida escreva o código de programa em um editor de texto, gravando-o com o nome **arq01.lua** e execute-o com a linha de comando **lua 5.1 arq01.lua**.

O programa estabelece para a variável ARQ está sendo atribuída pela função **io.open()** que tem por finalidade abrir um arquivo para escrita, leitura ou adição. Caso o arquivo não exista o mesmo é automaticamente criado. A função **io.open()** utiliza entre aspas inglesas dois argumentos para sua operação:

- nome (primeiro argumento);
- modo (segundo argumento).

O argumento nome caracteriza-se por ser o nome do arquivo a ser aberto ou criado, neste caso pares.txt e o argumento modo indica a forma de operação sobre o arquivo, podendo ser:

- r – modo leitura (forma padrão);
- w – modo escrita;
- a – modo adição (inserção);
- r+ – modo actualização, todos os registos existentes são preservados;
- w+ – modo actualização, todos os registos existentes são apagados;
- a+ – modo actualização de adição, todos os registos existentes são preservados e as novas inserções ocorrem após o final do arquivo;
- b – modo de operação para manipulação de arquivos binários, usado ao lado direito da definição dos modo “r”, “w” e “a”.

A linha de instrução R = N % 2 obtém o valor do resto da divisão do valor da variável N por 2. Este valor poderá ser 1 (se o valor for ímpar) ou 0 (se o valor for par).

A instrução if (R == 0) then verifica se o valor do resto da divisão é zero, sendo executa a instrução **ARQ:write()** que escreve no arquivo o valor par obtido. O indicativo “\n” está sendo usado para armazenar no arquivo um valor numérico por linha. Sem este recurso os valores numéricos seriam armazenados um ao lado do outro.

Ao final a instrução **ARQ:close()** efectua o fechamento do arquivo.

Se quiser ver o conteúdo do arquivo basta pedir para listar seu conteúdo directamente na linha de comando do sistema.

O programa seguinte efectua a leitura do arquivo **pares.txt** e faz o somatório do valores e mostra seu resultado.

```
-- inicio do programa ARQ02

  ARQ = io.open("pares.txt","r")

  SOMA = 0
  for VALOR in ARQ:lines() do
    SOMA = SOMA + tonumber(VALOR)
  end

  ARQ:close()

  print(SOMA)

-- fim do programa ARQ02
```

Em seguida escreva o código de programa em um editor de texto, gravando-o com o nome **arq02.lua** e execute-o com a linha de comando **lua 5.1 arq02.lua**.

Lua – Linguagem de Programação (Parte 6)

O programa anterior faz uso do argumento de modo r (para leitura) quando da abertura do arquivo **pares.txt** por parte da instrução **io.open()**.

Quando da explanação do uso de laços a instrução for foi apresentada na sua forma mais simples. Note que neste exemplo a instrução for faz uso do comando in definida como for VALOR in **ARQ:lines()** do. Esta forma de uso da instrução for denomina-se uso genérico.

Neste caso, a instrução for faz com que a variável VALOR seja atribuída com o conteúdo (in) existente no arquivo **pares.txt** representado pela variável ARQ. Note o uso da função **lines()** anexa a variável ARQ que permite avançar no arquivo uma linha a frente posicionando-se sobre cada um dos registos.

A função **tonumber()** faz com que cada valor numérico armazenado no arquivo na forma de string quando capturado seja convertido para sua forma numérica. Apesar da linguagem Lua fazer este tipo de conversão automaticamente é fortemente recomendado que se defina este tipo de operação de forma explicita como feira no programa **arq02.lua**.

O próximo programa tem por objectivo fazer a entrada de dados com nome e telefone e armazenar os dados em uma agenda.

```
-- inicio do programa ARQ03

  ARQ = io.open("agenda.txt","a+")

  RESP = "S"
  while (RESP == "S") do
    io.write("Nome .......: ")
    NOME = string.upper(io.read())
    io.write("Telefone ...: ")
    TELE = io.read();
    REGISTO = NOME.." "..TELE
    ARQ:write(REGISTO.."\n")
    io.write("[+]registo? S/N ")
    RESP = string.upper(io.read())
  end

  ARQ:close()

-- fim do programa ARQ03
```

Em seguida escreva o código de programa em um editor de texto, gravando-o com o nome **arq03.lua** e execute-o com a linha de comando **lua 5.1 arq03.lua**.

O programa usa na função **io.open()** o modo de acção a+ para permitir a inserção de novos registos ao arquivo.

Um novidade neste programa é a função **string.upper()** que transforma em maiúsculo as entradas efectuadas junto as variáveis NOME e RESP.

O programa seguinte efectua a leitura dos dados do arquivo **agenda.txt**.

```
-- inicio do programa ARQ04

  ARQ = io.open("agenda.txt","r")

  for REGISTO in
    io.lines("agenda.txt") do
    io.write(REGISTO.."\n")
  end

  ARQ:close()

-- fim do programa ARQ04
```

Em seguida escreva o código de programa em um editor de texto, gravando-o com o nome **arq04.lua** e execute-o com a linha de comando **lua 5.1 arq04.lua**.

O programa usa a função **lines()** um pouco diferente do que fora utilizada no programa **arq02.lua**. Neste versão usa-se a função como **io.lines()** com o nome do arquivo **agenda.txt** como argumento.

Quando se utiliza arquivos na linguagem Lua pode-se fazer uso de duas abordagens de acesso aos seus descritores, que pode ser implícito ou explicito.

Quando se faz uso de descritores implícitos, todas as operações de acesso ocorrem com o uso dos recursos da tabela io. Quando se faz uso de descritores explícitos, todas as operações são providas como métodos do descritor de arquivo.

É possível com a linguagem Lua criar um programa que efectue a criação automática de páginas HTML/XHTML.

O programa a seguir cria uma página em código XHTML.

```
-- inicio do programa ARQ04

  ARQ = io.open("agenda.txt","r")

  for REGISTO in
    io.lines("agenda.txt") do
    io.write(REGISTO.."\n")
  end

  ARQ:close()

-- fim do programa ARQ04
```

Em seguida escreva o código de programa em um editor de texto, gravando-o com o nome **arq05.lua** e execute-o com a linha de comando **lua 5.1 arq05.lua**. Observe junto ao programa as definições das tags XHTML para criação da página com o nome teste.html.

Conclusão

Neste artigo foi dado atenção as acções de uso de arquivos em modo texto na linguagem Lua.
Foi também fornecido informações sobre o uso do ambiente em modo interactivo.
No próximo artigo será tratado o uso de funções para acesso em tabelas, escopos de variáveis, descrição de dados, meta-tabelas e orientação a objectos em Lua.

Errata

No primeiro artigo desta série foi informado equivocadamente que o resto de divisão entre dois valores poderia ser obtido a partir da função mod(). De fato, pode-se fazer uso desta função para se obter o valor do resto de uma divisão desde que esta função esteja associada a sua biblioteca, neste caso, a biblioteca math, por meio da sintaxe math.mod() ou então pode-se fazer uso do operador % mostrado neste artigo.

AUTOR

Escrito por **Augusto Manzano**
Natural da Cidade de São Paulo, tem experiência em ensino e desenvolvimento de programação de software desde 1986. 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.

E se amanhã a Apple dominar?

Já tenho idade e experiência suficiente para poder ter uma perspectiva sólida sobre a evolução de plataformas computacionais de desenvolvimento. Já percorri parte da "travessia do deserto" que a Apple efectuou no início dos anos noventa, altura em que mesmo os mais cépticos se questionavam sobre o futuro da Companhia. Se é verdade que a imprensa anunciou várias vezes a "morte" da marca de Cupertino (notícias que felizmente se veio a concluir serem manifestamente exageradas), não é menos verdade que embora uma plataforma nicho, esteve sempre muito à frente (e por vezes demasiado na vanguarda), da inovação e na capacidade de se recompor de cada desaire.

Sejamos francos, hoje a marca Apple é "trendy", "fashionable" e mais umas dúzias de palavrões do jargão IT. Arrasta multidões (e você pode muito bem ser um deles...) como uma lâmpada acesa atrai mosquitos. Nem sempre foi assim. Reduzidos a uma franja residual de utilizadores, autênticos gauleses irredutíveis na sua aldeia, a Apple sobreviveu graças à capacidade de visão dos que presidiram aos seus destinos (Steve Jobs, anyone?), mas também graças a uma capacidade inigualável de R&D (que também teve as suas nódoas negras no caminho), e a um desempenho industrial quase irrepreensível.

Se uma nuvem de "copycats" me irritou no passado, hoje isso quase que se tornou banal. Todos os trimestres se perfilam no mercado "Killers" de isto e daquilo. A maioria deles pretende ser "killer" de um produto Apple, mas não é menos verdade que há cemitérios inteiros cheios de produtos que se arrogam a ser a próxima grande descoberta. O verdadeiro "killer" está ainda, penso eu, nos estiradores dos conceitos...

Ipod, iPhone, iPad são apenas os três últimos exemplos do que acabei de tentar demonstrar.

> “a Apple fez de um mercado que não existia (...) um império colossal que rende milhões de dólares”

O caso do iPod é paradigmático na minha linha de pensamento: Não é a criação do hardware, não é a criação do software (que, recordo, foi Mac only durante um ano e alguns meses...) que fazem (ou fizeram) a revolução. É a experiência do utilizador. Pela primeira vez, um ecossistema permitia ao cliente não "power user", uma experiência de uso integrada, pacífica e totalmente tranquila. Esse é um dos grandes capitais da Apple. Colocar no mercado produtos pensados (verdadeiramente pensados) para o utilizador comum, que de uma forma tranquila se vai rendendo aos encantos desse esforço. Sim, a Apple fez de um mercado que não existia (Napster. Anyone?), um império colossal que rende milhões de dólares. Artistas, compositores e músicos querem hoje cortar o "middleman" e estar de forma directa na Store. Meia dúzia de anos passados, quem se lembra de como eram as coisas anteriormente?

Replicar a experiência noutras plataformas foi apenas uma questão de poucos anos. Não passa nenhuma semana no meu calendário que não tenha contactos de programadores

experientes ou simples iniciantes que me contactam em busca desse Santo Graal que é a iniciação ao desenvolvimento para iPhone. Sim, disse apenas iPhone; foi com este produto que tudo realmente começou. E ao longo dos últimos três anos, no início dos quais se contavam pelos dedos de ambas as mãos de um lançador de foguetes destreinado, a quantidade de gente portuguesa envolvida no desenvolvimento, passámos a contá-los por centenas e posteriormente já na casa do milhar.

> “Sim, disse apenas iPhone; foi com este produto que tudo começou”

É divertido. Divertido e reconfortante assistir a esta corrida ao ouro. Numa Store que todos invejam e tentam copiar (por vezes demasiado mal), há espaço para todos e todos são tratados por igual. É o mercado a funcionar. Destacam-se os bons dos medianos e maus, ninguém fica verdadeiramente prejudicado pois a escolha (compra) do cliente final é que é verdadeiramente decisória. Vingam as boas ideias. E essas, reconhecidamente temo-las em Portugal com programadores portugueses. Alguns deles já reconhecidos internacionalmente com prémios do próprio fabricante.

Quem sabe, se uma das suas ideias não chega ao top da AppStore?

AUTOR

Escrito por **Pedro Aniceto**
Pedro Aniceto, é Gestor de Produto Apple desde 1998 e utilizador da marca desde bem mais cedo. Não acredita em monstros, mas espreita de quando em vez para debaixo da secretária.

SQL em Oracle

Introdução

Quando desenvolvemos uma aplicação assente numa base de dados, um dos processos que normalmente nos passa ao lado é a optimização de queries.

Preocupamo-nos com a interacção da aplicação com o utilizador ou com questões de segurança, não dando importância a pormenores de SQL que podem, muitas vezes, condicionar a performance da aplicação. O tempo de espera na execução de uma query pode arruinar uma aplicação e, em última instância, uma base de dados inteira.

> "O tempo de espera na execução de uma query pode arruinar uma aplicação e, em última instância, uma base de dados inteira"

A solução passa, muitas vezes, por melhorar o hardware, culpando-o da baixa performance das nossas queries. Mas, nem sempre o problema reside no hardware. Aliás, muitas vezes, a lentidão de uma aplicação com acessos à base de dados resulta de SQL mal construído ou de estruturas de dados desadequadas ao nosso projecto.

Não existe uma fórmula para optimizar SQL. Cada caso é um caso. A estrutura de dados, a quantidade ou a qualidade dos dados são factores que influenciam o optimizador (o cérebro da base de dados). Alguns desses dados variam ao longo do tempo. Por isso, ou o optimizador é inteligente ao ponto de escolher sempre o melhor caminho, ou teremos de ser nós – programadores - a ensiná-lo.

> "a lentidão de uma aplicação com acessos à base de dados resulta de SQL mal construído ou de estruturas de dados desadequadas"

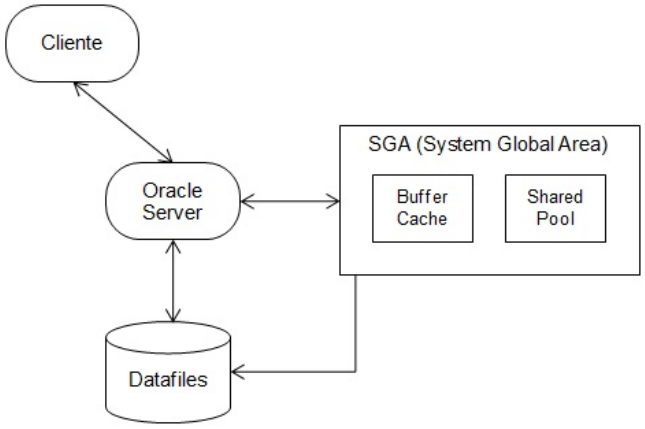
As bases de dados recentes, e em especial a Oracle, já têm algoritmos bastante poderosos que cobrem a maioria das situações. No entanto, há ainda alguma margem de "invenção" para o programador poder influenciar o desempenho do SQL.

Estrutura ultra simplificada de uma BD Oracle

Para sabermos o que optimizar e como optimizar, há que conhecer a estrutura interna de uma base de dados Oracle.

Como este artigo não pretende ser exaustivo, mostra-se apenas uma visão muito simplista do

modo de funcionamento de uma base de dados Oracle, o suficiente para perceber os conceitos aqui referidos:



Cliente – a "nossa" aplicação que comunica com a base de dados.

Oracle Server Process – processo que recebe as instruções do cliente e as executa.

Datafiles – ficheiros onde é armazenada a estrutura e os dados de uma BD.

SGA – zona de memória utilizada para guardar dados partilhados pelos vários utilizadores/aplicações da BD. Armazena dados no *Buffer* Cache e instruções SQL na *Shared Pool*.

As instruções SQL e os dados solicitados através de *queries* são guardados na SGA,

poupando tempos de processamento e acessos ao disco, em futuras chamadas da mesma *query*.

Como o Oracle processa um comando

Tomemos como exemplo o seguinte query:

```
select nome, morada, telefone from t_clientes
where nif = 12345;
```

1. O cliente envia a instrução SQL para o servidor.

2. O *server process* procura na *shared pool* uma instrução igual à solicitada. Se não encontrar, compila a nova instrução (*parsing*) e armazena-a na *shared pool*.

3. O *server process* procura na *buffer cache* os blocos de dados que necessita. Se não encontrar, faz uma consulta aos *data files*.

4. Os dados são enviados para o cliente.

Se repetirmos a mesma *query*, de seguida, o processo será o seguinte:

1. O cliente envia a instrução SQL para o servidor.

2. O *server process* procura na *shared pool* uma instrução igual à solicitada. Encontra-a e já sabe como encontrar os dados.

3. O *server process* procura na *buffer* cache os blocos de dados que necessita. Acabaram de ser requisitados na *query* anterior. Ainda lá estão. Não é necessário recorrer aos datafiles, pois os dados encontram-se todos em memória.

4. Os dados são enviados da SGA para o cliente.

Por onde começar?

Verificámos que o Oracle guarda as queries pedidas em memória com a informação do

melhor percurso para obter os dados. Imaginemos uma aplicação que executa, sistematicamente, esta query com diferentes condições:

```
select nome, morada, telefone from t_clientes
where nif = 12345;

select nome, morada, telefone from t_clientes
where nif = 12346;

select nome, morada, telefone from t_clientes
where nif = 23245;
(...)
```

As queries são todas diferentes. Logo, o Oracle vai armazenar cada uma delas na shared pool. Mas, na prática, o processo de obtenção dos dados é sempre o mesmo.

Na verdade, as queries são todas iguais, excepto o critério da condição where. O optimizador não é suficientemente esperto ao ponto de perceber que o processo é sempre o mesmo.

Poderíamos, facilmente, encontrar estas queries em memória, na SGA, executando a seguinte consulta:

```
select hash_value,sql_text, parse_calls,
loads, executions
from v$sqlarea
where sql_text like 'select nome, morada,
telefone from t_clientes where nif =%';
```

Verificaríamos inúmeras *queries* similares, com variância no parâmetro *nif*.

Uma funcionalidade do Oracle e que permite optimizar a utilização repetida de queries iguais são as **bind variables**. Estas variáveis permitem executar a instrução da seguinte forma:

```
                variable nif_n number;
                exec :nif_n := 12345;
query ->        select nome, morada,
telefone from t_clientes where nif =
:nif_n;
                exec :nif_n := 12346;
query ->        select nome, morada,
telefone from t_clientes where nif =
:nif_n;
```

Agora, sim, a *query* é sempre igual, aos olhos do optimizador Oracle e, em vez de guardar inúmeras *queries* iguais em memória, guarda apenas uma:

```
select nome, morada, telefone from
t_clientes where nif = :nif_n;
```

Estas situações tomam especial importância em bases de dados onde a mesma *query* é repetida milhares de vezes ficando a memória entupida com instruções iguais.

A repetição da mesma *query* milhares de vezes pode rapidamente ocupar toda a memória disponível. Ao utilizarmos *bind variables*, estamos a poupar CPU no *parsing* e RAM na *shared pool*.

Como caso prático da utilização de **bind variables** podemos referir um website de venda de automóveis usados.

Se o campo de pesquisa for a marca, modelo e ano, estaremos a executar uma *query* que procura numa tabela os automóveis com a marca X, modelo Y e ano Z, que podem ser seleccionados recorrendo a **bind variables**, evitando assim milhares de *queries* similares.

Conclusão

A utilização de **bind variables** é uma prática que deve ser adoptada sempre que possível. Se a nossa aplicação já está em produção, uma análise à vista dinâmica **v$sqlarea** permite detectar as instruções SQL similares que podem ser reduzidas a uma só.

> "A utilização de bind variables é uma prática que deve ser adoptada sempre que possível"

Para isso, podemos olhar para as colunas *parse_calls* e executions.

Uma *query* muito frequente está optimizada quando tem apenas uma *parse_calls* e muitas *executions*.

Naturalmente que, em queries pouco frequentes, não é necessário preocuparmo-nos com esta optimização e nem devemos generalizar a optimização somente às **bind variables**.

Existem muitas maneiras de tornar o código SQL mais eficiente e esta é apenas uma delas e de fácil implementação.

Uma análise mais aprofundada da estrutura de uma base de dados Oracle permite um conhecimento mais vasto sobre o modo como o Oracle processa os pedidos de informação.

Na prática, os conceitos aqui referidos são bem mais complexos. No entanto, a nível do programador que estrutura o SQL e que, eventualmente, cria a estrutura de dados, estes conceitos são suficientes para este tipo de optimização.

O detalhe vai interessar mais ao DBA que também poderá fazer "maravilhas" no *tunning* da base de dados.

AUTOR

Escrito por **Ricardo Trindade**
É actualmente o responsável pela área informática dos Serviços Sociais da Câmara Municipal de Lisboa onde efectua, desde administração de sistemas, até programação em .NET. Está também ligado ao projecto N-Ideias na área da consultoria, webdesign e software, trabalhando essencialmente com BD's Oracle, PostgreSQL e SQL Server.

Aspectos low-level de 3D

Introduçao ao 3D

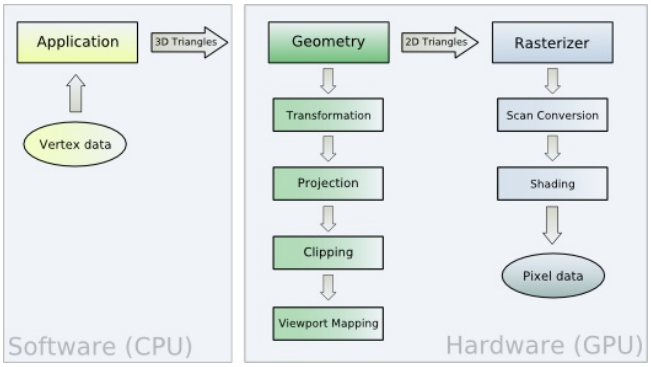
Este artigo explica alguma da teoria relacionada com o funcionamento de baixo nível de gráficos 3D, desde a placa gráfica, às bibliotecas que se usam para as programar, e alguma da magia (matemática) necessária para que tudo funcione. Recomenda-se alguma experiência básica com conceitos 3D.

Quem não se lembra de jogos como Quake e Duke Nukem 3D? Estes jogos (pelo menos as versões originais), eram "renderizados" (transformados numa imagem para mostrar no ecrã) completamente por software, ou seja, sem recurso a aceleração por hardware.

Só anos depois, com a popularidade dos jogos na plataforma PC, é que as placas gráficas começaram a ser vendidas e distribuídas com os computadores. As placas gráficas (GPU) são utilizadas para acelerar os cálculos necessários para representar realisticamente os objectos em 3D.

As placas são constituídas por processadores especializados, e com grande capacidade de paralelização, em contraste com um processador normal (CPU), que está preparado para executar os mais variados tipos de aplicações.

Com o aparecimento das primeiras placas gráficas surge a necessidade de criar APIs para tirar proveito das capacidades de processamento do hardware. As bibliotecas mais conhecidas são o OpenGL e o Direct3D, embora alguns talvez ainda se lembrem do Glide, que hoje em dia não passa de um artefacto histórico. Como funcionam e qual a relação entre a API (normalmente implementada nos drivers) e a placa gráfica?



Pensem na pipeline 3D como uma linha de execução de uma fábrica. Os materiais são introduzidos, processados em várias fases, e obtemos o produto final. Convém também ficar claro que existem vários métodos diferentes para fazer render de gráficos 3D. O que as placas gráficas actuais usam (e os primeiros jogos 3D que funcionavam apenas por software) é chamado de rasterisation. Existem outros métodos como ray tracing, mas não vão ser explicados.

As primeiras gerações de placas gráficas apenas executavam a fase Rasterizer, sendo que a fase de Geometry era totalmente processada pelo CPU. Devido à crescente complexidade das aplicações 3D, o CPU não conseguia processar toda a informação em tempo útil, e a fase Geometry passou a ser implementada no GPU. Esta capacidade é normalmente referida nas funcionalidades das placas gráficas por hardware T&L (Transform & Lighting).

Esta pipeline está de acordo com um modelo conhecido por Fixed-function pipeline. As placas gráficas modernas são muito mais flexíveis e dispõem de uma pipeline programável, utilizando shaders. Ainda é possível programar com este

modelo não-programável nas placas gráficas mais recentes, mas o seu funcionamento é "emulado" pela API gráfica.

A esmagadora maioria dos chipsets gráficos produzidos hoje em dia são programáveis, sendo a excepção mais notável a consola Wii da Nintendo. Os dispositivos móveis também demoraram um pouco a adoptar este novo modelo, mas as novas versões do iPhone e dispositivos Android também já suportam shaders (através do OpenGL ES 2). A web também está a caminhar para este modelo, com a adopção do WebGL por vários browsers.

Application stage

Tal como o nome implica, a primeira fase ocorre na aplicação desenvolvida pelo programador, e é processada apenas no CPU. Nesta etapa, são normalmente processadas tarefas como detecção de colisão (Collision detection), animação, simulação de física, algoritmos de aceleração (Space partitioning, Culling), entre outros, dependendo do tipo de aplicação.

A aplicação é também responsável por enviar os dados dos objectos 3D para a API. Os objectos em 3D são geralmente representados por polígonos: triângulos e quadriláteros. Existem outras formas de representação, como curvas/superfícies NURBS (Non-uniform Rational B-splines), mas apesar de permitirem a modelação de alguns tipos de objectos de forma mais precisa (através de modelos matemáticos) precisam de ser convertidas noutras representações.

Por questões de optimização todos os cálculos internos são feitos em triângulos, por ser uma representação simples. Consequentemente, todas as outras formas de representação são convertidas para triângulos, usando um processo chamado tessellation (ou Polygon triangulation).

Realizada esta conversão os dados são enviados pelo bus (AGP ou PCI-Express) para a placa gráfica.

Geometry stage

Esta fase recebe os vértices 3D enviados para a placa gráfica pela fase anterior, e executa uma série de transformações necessárias para adaptar estes dados num formato adequado para o processamento da próxima fase. Esta é a parte mais importante da pipeline, e onde são efectuados a maior parte dos cálculos.

Antes de analisarmos em mais detalhe cada uma das transformações desta etapa, vamos ver o que são espaços de coordenadas e como são representados os vértices.

Em 3D é comum utilizarmos vários espaços de coordenadas pois faz sentido falar de diferentes coordenadas dependendo do objecto ou referência que estejamos a considerar. Os espaços fundamentais são denominados por Model/Object Space, World Space e Camera/Eye Space, pela ordem que são processados na pipeline.

Inicialmente um objecto em 3D reside num espaço chamado Model Space, que significa que

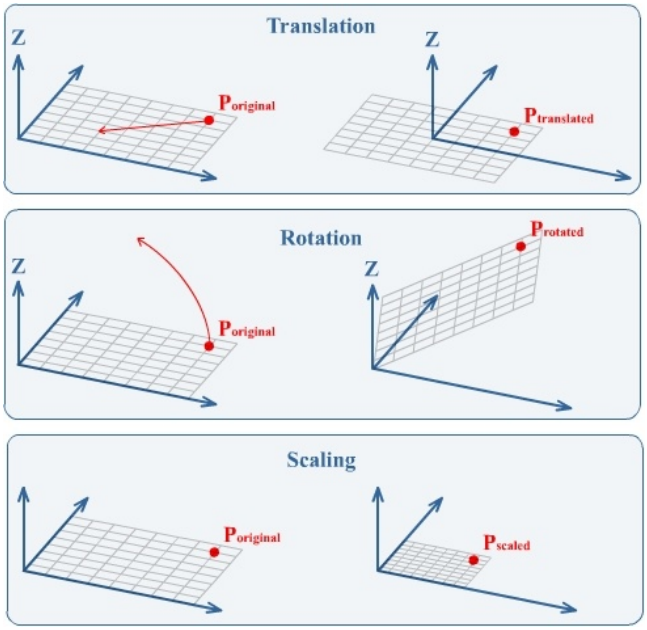
o objecto ainda não foi transformado e se encontra no seu próprio espaço de coordenadas. É importante que cada objecto defina o seu próprio espaço, pois podemos associar várias transformações a um objecto inicial, obtendo várias cópias no objecto no mundo com localizações, orientações e tamanhos diferentes, sem ser necessário enviar todos estes objectos diferentes para a placa gráfica (esta técnica é conhecida por Instancing). Neste caso os objectos estão normalmente posicionados na origem para fácil manipulação.

World Space é um espaço de coordenadas "especial" pois todos os outros espaços vão ser referenciados em relação a este. Em outras palavras este é o "maior" espaço de coordenadas no mundo.

É mais fácil perceber a relação entre estes diferentes espaços utilizando um exemplo. Todos nós temos o nosso próprio espaço de coordenadas. Se eu vos disser para darem um passo em frente (Object Space), não faço ideia se vão para Norte, Sul, Este ou Oeste (World Space).

Antes de estudarmos os diferentes tipos de transformações que os objectos sofrem, é necessário um modelo que possa ser facilmente trabalhado. Os pioneiros em computação gráfica escolheram vectores como a forma de representação dos vértices. Esta representação permite reutilizar todo o conhecimento da área matemática conhecida por álgebra linear. Esta área estuda as propriedades dos vectores e dos conceitos associados (espaços vectoriais).

Esta representação vai permitir o uso de matrizes (que podem ser visualizadas como um conjunto de vectores) para a manipulação dos vértices.



As matrizes vão corresponder a transformações de um espaço de coordenadas para outro diferente. Exemplos de transformações representáveis com matrizes são translações, rotações, escalamentos ou redimensionamentos, reflexões, projecções, entre outros.

Existem várias formas de representar matrizes na memória: row-major order e column-major order. É fundamental que não se misturem os dois tipos de representações, pois a ordem da multiplicação para a concatenação de transformações é diferente.

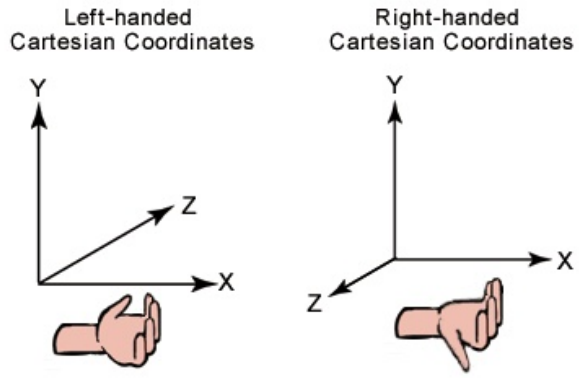
Com o uso de matrizes também se torna possível a concatenação de diferentes tipos de transformações (representadas em diferentes matrizes) numa única matriz. Isto é bastante útil, pois apenas temos de multiplicar os vértices por essa matriz composta e obtemos logo o resultado equivalente a multiplicar o vértice por cada transformação em separado.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{pmatrix}$$

Existem bastantes mais detalhes sobre matrizes que convém estudar para se conseguir perceber detalhadamente como a teoria (e a prática!) funciona. Mais concretamente, é fortemente recomendada a leitura sobre coordenadas homogéneas, transformações lineares e transformações afins.

É bom que estudem também como as diferentes transformações funcionam em termos de operações matriciais. Como exemplo, vejam uma matriz de translação (apresentada na figura) e tentem multiplicar um vector pela matriz.

As diferentes APIs gráficas usam diferentes convenções para a representação de matrizes e vectores em memória. Vamos ver as diferenças:



Convém também perceber as convenções de eixos usadas nas principais APIs gráficas podem ser vistas de diferentes formas. É preciso ter cuidado se temos um sistema de coordenadas left-handed ou um sistema de coordenadas right-handed.

Em relação à correspondência dos eixos, a convenção usada é que o Y corresponde a cima/baixo, o X a esquerda/direita e o Z a frente/trás (ver imagens), embora isto possa ser alterado.

Agora que já vimos o modelo usado para trabalhar os vértices, e como eles são representados no computador, vamos analisar as transformações principais que acontecem internamente na pipeline (não se preocupem se não perceberem os diferentes espaços de coordenadas, vão ser explicados mais à frente):

1. Model transform, transformação de Model Space para World Space.
2. View transform, transformação de World Space para Camera Space.
3. Projection transform, transformação de Camera Space para Clip Space.

Usando notação matemática, podemos representar estas transformações pela seguinte multiplicação de matrizes:

$$v_{clip} = (v_{model})(M_{model \to world})(M_{world \to camera})(M_{camera \to clip})$$

Model & View Transform

A primeira transformação que ocorre é a transformação designada por Model. Esta transformação é usada para colocar os objectos na sua posição do mundo. Como já vimos, os objectos começam com o seu próprio espaço de coordenadas, e normalmente estão centrados na origem. Se quisermos colocar dois cubos no mundo 3D (definidos num ficheiro externo e centrados na origem), vamos ter duas matrizes de model diferentes, que os colocam em posições diferentes, e que também permitem que eles tenham tamanhos diferentes ou que não estejam orientados da mesma forma.

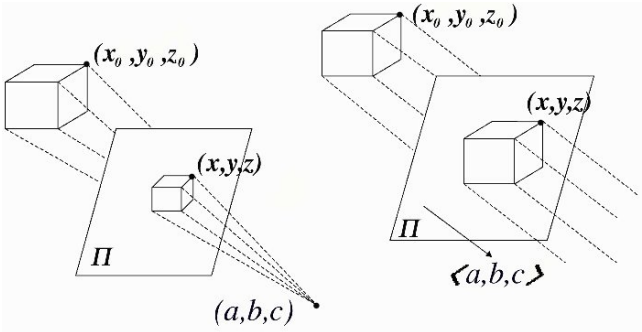
A transformação View vai colocar os objectos centrados no espaço da câmara. Ou seja, objectos mesmo à frente da lente da câmara vão estar na origem desse novo espaço. Ao início pode parecer confuso, pois ao contrário do que estamos habituado no mundo real (onde os objectos não mudam de sítio), em 3D são os vértices dos objectos que são multiplicados por

uma matriz, e é normal que sejam os objectos que mudam de sitio para serem vistos de diferentes posições.

Projection

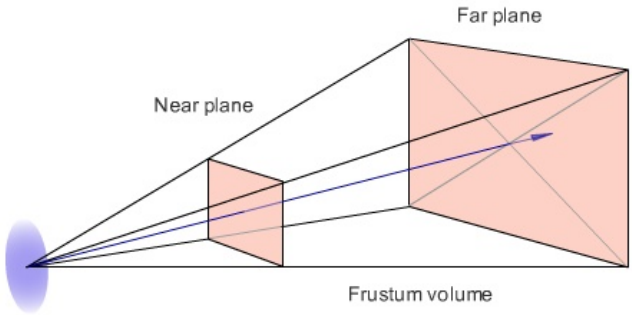
Depois de os objectos estarem posicionados no espaço da câmara, é necessário que eles sejam projectados. A projecção consiste no mapeamento dos vértices (em 3D) para um certo plano com representação bi-dimensional (2D). Podemos considerar dois tipos de projecções:

- Ortogonais (ortographic projection)
- Perspectivas (perspective projection).



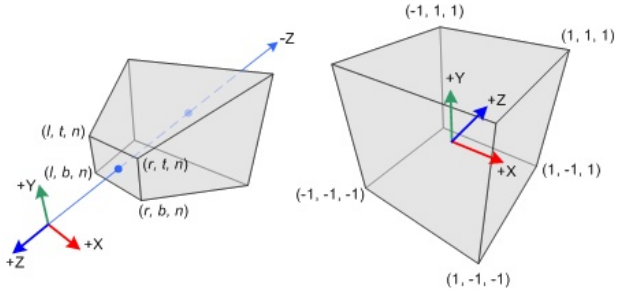
Tal como representado na figura, podemos ver uma projecção perspectiva na esquerda. As linhas, quando vistas no horizonte, focam-se num ponto (vanishing point). A este efeito chama-se foreshortening. Tem a característica que objectos mais longe parecem mais pequenos. Este é o tipo de projecção a que estamos habituados a ver no dia-a-dia.

Outro tipo de projecção, apresentado na direita, é a projecção ortográfica. Esta projecção tem a característica que linhas paralelas continuam paralelas depois da projecção. Por exemplo, estamos habituados a ver este tipo de projecção em plantas de edifícios.



No final da projecção, o resultado de multiplicar todos os vértices pelas matrizes de transformação e projecção vai ser diferente, dependendo do tipo de projecção escolhido. No caso da projecção ortográfica, o resultado vai ser um paralelepípedo, enquanto no caso da projecção perspectiva vai ser uma pirâmide invertida com o topo truncado, a que se dá o nome de frustum (exemplo na figura).

No final da projecção, o resultado de multiplicar todos os vértices pelas matrizes de transformação e projecção vai ser diferente, dependendo do tipo de projecção escolhido. No caso da projecção ortográfica, o resultado vai ser um paralelepípedo, enquanto no caso da projecção perspectiva vai ser uma pirâmide invertida com o topo truncado, a que se dá o nome de frustum (exemplo na figura).



As APIs gráficas normalizam o resultado de todos os tipos de projecção num cubo centrado na origem com dimensões que variam de (-1,1) no eixo X, Y e Z. Isto permite que os restantes

passos da pipeline sejam executados de forma uniforme. O espaço resultante é chamado de clip space pois nesta mesma fase todos os vértices que estejam fora do espaço definido pelo cubo, são ignorados (também conhecido por clipping).

É de notar que o funcionamento apresentado pode ser ligeiramente diferente entre o OpenGL e o Direct3D, mas a teoria é equivalente entre os dois. Normalmente só alguns pormenores variam, sendo recomendada a consulta da documentação oficial de cada API para saber mais concretamente de como são executados todos estes passos. Alguns pequenos pormenores também foram omitidos pois requerem conhecimento mais avançado sobre a representação de coordenadas homogéneas em matrizes (perspective divide e normalized device coordinates).

Viewport Mapping

As APIs gráficas normalmente permitem a definição de uma sub-região da janela onde o output final é mostrado. Imaginem por exemplo um jogo de carros, com suporte para 2 jogadores no mesmo ecrã (o que os jogos normalmente chamam de split-screen). Normalmente o ecrã é dividido ao meio e é mostrada a área de jogo de cada jogador em cada uma dessas regiões. Nas APIs gráficas, cada uma dessas regiões é conhecida por viewport.

Depois de obter as coordenadas normalizadas no cubo, é feito o mapeamento para o espaço de coordenadas da janela, ou Window Space. Este processo é conhecido por Viewport Mapping, pois tem em conta as definições do viewport actual, para permitir que diferentes imagens sejam mostradas na mesma janela.

Rasterizer stage

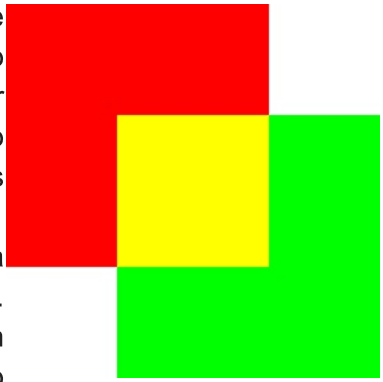
Esta fase utiliza as primitivas geradas na fase anterior para determinar os píxeis finais que vão ser escritos no framebuffer. O primeiro passo a ser processado é conhecido por Scan conversion. Este passo transforma os triângulos 2D (mais a sua profundidade) em fragmentos. Fragmentos são potenciais píxeis que vão ser escritos no framebuffer. Têm associados informação como localização, cor, coordenadas de texturação e profundidade.

A informação que está associada aos fragmentos é usada para determinar a sua cor. A cor de um fragmento pode depender de vários factores, por exemplo, se existem luzes que afectam o fragmento, ou se o fragmento faz uso de texturas. Esta fase é chamada de shading e existem vários algoritmos de diferentes complexidades que dão resultados melhores ou piores de acordo com o seu custo de processamento.

Nos primeiros jogos as fórmulas utilizadas simplificavam bastante as equações necessárias pois ainda não existia poder de processamento suficiente para utilizar técnicas mais avançadas. A informação era normalmente processada para cada vértice (per-vertex shading) e depois interpolada em cada fragmento (Gouraud shading). Hoje em dia já temos poder de processamento suficiente para calcular a informação da cor por cada fragmento (per-pixel shading) com técnicas como o Phong shading. Esta fase é normalmente implementada nas placas modernas com shaders, que permitem total controlo e flexibilidade sobre o algoritmo utilizado já que são escritos pelo utilizador.

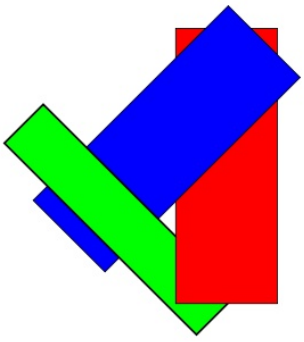
Depois de calculada a cor de cada fragmento, essa cor pode ser misturada com a já existente

no framebuffer (de fragmentos anteriores que ocupam a mesma posição). Esta técnica é chamada de blending e o OpenGL por exemplo permite o controlo das equações utilizadas para efectuar a mistura. É possível assim obter efeitos como transparências parciais ou totais.



O que vemos na figura é um exemplo de blending por adição. Os fragmentos do quadrado verde são misturados com os já existentes do quadrado vermelho, dando origem ao amarelo.

Utilizando a mesma figura, outro conceito importante a perceber é o controlo da profundidade dos objectos. Por exemplo, se tiverem um quadrado à frente de outro, só querem que o quadrado mais próximo da câmara seja mostrado, pois está a tapar o quadrado de trás. A profundidade é controlada com o valor de profundidade do vértice quando projectado no espaço de coordenadas da câmara (eixo dos Z).



Para controlar e saber quais os objectos a mostrar, o OpenGL podia ordenar os objectos e desenhar os que estão mais longe primeiro (o chamado Painter's algorithm), mas isso causa problemas com objectos que se intersectam (ver figura). A solução utilizada pelo OpenGL e Direct3D é ordenar os objectos ao nível do pixel, utilizando uma zona de memória que armazena a profundidade de cada pixel (chamado de depth- ou Z-buffer).

Depois de conhecida então a cor do fragmento, quando um fragmento está para ser escrito no framebuffer, a sua profundidade é comparada com a do fragmento que está no buffer (depth buffer test). Só se o fragmento novo estiver mais perto (se o seu valor for menor) é que o novo valor de cor é escrito no framebuffer.

No fim do processamento de todos os fragmentos, a aplicação mostra o conteúdo do framebuffer no ecrã (normalmente numa janela) ou grava o conteúdo para uma imagem.

Chegamos ao fim do artigo! Embora este artigo já dê uma pequena ideia sobre o mundo que é o 3D, foi apenas uma pequena introdução sobre o tema. Espero que tenham gostado e aprendido com o artigo, e se quiserem aprender mais, podem consultar os seguintes recursos.

Real-Time Rendering
Essential Math for Game Developers
GameDev.net

AUTOR

Escrito por **João Matos**
Também conhecido por triton, frequenta o segundo ano do curso de Engenharia Informática e Computadores no IST. Tem como principais interesses as áreas de Sistemas Operativos, Computação Gráfica e Arquitectura de Computadores.

Quando se é ! Produtivo

Quem ganha a vida a bater código está habituado a ver a sua produtividade medida. Por vezes essa medição não é efectuada de forma correcta e, pior, muitas vezes não é sequer efectuada de forma justa. O resultado é, muitas vezes, uma enorme irritação para com a gestão por esta se refugiar em frases, quase infindáveis, como “Estás quase sempre atrasado nos prazos de entrega”, ou “O desenvolvimento nunca entrega as coisas a tempo e horas”.

As desculpas, ou justificações, por quem faz desenvolvimento são também elas inúmeras, vão do simples “O servidor é muito lento” ao “Estão sempre a mudar as especificações do produto”.

Esta parece uma luta desigual, entre o desenvolvimento e a gestão, mas na verdade apenas o é porque tipicamente o desenvolvimento não faz uso daquilo que melhor tem ao seu alcance: o seu cérebro e o poder de observação.

Se a produtividade pode ser medida, a ineficiência também. E quando se mede a ineficiência então há factos que sustentam a falta de produtividade e que, por sua vez, trazem luz à verdadeira causa dessas ineficiências.

> “Se a produtividade pode ser medida, a ineficiência também”

Quando fazemos este exercício de medir a ineficiência, posso assegurar-vos que a larga maioria das vezes a tal falta de produtividade de que as equipas de desenvolvimento são acusadas cai por terra. Muitas das vezes até é possível mostrar que a verdadeira razão da falta de produtividade vem da equipa de gestão.

Mas então, como encontrar e medir essas ineficiências? É aqui que entra o poder de observação e o cérebro.

O primeiro passo é identificar todas as causas que nos levam ao desespero quando fazemos desenvolvimento. Já me deparei com coisas tão variadas como intromissões de anti-vírus, falta de permissões no sistema de desenvolvimento, ou uso de máquinas virtuais.

Para exemplificar, no caso do anti-vírus o mesmo não podia ser configurado pelo utilizador para não verificar os directórios referentes ao código fonte, ao código compilado, às bibliotecas usadas, etc.; enquanto que o clássico caso de falta de permissões, faz com que qualquer alteração necessite de uma longa interacção entre efectuar um pedido para a alteração e o mesmo ser satisfeito; já o caso da máquina virtual, há quem tenha de desenvolver usando uma máquina virtual que sub-aproveita os recursos da máquina física e, devido à quantidade de camadas intermédias envolvidas, torna toda a interacção com o sistema muito mais lenta.

O segundo passo é encontrar uma forma de medir os problemas identificados e convertê-los numa medida que a gestão compreenda.

Neste caso o tempo é uma das escolhas naturais. Por exemplo, no caso do anti-vírus, o tempo de compilação da aplicação sofreu um aumento significativo, que somado ao longo de uma semana resultava em, aproximadamente,

duas horas de inactividade, ou seja um dia por mês de inactividade só devido ao facto do anti-vírus estar incorrectamente configurado.

Esta situação levou-me a questionar a gestão se eu podia configurar o anti-vírus correctamente e tirar um dia de folga a mais por mês, afinal o resultado seria exactamente o mesmo.

No caso das máquinas virtuais voltei a medir o tempo, mas desta vez as coisas foram mais drásticas. E não era só pelo facto da mesma máquina física ter de correr dois sistemas operativos com dois anti-vírus e desperdiçar 500MB de RAM porque a máquina virtual apenas conseguia alocar 1.5GB de RAM...

Era um conjunto de situações que somado resultava num aumento de quatro vezes o tempo de espera na operação de compilação e deploy da aplicação.

Sim, a produtividade caiu para valores críticos.

Após efectuar esta medição, a extrapolação dos valores obtidos mostrou 33 horas de inactividade mensal. Trinta e três horas por mês num mês que tem quarenta horas de trabalho semanal resulta quase numa semana de inactividade.

O caso tomou proporções interessantes uma vez que esta situação se passou num cliente que tinha mais de oitenta pessoas externas a fazer desenvolvimento nestas condições. Quando se tem oitenta pessoas em regime de outsourcing a quem se pagam 4 semanas de trabalho para trabalharem pouco mais de 3 semanas, o resultado financeiro é desastroso.

> "Após efectuar esta medição, a extrapolação dos valores obtidos mostrou 33 horas de inactividade mensal"

Tudo pode ser medido, pelo que o desafio está em encontrar uma forma inteligente de medir o que está mal e apresentar esses resultados.

Certamente que não é o desenvolvimento que não é produtivo quando um bug ocorreu devido a um requisito mal especificado, ou quando existe um processo moroso para satisfazer uma solicitação, ou quando ocorre uma alteração na especificação do sistema.

Da próxima vez, não se queixem, identifiquem a ineficiência, meçam a ineficiência e apresentem essa ineficiência à vossa gestão.

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.

Client Object Model para Silverlight

Com o lançamento da nova versão do SharePoint, mais propiamente SharePoint 2010 no início deste ano, e o seu sucesso a nível mundial da sua adopção como uma das ferramenta empressariais de Portais e Colaboração, este artigo tem como prespectiva a sua demostração como ferramenta de desenvolvimento para Soluções empresariais.

Neste artigo irei ainda dar atenção a um dos novos serviços disponibilizados pelo SharePoint 2010 que é o Client Object Model para Silverlight, com a criação de um projecto e chamada de métodos para demostrar a sua capacidade de acesso a conteudos existente nos sites em SharePoint.

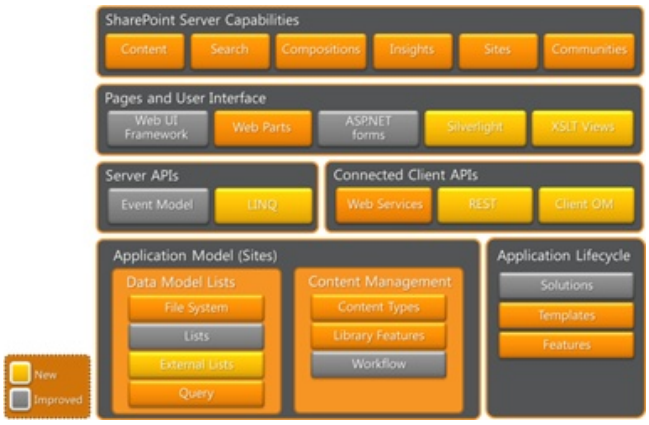
A plataforma SharePoint é um conjunto integrado de recursos que pode ajudar a melhorar a eficácia organizacional fornecendo uma gestão de conteúdo abrangente e pesquisa empresarial, acelerando os processos comerciais compartilhados e facilitando a partilha de informações para uma melhor visão dos negócios.

Os serviços disponibilizados vão desde Gestão de Conteúdos, Gestão Documental, Business Intelligence, Comunidades, Integração de Plataforma externas a Microsoft através dos BCS(Business Connectivity Services), Workflows, Office Web app (Integração das ferramentas Microsoft Office em formato Web), SharePoint Designer 2010, Gestão de Projectos, Serviços de Pesquisa de Portal e Conteúdos Externo, com diversos tipos de autenticação utilização de federações entre outros serviços. Enfim uma plataforma bastante completa.

Durante os últimos anos a plataforma tem crescido a nível de serviço e a nível de importância nas instituições empresariais, tornando uma aposta mais que decisiva da Microsoft, com a sua implementação e integração de todos os seus produtos nesta plataforma, tornando-se o sistema operativo das empresas.

Uma das suas vantagens é a capacidade de integração entre as diversas tecnologias Microsoft e a capacidade de integrações de plataformas externas, como SAP, Oracle entre outras, de forma a torna um ponto de partilha única da Informação da empresa, sendo construido numa plataforma Web o que torna bastante acessível o seu acesso.

No próximo quadro encontram-se os serviços que estão disponíveis para o SharePoint 2010.



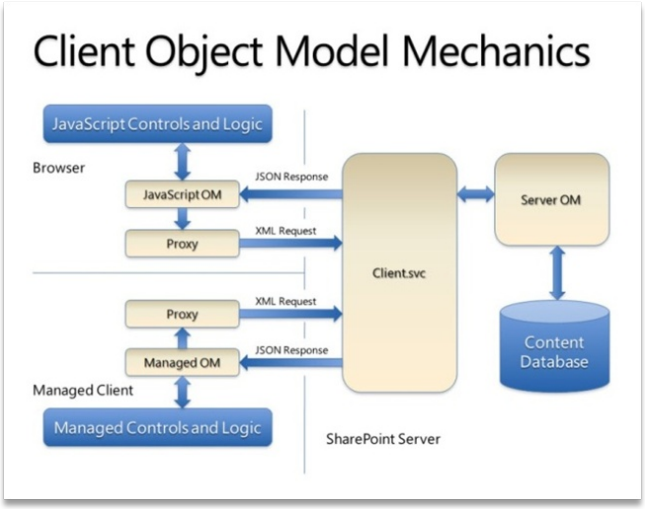
Com o lançamento do SharePoint 2010 foram disponibilizados múltiplos novos serviços como LINQ, novas API's para acesso a dados (utilizando o Client Object Model), utilização de REST - ADO.NET Data Services e utilização de Ajax em webParts com processos Asynchronous (sem utilização de postback nas páginas) entre outros novos serviços.
O SharePoint 2010 apresenta 3 novas Client

Client Object Model para Silverlight

API's para interagir com conteúdos dos sites de SharePoint:

- A partir de aplicações .Net "nunca inferiores a Framework 3.5";
- A partir de aplicações Silverlight "nunca inferiores a Silverlight 2.0"
- A partis de ECMAScript ou seja de Javascript, Jscript

Arquitectura do Client Object Model



Estas novas API em Client Object Model fornecem um sistema orientado a objectos para interagir com os dados do SharePoint remotamente, sendo bastante fáceis de utilizar porque a sua nomenclatura é bastante parecida do Server Object Model do SharePoint.

Para demostrar a forma de acesso aos Objectos da API na próxima tabela mostra a diferentes API's para as diversas plataformas e a forma idêntica da utilização do Object Model.

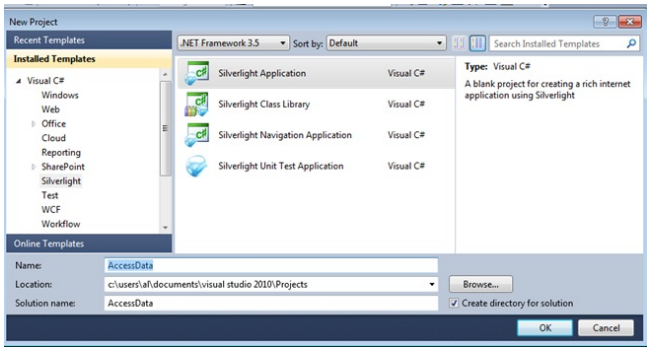
Neste exemplo, irei criar um pequeno Projecto no Visual Studio 2010, com uma aplicação em Silverlight 4, utilizando a API do SharePoint e integrando do SharePoint 2010.

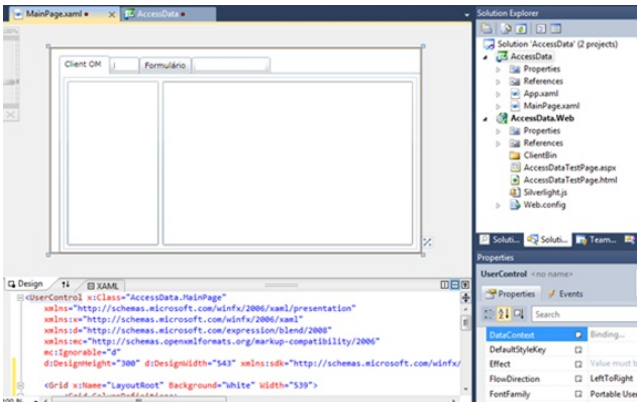
Server (Microsoft .Sharepoint)	.Net (Microsoft.Sharepoint.Client)	Silverlight (Microsoft.SharePoint.Client.Silverlight)	ECMAScript (SP.js)
SPcontext	ClientContext	ClientContext	ClientContext
SPSite	Site	Site	Site
SPWeb	Web	Web	Web
SPList	List	List	List
SPListItem	ListItem	ListItem	ListItem
SPField	Field	Field	Field

Para suporte da nossa solução em SharePoint eu criei uma Lista chamada "*Geral*" com um conjunto de dados para teste.

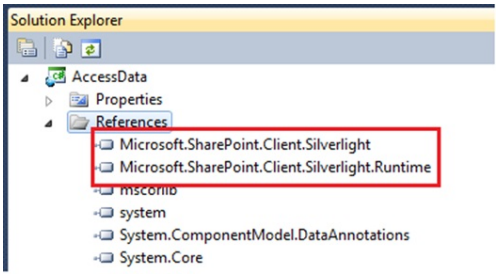
Criar um novo Projecto com o Visual Studio

2010 no Template "Silverlight" e seleccionar a opção para associar a um novo Web site com a versão do Silverlight 4:



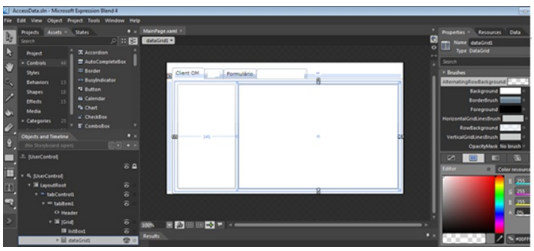


Para realizar as chamadas ao conteúdo do SharePoint, através da API em Silverlight, será necessário adicionar como referencias as DLL's que se encontram na pasta da instalação do SharePoint 2010 (Microsoft.SharePoint.Client.Silverlight e Microsoft.SharePoint.Client.Silverlight.Runtime), tipicamente em C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin.



Futuramente, a Microsoft irá disponibilizar um package com as DLLs em SharePoint .NET Client Object Model, onde se poderá realizar o download e a sua distribuição ao cliente.

Com a utilização do Visual Studio 2010 existe a possibilidade de integrar o Microsoft Expression Blend 4 com as nossas aplicações desenvolvidas em Silverlight.



Depois de termos realizado a nossa customização podemos voltar à nossa solução em Visual Studio 2010 e começar a criar os nosso métodos para retornar conteúdo do Site. Para isso criei 2 métodos, um para retornar a listas do nosso site em SharePoint e outro para retornar o conteúdo de uma Lista numa Datagrid.

Método 1:

Retorna o nome das Listas existentes no site de SharePoint utilizando o Client Object Mode. Foi utilizado uma expressão Lambda para retornar as listas e o Titulo associado.

```
private void LoadLists()
{
    ClientContext clientContext =
        new ClientContext("http://[Site]");
    oWebsite = clientContext.Web;
    collList = oWebsite.Lists;

    clientContext.Load(oWebsite,
       website => website.Title);

//Lambda expression para retornar as Listas
listInfo = clientContext.LoadQuery(
       collList.Include(
        list => list.Title,
        list => list.Fields.Include(
        field => field.Title).Where(
        field => field.Required == true
       && field.Hidden != true)));

clientContext.ExecuteQueryAsync(onQuerySuccee
ded, onQueryFailed);
 }

private void onQuerySucceeded(object sender,

ClientRequestSucceededEventArgs args)
{
    UpdateUIMethod updateUI = DisplayInfo;
    this.Dispatcher.BeginInvoke(updateUI);
}

private void onQueryFailed(object sender,
       ClientRequestFailedEventArgs args)
{
    MessageBox.Show("Request failed. " +
                    args.Message +
                    "\n" + args.StackTrace);
```

```
}

private void DisplayInfo()
{
    collList = oWebsite.Lists;
    listBox1.ItemsSource = listInfo;
    listBox1.DisplayMemberPath = "Title";
}

private delegate void UpdateUIMethod();
```

Exemplo:
http://msdn.microsoft.com/en-us/library/ee538971.aspx

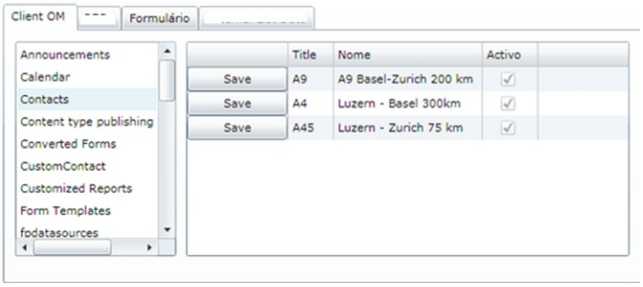
Método 2:

Retorna dados da Lista "Geral" para uma DataGrid utilizando CAML para realizar os filtros dos conteúdos a retornar:

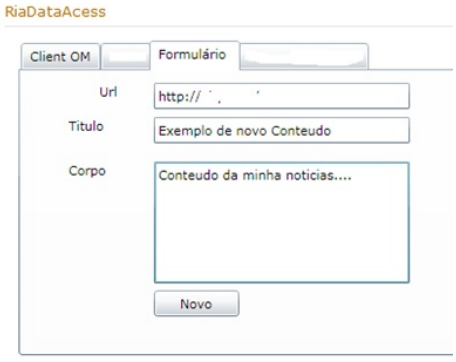
```
private void LoadGrid()
{
    //Contexto do site em SharePoint a chamar
                var    context    =    new
ClientContext("http://[Site]");
    context.Load(context.Web);

      //retornar o objecto List com o nome
"Geral"
                      List     projects     =
context.Web.Lists.GetByTitle("Geral");
    context.Load(projects);

     //CAML Query para Filtar os valores da
nossa Lista
    CamlQuery camlQuery = new CamlQuery();
                     camlQuery.ViewXml     =
"<View><Query><Where><Geq><FieldRef
Name='ID'/>" +
                                        "<Value
Type='Number'>0</Value></Geq></Where></Query>
<RowLimit>100</RowLimit></View>";
                            _projectItems      =
projects.GetItems(camlQuery);
    context.Load(_projectItems);

context.ExecuteQueryAsync(OnRequestSucceeded,
null);
}

#region Methods

   private   void   OnRequestSucceeded(Object
sender, ClientRequestSucceededEventArgs args)
 {
     Dispatcher.BeginInvoke(BindData);

 }
 private void BindData()
 {
       //Utilizando o conteúdo numa lista de
Objecto do tipo Geral como a Lista
     var list = new List<Geral>();
     foreach (var li in _projectItems)
     {
         list.Add(new Geral
         {
             Title = li["Title"].ToString(),
             Nome = li["Nome"].ToString(),
                                    Activo   =
Convert.ToBoolean(li["Activo"].ToString())
         });
     }
          //adicionar os Dados da Lista na
Datagrid
     dataGrid1.ItemsSource = list;
 }
```

No seguinte exemplo foi criado um pequeno formulário no Silverlight para a lista "Anuncios". Esta lista tem um conjunto de artigos de referência ao Site e este método mostra como se consegue criar um formulário de uma forma simples, sem ter a necessidade de realizar

customização de páginas em aspx.



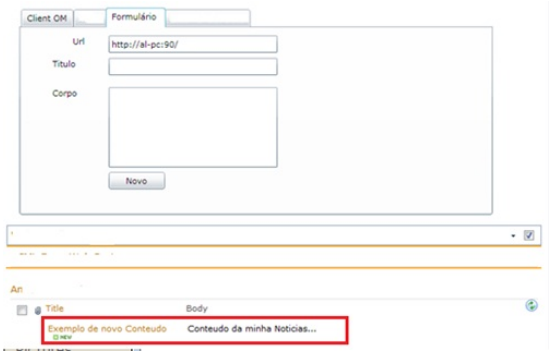
Foram adicionadas na Tab "formulário" do Silverlight, 3 campos de preenchimento do nosso Artigo, como o Url do site onde se destina, Titulo e corpo.



Método 3:

Este método ira criar uma nova entrada na Lista de SharePoint de um novo anúncio.

```
if (!TxtTitle.Text.Equals(""))
            {
                ClientContext clientContext =
new ClientContext(TxtUrl.Text);
                            Web  oWebsite  =
clientContext.Web;
                    ListCollection collList =
oWebsite.Lists;

                                    oList  =
clientContext.Web.Lists.GetByTitle("Anuncios"
);

                        ListItem oListItem =
oList.AddItem(new
ListItemCreationInformation());
                            oListItem["Title"] =
TxtTitle.Text;
                             oListItem["Body"] =
TxtBody.Text;
                oListItem.Update();

                clientContext.Load(oList,
                    list => list.Title);


clientContext.ExecuteQueryAsync(onQueryInsert
Succeeded, onQueryInsertFailed);
            }
            else
            {
                MessageBox.Show("Vazio!!");
            }
```

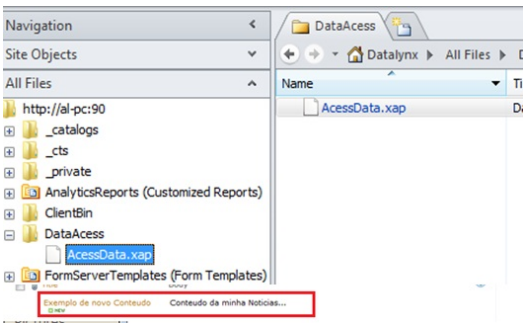


Com a criação deste 2 pequenos exemplos podemos criar a nossa solução "AccessData.XAP".

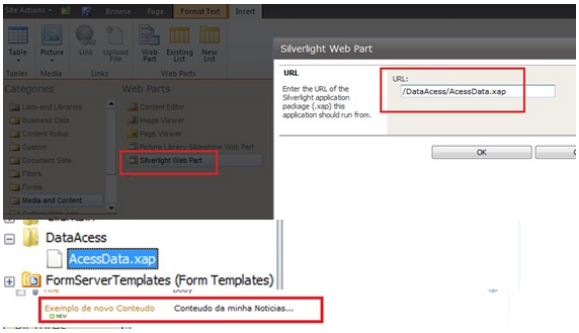
SharePoint Silverlight Web Part

SharePoint disponibiliza ferramentas para integrar o projecto desenvolvido em Silverlight e adicionar nos nossos sites em SharePoint de uma forma bastante simples.
O primeiro passo será adicionar o nosso projecto em Silverlight, neste caso "AccessData.XAP" e adicionar no nosso site em SharePoint, podendo ser uma Folder, uma Document Library ou até na

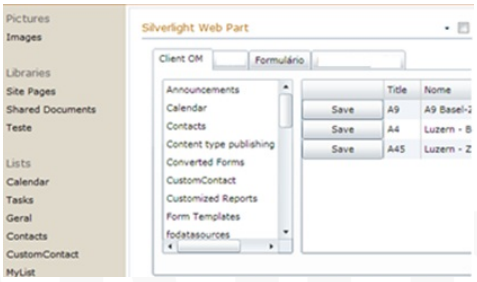
directoria ou pasta “Hive” onde se encontra instalado o SharePoint no servidor. Neste caso criei uma Folder dentro do site em SharePoint. Estas pastas não são visíveis aos utilizadores finais, garantido que o ficheiro não é apagado acidentalmente.



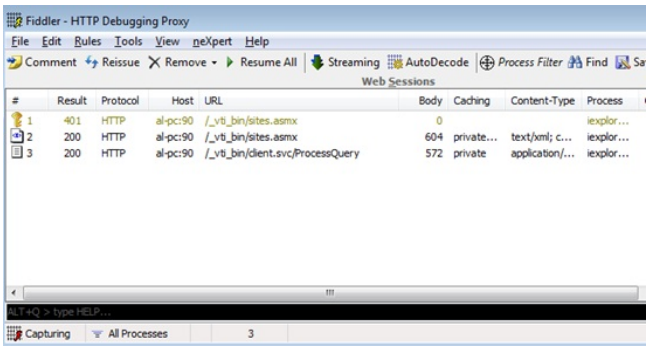
Depois de adicionado a nossa solução em Silverlight, acedemos ao nosso site em SharePoint e adicionamos uma Web Part que se encontra por defeito no SharePoint “Silverlight Web Part”. Um dos parâmetros que é requisitado é o caminho de onde se encontra a nossa solução em Silverlight.



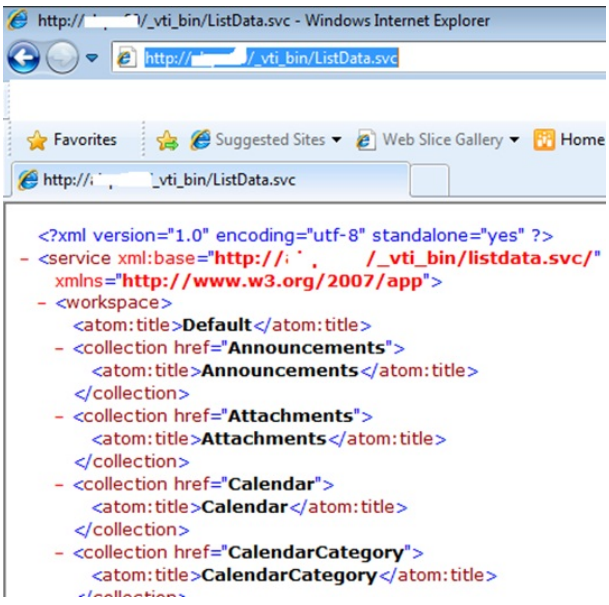
Assim que esteja tudo correctamente adicionado, irá ser disponibilizado na nossa página de internet a nossa solução de Silverlight, dentro do SharePoint, numa WebPart podendo trabalhar e interagir com o conteúdo do Site onde se encontra.



Quando nos encontramos a trabalhar em ambiente Web a aplicação Fiddler é uma ferramenta essencial para captura e validação do nosso tráfego na Web. Estando ela em funcionamento podemos encontrar as transacções realizadas, através da Client OM, e encontramos uma chamada ao WCF Web Service “client.svc” e um Web Service XML para comunicação e transacção das operações.



Existem outra forma de poder interagir com os dados em SharePoint, com o Silverlight através dos “REST” em WCF e com todas as suas potencialidades.



Conclusão

Este foi um pequeno exemplo de como o Client

Object Model e como o SharePoint 2010 consegue fornecer informação e integração com Silverlight.
Teria muita mais que falar, isto é um tema para diversos subtópicos e com múltiplas formas de tratar, apenas quero demostra as portas que podem ser abertas e como o SharePoint 2010 é uma ferramenta de integração com todas as diversas Tecnologias Microsoft.

Alguns links úteis:
Using the Silverlight Object Model
http://msdn.microsoft.com/en-us/library/ee538971.aspx
Using the SharePoint Foundation 2010 Managed Client Object Model
http://msdn.microsoft.com/en-us/library/ee857094.aspx
Using the SharePoint Foundation 2010 Managed Client Object Model with the Open XML SDK 2.0
http://msdn.microsoft.com/en-us/library/ee956524.aspx
SharePoint 2010 Reference: Software Development Kit
http://www.microsoft.com/downloads/en/details.aspx?FamilyID=f0c9daf3-4c54-45ed-9bde-7b4d83a8f26f&displaylang=en
Create a Silverlight 4 Web Part for SharePoint 2010
http://msdn.microsoft.com/en-us/magazine/ff956224.aspx
Página Oficial do SharePoint 2010
http://sharepoint.microsoft.com/pt-pt/Pages/default.aspx

Este e outros artigos relacionados com a Tecnologia Microsoft SharePoint estão disponibilizados pelo site da Comunidade Portuguesa de SharePoint. Podem tornar-se membros activos, criando artigos e a participando em eventos mensais da Comunidade sobre a tecnologia SharePoint.

Comunidade Portuguesa de SharePoint
http://www.sharepointpt.org/

AUTOR

Escrito por **André Lage**
Microsoft SharePoint MVP, trabalhando na Suiça em empresa Farmaceutica e Banca.
Apoiando as diversas Comunidades em SharePoint pelo mundo com participaçoes em eventos como speaker e efectuando artigos com focus nas tecnologias SharePoint. Um dos administradores da Comunidade Portuguesa de SharePoint, Co-fundada por Rodrigo Pinto, www.Sharepointpt.org. Blog pessoal http://aaclage.blogspot.com/

DUVIDAS?

IDEIAS?

AJUDAS?

PROJECTOS?

portugal-a-programar.org