PROGRAMAR

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

EDIÇÃO #30 - AGOSTO 2011

ISSN 1647-0710

RAILS
INTRODUÇÃO AO RUBY ON RAILS

A PROGRAMAR

LUA LINGUAGEM DE PROGRAMAÇÃO PARTE 10

C# ATRIBUTOS EM C#

OBJECTIVE-C INTRODUÇÃO AO OBJECTIVE-C E À PLATAFORMA iOS

SISTEMA RSS NO SHAREPOINT ATRAVÉS DE UMA LISTA DE PÁGINAS

COLUNAS

VISUAL (NOT) BASIC TIPOS GENÉRICOS

COMUNIDADES

SANDBOXED SOLUTIONS EM SHAREPOINT ONLINE 2010 **SHAREPOINTPT**

NHIBERNATE - DO IMPORT-PACKAGE À PRIMEIRA ITERAÇÃO **NETPONTO**

WINDOWS AZURE TRAFFIC MANAGER **AZUREPT**

FERRAMENTAS DE TRABALHO COM SQL SERVER **SQLPORT**

EDITORIAL

EQUIPA PROGRAMAR

ISSN
1 647-071 0

Erro: Título não definido

Cada vez mais é exigido rigor em todos os aspectos nas mais variadas áreas. A programação não é excepção, tanto mais se pensarmos que cada vez mais a tecnologia faz parte da vida das pessoas. Nos dias que correm é natural exigir que qualquer aplicação não possua erros de compilação nem os famosos *warnings*. Todavia estes são sem dúvida os mais fáceis de corrigir. O problema é quando aparecem os denominados erros de *runtime*. Algo que o programador não previu no seu algoritmo e que se pode tornar numa autêntica bomba, capaz de detonar o programa todo. E se pensarmos que numa aplicação de contabilidade, o que poderá acontecer é ser necessário repetir a operação, no caso de uma aplicação de uma nave espacial, por exemplo, o caso de uma divisão por zero que não seja protegida, quando a nave está a calcular o ângulo de entrada na atmosfera, pode determinar a diferença entre o prejuízo de milhões e o lucro, talvez até entre a vida e a morte.

Ao contrário de muitas outras áreas, a programação não é ainda alvo da chamada prova científica, apesar de Informática ser considerada uma ciência. Se perguntarmos a um Engenheiro Civil a razão porque determinada ponte não caí, ele explicar-nos-á com cálculos matemáticos que garantem a estabilidade da ponte. Se perguntarmos à grande maioria dos Programadores ou até Engenheiros Informáticos o porquê do programa funcionar correctamente, muito dificilmente obteremos uma resposta com prova científica.

Para percebermos o parte do drama basta ver problemas de segurança básicos que existem em muitas aplicações na Web e fora dela. Muitas vezes a criação de um programa deixa de ser uma sequência de passos já definidos, para ser inspiração da musa da programação, e onde se escrevem linhas e linhas de código, simples frutos de súbita (des)inspiração. Em vez de programadores passam a poetas, que se inspiram na sua musa, e que em vez de Redondilhas criam “Centilhas”. Muitas vezes sem perceberem que estão a fazer algo e a desfazer nas linhas seguintes, e onde só está a ser gasto processamento.

A verdade é que os erros não vão desaparecer completamente, porque continuámos a ver erros na nossa civilização que custam dinheiro e até vidas. Todavia, e felizmente o número de pontes, aviões e outras invenções não “caem” com tanta frequência como “caem” as aplicações. Porque aí o problema está muitas vezes no ser humano que verificou, e que fez um erro na verificação. Por isso era provável que conseguíssemos melhores resultados se fosse feita uma aposta forte no cálculo e correcção de algoritmos.

António Silva <antonio.silva@revista-programar.info>

ÍNDICE

TEMA DE CAPA

A PROGRAMAR

COLUNAS

COMUNIDADES

EVENTOS

08 Ago - **Damian Conway em Lisboa - Fun with Dead Languages**
10 Ago - **Summer Tech Refresh - Lisboa**
16 Ago - **XIV encontro da Comunidade SQLPort**
27 Ago - **Reunião presencial Comunidade NetPonto - Lisboa**
24 Set - **WordCamp Lisboa 2011 (Comunidade Portuguesa de WordPress)**
24 Set - **2º Aniversário da Comunidade NetPonto**

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

NOTICIAS

Investigadores do ISEP desenvolvem software para tornar os motores de busca mais eficientes

"Uma equipa de investigadores portugueses do Instituto Superior de Engenharia do Porto (ISEP) está envolvido num projecto da Microsoft que pretende desenvolver um novo software, com o objectivo de tornar os motores de busca mais rápidos, mais directos, mais eficazes e mais "inteligentes". A ideia é permitir que os utilizadores da Internet obtenham informação mais concreta e direccionada, na altura da pesquisa. Ou seja, "o objectivo é que os motores de busca sejam capazes de relacionar os parâmetros pedidos pelos cibernautas e forneçam respostas concretas, relacionando a informação como um raciocínio lógico", explicam os investigadores do ISEP. Actualmente, o software implementado nos motores de busca não é capaz de absorver e processar a informação que é exclusivamente destinada ao ser humano. O sistema é apenas orientado para a apresentação de informação. O processamento e as conclusões ficam para o utilizador. O software que está a ser desenvolvido pelo ISEP será capaz de processar os dados pedidos pelo cibernauta e pesquisar os conceitos na web, não de forma isolada, mas relacionando-os entre si. Olhando para um caso concreto, à pergunta "quais os apartamentos T2 existentes, com menos de 10 anos, para arrendar, a menos de 500 metros da Trindade, Pólo Universitário ou Salgueiros, por valor inferior a 500 euros?", o mais certo é surgirem inúmeros sites com referências dispersas a cada um dos parâmetros, mas nenhum site que os relacione entre si, devolvendo-nos todos os resultados que contenham as expressões "apartamentos T2, Pólo universitário, ou Salgueiros". No futuro, o objectivo é que a pesquisa apenas devolva, de facto, a informação acerca dos apartamentos que reúnam todas as condições especificadas e com validade temporal. "

Fonte: Wintech

Microsoft lança Platform Preview 2 do Internet Explorer 10

"A Microsoft lançou esta semana a Platform Preview 2 do Internet Explorer 10. Segundo é revelado pela empresa Norte-Americana, esta nova versão "Preview" apresenta o suporte a HTML5 Parser, HTML5 Sandbox, Web Workers, HTML5 Forms, Media Query Listerners entre muitas outras novidades. Para além destas características, existem também melhorias no desempenho na abertura e apresentação de paginas web. Como é sabido, a versão final Internet Explorer 10 não possui ainda uma data de lançamento, no entanto especula-se que a versão Beta deve chegar em Setembro, altura em que deverá ser revelada também a versão Beta do Windows 8. O Internet Explorer 10 Platform Preview 2 é apenas compatível com o Windows 7 (x86 e x64)."

Fonte: Wintech

Mozilla prepara sistema operativo móvel baseado na Web

"A Mozilla está a trabalhar no desenvolvimento de um sistema operativo aberto e baseado na Web, revelou ontem fonte da fundação, num grupo de discussão para programadores que colaboram com a criadora do Browser Firefox. O projecto a que a Mozilla chama "Boot to Gecko" tem por base o motor de renderização Gecko, usado no browser Firefox e no cliente de email Thunderbird e dirige-se, numa primeira fase, a dispositivos móveis, como os smartphones e tablets. O objectivo é criar "um sistema operativo completo e independente para uma Web aberta", rompendo com o "estrangulamento levado a cabo pelas tecnologias proprietárias no mundo dos dispositivos móveis", afirma Andread Gal, um dos investigadores da Mozilla, alertando para o facto deste ainda se encontrar numa fase muito inicial. "Algumas das suas componentes ainda estão apenas nas nossas cabeças, outras ainda não foram totalmente exploradas. Estamos a falar disto agora porque queremos contar com os conhecimentos de todos na Mozilla - e das pessoas que ainda não fazem parte da Mozilla - para ajudarem a dar forma e construir o projecto que estamos a preparar", explica. Parte do Boot to Gecko - cujo código será disponibilizado "em tempo real" à medida que forem feitos avanços - é baseado no sistema operativo móvel da Google, mas apenas no que respeita ao kernel (ou núcleo) e aos drivers, uma opção que lhe assegura compatibilidade com os dispositivos Android. O desenvolvimento deverá primeiro passar por criar uma plataforma com capacidade para desempenhar as funções básicas de um dispositivo móvel (chamadas, SMS, câmara, USB, Bluetooth) e APIs para lidar com elas. O objectivo passa depois por encontrar uma forma de as páginas Web e aplicações acederem de forma segura aos componentes de que precisem, possibilitando assim a criação de aplicações nativas que corram directamente a partir da Web em vez de estarem disponíveis em apenas alguns dispositivos.

Quem quiser ficar com uma ideia do que esperar, poderá imaginar, numa fase inicial, qualquer coisa como um smartphone com o Firefox para Android como ecrã inicial, com algumas APIs personalizadas."

Fonte: Sapo.pt

Táxis do Porto abrem caminho ao automóvel do futuro

"A tecnologia necessária a mais este avanço da indústria automóvel está a ser estudada em vários institutos e também na cidade do Porto, onde os 448 táxis da cooperativa Raditáxis estão já equipados com um dispositivo experimental desenvolvido em parceria pelas faculdades de Ciências e de Engenharia da Universidade do Porto, pela Universidade de Aveiro, pela Carnegie Mellon University, dos EUA, e pelas empresas NDrive e Geolink."

Fonte: Público /local

Eclipse Indigo lançado com a ferramenta WindowBuilder GUI e EGit 1.0

"A Fundação Eclipse anunciou o lançamento do Eclipse 3.7, nome de código Indigo. A versão mais recente do popular ambiente de desenvolvimento integrado (IDE) open-source introduz alguns novos componentes e funcionalidade melhorada.

O design modular do Eclipse e ênfase na extensibilidade ajudaram a atrair um largo ecossistema em volta do software. É construído e mantido como uma plataforma de ferramentas em vez de ser apenas uma aplicação que funciona sozinha. Uma grande quantidade de funcionalidade especializada está implementada em plug-ins, permitindo ao IDE integrar-se com muitas ferramentas e suportar uma larga abrangência de linguagens de programação e kits de ferramentas de desenvolvimento.

Os componentes do núcleo do IDE e muitos dos plug-ins são desenvolvidos como projectos de open-source individuais dentro da comunidade Eclipse. A Fundação Eclipse organiza um "exercício de lançamento" anual o qual envolve um lançamento coordenada a larga escala que abrange o ecossistema Eclipse. De acordo com estatísticas publicadas pela fundação, 62 projectos Eclipse participaram no exercício de lançamento. O lançamento consiste em aproximadamente 46 milhões de linhas de código de 408 programadores individuais.

Devido ao alcance prodigioso de um lançamento Eclipse, é possível realçar todas as alterações e melhoramentos. Existem contudo, um número significativo de acréscimos que são dignos de nota.

O projecto mais significativo no lançamento Indigo é o WindowBuilder, uma ferramenta sofisticada para construir interfaces gráficos de utilizador com SWT e Swing. O WindowBuilder era anteriormente uma ferramenta comercial desenvolvida pela Instantations, mas foi adquirida pela Google no ano passado. A Google libertou o código fonte e deu-o como contribuição à Fundação Eclipse para que pudesse ser incorporado no IDE como a ferramenta padrão de design do interface do utilizador. A Google tem continuado a desenvolver activamente o projecto desde que libertou o código fonte.

O WindowBuilder suporta o tipo de design visual de interface drag-and-drop que os programadores têm vindo a esperar dos ambientes modernos de desenvolvimento. Possui uma característica de geração de código bi-direccional inteligente que lhe permite interagir com código de interface de utilizador editado manualmente. Tendo esta característica incluída no IDE por predefinição é um grande ganho para programadores Java que criam aplicações gráficas.

A integração do Eclipse com ferramentas de desenvolvimento populares também foi melhorada no lançamento Indigo. Sobretudo, EGit 1.0 foi lançado como parte do ambiente padrão do Eclipse, trazendo o muito necessário suporte fora-da-caixa para o popular sistema de controlo da versão distribuída de git. O EGit é construído em cima da biblioteca JGit, a qual providencia uma implementação Java nativa da funcionalidade de cliente git. Para além do suporte git incluído, o Indigo também oferece uma interacção estreita com a ferramenta de gestão do projecto Maven através do novo projecto m2e.

O Eclipse Indigo está disponível para download a partir do site oficial Eclipse. O Eclipse vem com muitos sabores – existem vários pacotes para download com diferentes configurações e componentes. Obviamente, pode utilizar o cliente Eclipse Marketplace incluído para instalar componentes adicionais independentemente de qual pacote inicial for utilizado. Testámos o pacote padrão de desenvolvimento Java em Mac OS X e concluímos que funciona como esperado.

Para mais informações sobre o lançamento, pode aceder ao anúncio oficial do micro-site Indigo. A Fundação Eclipse está a fazer alguma angariação de fundos a par da actualização. Se quer ajudar a fundação a atingir o seu objectivo de recrutar 500 novos Amigos de Eclipse, pode aceder à página 500 Indigo.

Fonte: http://arstechnica.com/open-source/news/2011/06/eclipse-indigo-released-with-windowbuilder-gui-tool-and-egit-10.ars

(Tradução: Sara Santos)

TEMA DA CAPA

Introdução ao Ruby on Rails

Introdução ao Ruby on Rails

Introdução

O que esperar deste artigo?

Este artigo pretende servir como introdução à construção de websites recorrendo à tecnologia RubyOnRails. Como facilmente se entende, não é possível transmitir todos os pormenores inerentes a um processo desta complexidade num só artigo. Como tal, serão abordados alguns aspectos importantes, que permitirão a alguém com total inexperiência com esta tecnologia, construir uma aplicação web simples com acesso a uma base de dados.

Durante o artigo, e como os aspectos são focados de uma forma muito superficial, existem ligações para que os temas analisados possam ser estudados e compreendidos com maior profundidade. No final do artigo encontram-se também algumas ligações úteis, bem como o local para descarregar o código da aplicação que vai ser construída ao longo do artigo.

O que é isto do "Ruby on Rails" e porque é que me devo "preocupar" em conhecer esta tecnologia?

RoR, muitas vezes abreviado simplesmente para Rails é de facto um conjunto composto por várias tecnologias, de onde se destacam duas: Ruby e Rails. Vamos ver mais à frente e com maior detalhe o que é cada uma das partes.

Por agora, vamos apenas considerar o RoR simplesmente como uma "plataforma" ou "tecnologia" de desenvolvimento web.

Existem várias tecnologias de desenvolvimento web, umas mais conhecidas, como o PHP, ASP clássico, ASP.NET, JSP, … e outras em fase de expansão e ainda não tão difundidas (embora esta tendência esteja claramente mudança), como é o caso do RoR.

Ok, então o RoR é uma plataforma de desenvolvimento web, mas há muitas outras, porque é que me devo interessar em aprender RoR?

Os motivos são inúmeros, por isso vamos apenas focar alguns que fazem parte de um conjunto bem maior, mas que ajudam a perceber que tipo de plataforma de desenvolvimento temos à nossa frente. No final, o facto de gostarmos

ou não desta plataforma acaba também por se reflectir na forma que temos de organizar o nosso trabalho. Vejamos então alguns dos factores que tornam esta tecnologia tão apetecível:

- **É uma tecnologia livre** – O facto de não estar dependente de terceiros (leia-se empresas e tecnologias proprietárias), é para muitos uma vantagem e um factor preponderante na escolha da tecnologia;
- **Utiliza a linguagem de programação Ruby** - Ruby é como uma certa bebida que por aí anda: primeiro estranha-se, e depois entranha-se. Para quem nunca programou em Ruby, a sintaxe pode parecer muito estranha, parecendo que, por vezes, se está a escrever em inglês em vez de programar. Outras vezes parece que estamos a pensar da forma inversa ao que fazemos ao programar em outras linguagens. Depois de algum tempo a programar em Ruby, apercebemo-nos da elegância da linguagem e com o tempo e o acumular de experiência torna-se um forte aliado.
- **Tem uma comunidade grande e activa** - É extremamente simples obter ajuda nas várias comunidades existentes (também em Portugal: ver fim do artigo), e a maior parte das vezes alguém já teve o mesmo erro/problema com que nos deparamos.
- **Arquitectura "Out-of-the-box"** - Quem é que nunca se sentiu "frustrado" ao iniciar a construção de uma aplicação web ou website? Há tanto para fazer que é difícil saber onde começar. Com RoR, ao ser criada uma nova aplicação, a arquitectura está definida, bastando acrescentar os "ingredientes".
- **Sistema de "templating" robusto e versátil** - Não só

permite a renderização de páginas web, mas também XML, JSON e até mensagens de email de uma forma simples e prática.

- **Active Record** - Uma ferramenta de Object Relationship Mapping (ORM) perfeitamente integrada na plataforma, que permite, entre outras coisas, que a base de dados seja gerada a partir do modelo de dados da aplicação, e liberta o programador da tarefa de escrever SQL.
- **Convention over Configuration** - Este pode parecer um ponto controverso, pois se por um lado o facto de grande parte das definições estarem perfeitamente definidas por convenção, pode parecer não existe um controlo tão apertado da forma como as coisas vão acontecer. Tal não se verifica, pois apesar de as convenções existirem para facilitar a tarefa dos Developers, podem ser facilmente substituídas, alteradas e redefinidas, adaptando a plataforma às necessidades de cada Developer e de cada situação.

Instalação

Antes de continuarmos, vamos proceder à instalação da plataforma. Desta forma alguns dos pequenos exemplos de código poderão ser testados à medida que o artigo se vai desenvolvendo. Actualmente a plataforma mais utilizada para desenvolver RoR é Mac OS X, mas vamos demonstrar a instalação em Windows, dado que este será (em princípio) o sistema utilizado pela maioria dos leitores. Contudo ficam também ligações para os sítios onde se poderão encontrar informação sobre a instalação desta plataforma em Mac e em Linux.

Instalação em Windows

A plataforma RoR é constituída por várias tecnologias, nomeadamente:

- Ruby
- Rails
- GIT
- SQLite

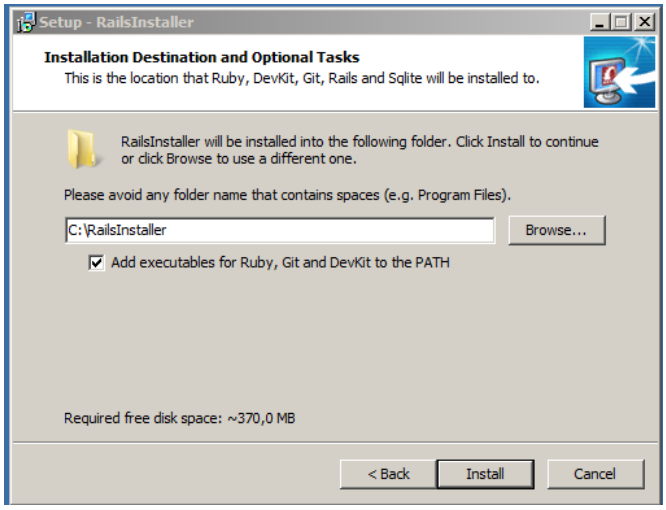
Uma das abordagens seria instalar isoladamente estes itens, mas existe uma forma de instalar tudo isto (e mais ainda) de uma vez só, em Windows.

Essa ferramenta chama-se **"Rails Installer"**:

http://railsinstaller.org

Basta escolher a opção "Download the Kit" e correr o executável que é descarregado.

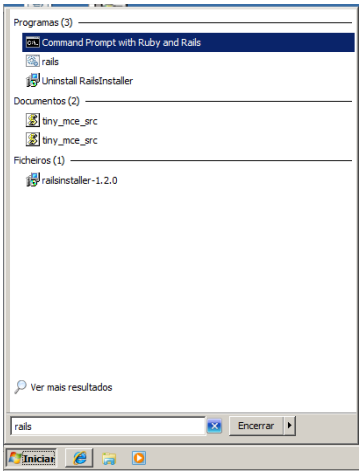
A instalação é simples. Devemos apenas ter em atenção a opção que adiciona os executáveis do Ruby, GIT e DevKit ao Path, para que mais tarde possamos chamá-los a partir da linha de comandos.



De seguida será pedido o nome e endereço de email do utilizador.

Estes dados servem para configuração do GIT (ver edição anterior da Revista) de modo a que possam utilizar repositórios GIT como por exemplo o GitHub (https://github.com).

Através do menu iniciar temos acesso a uma linha de comandos carregada com o PATH dos executáveis necessários ao funcionamento da plataforma:



Um dos comandos que interessa reter para já é o irb. Este comando, disponível na Prompt carregada com as ferramentas do RoR, abre uma consola de Ruby, onde podemos testar alguns dos exemplos que se seguem.

Nota: tanto em Mac como em Linux, é necessário instalar todos os elementos de forma separada, dado que até à data não existe um método de instalação que englobe todos os elementos necessários.

Instalação em Mac

http://pragmaticstudio.com/blog/2010/9/23/install-rails-ruby-mac

http://www.buildingwebapps.com/articles/79197-setting-up-rails-on-leopard-mac

Instalação em Linux (Ubuntu)

http://www.ubuntugeek.com/how-to-install-ruby-on-rails-ror-in-ubuntu.html

Introdução ao Ruby

De uma forma simples e rápida, Ruby é uma linguagem de programação orientada por objectos, interpretada, fortemente tipada e ao mesmo tempo dinâmica, com gestão "automática" de memória. Surgiu no Japão em 1995, pelas mãos de Yukihiro Matsumoto (também conhecido como Matz).

Existem, hoje em dia, várias tecnologias e linguagens baseadas em Ruby, como o JRuby, IronRuby, HotRuby, etc.

A linguagem ganhou alguma notoriedade com o aparecimento da Framework Rails em 2003/2004.

Algumas características do Ruby:

- Todas as variáveis são consideradas objectos

- Todos os tipos primitivos (inteiros, decimais) são classes

- Tudo é verdadeiro (true) excepto false e nil (equivalente ao null de outras linguagens)

- "Garbage Collection" automático

A sintaxe é muito elegante e permite fazer código simples e fácil de ler como este:

```
maioridade = "Maior de idade" unless idade <18
```

A instrução "unless" funciona um pouco como o if, mas com a lógica invertida.

Podemos ler a instrução anterior da seguinte forma: a variável maioridade toma o valor "Maior de idade" a menos que a idade seja inferior a 18. Simples :)

```
3.times do
      puts "Portugal-a-Programar"
end
```

Este é o exemplo mais simples de um loop. A facilidade com que se pode ler permite que, mesmo que nunca se tenha visto Ruby, se perceba facilmente o que este pedaço de código faz. Repare-se que o número inteiro "3", é considerado um objecto como qualquer outro, que tem métodos que podem ser invocados, como neste caso, o método "times".

Temos então, que o bloco de código (que termina no "end") é executado 3 vezes, tal como pode ser literalmente lido, quase como se fosse em inglês. Vamos ver com maior detalhe os loops um pouco mais à frente.

```
if pessoa.registada?
      puts "bem-vindo"
else
      puts "acesso negado"
end
```

Neste exemplo vemos a utilização do ponto de interrogação

na sintaxe. Para que este snippet funcione, a classe do objecto pessoa tem de ter definido o método "registada?". Tipicamente métodos terminados em "?" são métodos que devolve valores booleanos. Para quem tem experiência em Java ou.NET basta pensar nestes métodos como os métodos começados por "Is" (ex: boolean IsRegistada()) que tipicamente devolvem resultados booleanos.

Em Ruby todos os "tipos de dados" são classes. Vamos apenas abordar algumas das classes mais utilizadas, pois seria impossível abordá-las a todas.

O resto da API pode ser consultada aqui: http://www.ruby-doc.org/core/

Numeric

Como facilmente se depreende pelo nome, esta classe, representa valores numéricos. Existem classes derivadas desta, como a **Float**, a **Integer** ou a **BigNum**. Tipicamente irão ser utilizadas as classes **Double** e **Integer**.

Um texto pode ser transformado em inteiro com o método **to_i** e um número pode ser transformado em **string** (texto) com o método **to_s**.

Ex:

```
a = 10
b="20".to_i  * a
```

Referência:
http://www.ruby-doc.org/core/classes/Float.html
http://www.ruby-doc.org/core/classes/Integer.html

Strings

Como se depreende pelo nome, esta classe representa objectos de texto. Uma string pode delimitar-se com aspas (") ou plicas ('). O operador de concatenação é composto por dois sinais de menor (<<).

Ex:

```
pais="Port"
pais << 'ugal'
puts pais
```

Resultado: Portugal

```
frase="hoje está muito calor"
puts frase.capitalize
```

Resultado: Hoje está muito calor

Referência:
http://www.ruby-doc.org/core/classes/String.html

Arrays

Um array, em Ruby, é definido pela classe Array. O primeiro índice é o 0 (zero). É possível utilizar o índice -1 (menos um) para aceder ao último elemento do Array, -2 (menos dois) para o penúltimo, etc.

Ex:

Inicialização de um array vazio

```
dados = Array.new
```

Inicialização de um array com dois elementos

```
dados  =Array.new(1,2)
```

A adição de elementos ao array, faz-se com o operador << (duas vezes o sinal de menor):

```
dados = Array.new(2,3,4)
dados << 5
```

Resultado: [2,3,4,5]

Referência:
http://www.ruby-doc.org/core/classes/Array.html

Hash

Um Hash é uma colecção de pares chave-valor. Funciona de forma parecida a um Array, com a diferença de que o índice é substituído pela chave. É muito frequente o uso de Hashes em Ruby (e também em Rails), e muito utilizados juntamente com Symbols (abordados de seguida).

Um Hash, define-se da seguinte forma:

chave => valor (sinal de igual e maior)

Ex:

```
hash = { "a" => 1, "c" => 2 }

puts hash[“a”]
```

Resultado: 1

Referência:
http://www.ruby-doc.org/core/classes/Hash.html

Symbol

Este será talvez um dos conceitos mais difíceis de entender e de explicar, pois não tem um paralelo directo em outras linguagens.

Um símbolo serve para representar “nomes” e “textos” perante o interpretador de Ruby. Um símbolo é sempre constante num determinado contexto, no entanto pode representar num certo contexto uma classe, em outro contexto um método, e ainda num terceiro contexto diferente uma variável, mas será sempre o mesmo objecto, da classe Symbol.

A definição de um objecto do tipo Symbol em Ruby tem sido alvo de discussões, mas parece que a definição resultante indica que um símbolo deve ser usado sempre que é necessário referenciar um nome (identificador, palavra-chave) mesmo que estejamos a falar de algo que não existe ainda no nosso código. Confuso? Sim, concordo. Contudo com a utilização de símbolos, o seu propósito vai ficando mais claro e torna-se até intuitivo.

Um símbolo define-se colocando dois pontos (:) antes do seu nome. Os Símbolos são frequentemente usados como chaves em hashes.

Referência:
http://www.ruby-doc.org/core/classes/Symbol.html

Definição de Classes e métodos

Uma classe define-se da seguinte forma:

```
class NomeDaClasse

      #corpo da classe

end
```

Como é possível ver, o cardinal (#) define um comentário de uma linha.

Vamos então acrescentar um método a esta classe que receba dois parâmetros e que devolva a sua soma (assumindo que são números):

```
def soma (a,b)

      a+b

end
```

Um método define-se com a palavra def. A última instrução de um método é considerada o seu retorno, mas pode-se usar também a palavra return para melhor legibilidade.

A herança entre classes faz-se utilizando o operador menor (<). Imagine-se que temos a classe Automovel que “herda” ou “estende” a classe Veiculo:

```
class Automovel < Veiculo

end
```

Loops

De uma forma breve, alguns exemplos de ciclos em Ruby.

While

```
while condicao do

   #instruções

end
```

No caso de ser apenas uma instrução, podemos uma versão ligeiramente diferente, que torna o código bastante elegante e fácil de ler:

```
instrucao while condicao
```

Until

```
until condicao do
   #instruções
end
```

For

O ciclo for pode ser utilizado com intervalos ou com colecções de elementos (arrays, hashes). Um intervalo define-se da utilizando entre o valor inicial e final dois pontos seguidos (..) :

1..10 (1 ponto ponto 10) - Especifica um intervalo de 1 a 10 (inclusive).

Ex:

```
for i in 1..10 do
      #instruções
end
```

```
for i in [1,3,5,6,8] do
      #instrucoes
end
```

Each

```
coleccao.each do
      #instrucoes
end
```

Basicamente o método each, percorrer todos os elementos da colecção e executa o código contido dentro do bloco. Quem utiliza o foreach em outras linguagens está certamente a notar que falta algo.

Como é que referenciamos o "elemento actual" dentro do bloco? Mais do que referenciar, temos de "passar" para dentro do bloco o item actual. Para isso, rodeamos por barras verticais ou pipes (|) o nome pelo qual queremos referenciar o objecto dentro do bloco.

Uma forma engraçada de olhar para esta notação, tal como é possível ler no livro "Why's (poignant) Guide to Ruby", é que podemos olhar para estas barras verticais como uma pequena porta, por onde o valor entra para dentro do bloco de código.

Ex:

```
coleccao = Array.new(1,2,3)
coleccao.each do | item_actual |
      puts item_actual
end
```

Resultado:

1
2
3

Extensibilidade

Imagine-se que necessitamos de estender a classe "Integer" e adicionar um método que diga se um número é positivo ou não. Vejamos então como faze-lo:

```
class Integer
      def positivo?
          self>0
      end
end
```

Tão simples como isto. Este tipo de alteração/acrescento de funcionalidades em *runtime* designa-se de *Monkey Patching*.

Em outras linguagens (não dinâmicas) este tipo de abordagem iria gerar um erro de "Classe Duplicada", e a solução seria estender a classe, criando uma nova classe que herdasse da classe integer e criar os novos métodos ou fazer override dos métodos já existentes na classe base.

Neste caso o método "positivo?" será um "aumento" da classe Integer, e passará a fazer parte da classe.

Como é possível ver, a palavra self diz respeito ao próprio

objecto em si (o bem conhecido “this” do Java, C# e PHP).

Ruby Gems

Uma gem não é nada mais, nada menos do que uma biblioteca ou programa empacotado. Todas as gems têm um nome e uma versão. Pode ser feito um paralelismo com os ficheiros .dll no .NET ou os .jar no Java, embora existam diferenças.

As Gems podem ser instaladas de forma fácil, bastando para isso existir uma ligação à internet. A sua instalação funciona através de repositórios, de onde são descarregadas e instaladas.

Ao instalar o rails no Linux ou em Mac, foi necessário, a certa altura, executar este comando:

gem install rails

Este é o comando para instalação de novas gems. A gem é trazida do repositório e instalada.

É possível ver as Gems instaladas através do comando:

gem list

É possível criamos as nossas próprias gems e até submetê-las para o repositório, para que outros as possam utilizar.

Mais informação sobre as RubyGems:

http://rubygems.org/

http://sirupsen.com/create-your-first-ruby-gem-and-release-it-to-gemcutter/

Overview do MVC

Antes de passarmos à análise da framework Rails, que serve, juntamente com o Ruby, de base a esta tecnologia, vamos ver (ou rever, caso seja já um conceito familiar) como funciona uma aplicação MVC, dado que a framework funciona com base neste padrão.

A Sigla MVC deriva de Model-View-Controller. O MVC é um padrão de desenho (design pattern) utilizado na criação de aplicações informáticas. Uma das principais preocupações do MVC é a separação de responsabilidades (muitas vezes referida nos livros e pela comunidade por Separation Of Concerns, ou simplesmente SoC).

Existem, assim, três partes distintas do projecto, cada uma com responsabilidades próprias, e que interligadas permitem não só o funcionamento eficaz dos projectos, mas também facilitam de uma forma muito eficaz a manutenção dos projectos.

O Model, muitas vezes chamado de Domain visa representar o modelo de dados do projecto. É também aqui grande parte da lógica deve residir, de modo enriquecer o modelo de dados. É também o Model o responsável pela persistência de dados. Muitas vezes são utilizadas ferramentas de ORM acopladas ao Model, de modo a facilitar esta tarefa: NHibernate, Entity Framework, Hibernate, e o ORM que nos vai interessar particularmente: Active Record.

Um Controller é um ponto de interacção do utilizador com a aplicação. Um controlador é constituído por acções (Actions) e cada acção representa um método. É o controlador o responsável por validar dados e por fazer as chamadas necessárias ao Model de modo a passar-lhe dados ou simplesmente, a trazê-los para a aplicação. Um controlador invoca as Views de modo a poder apresentar os dados ao utilizador.

Uma View é a representação visual da aplicação. Pode receber dados do Controller, e pode até ser uma representação visual do Model. É sempre invocada por uma Action dentro de um Controller. É aqui que reside a parte visual da aplicação.

Como é possível perceber a partir da explicação anterior, existe uma total separação de conceitos, sendo que é possível alterar a interface da aplicação (alterando as Views) sem que o resto da aplicação sofra qualquer alteração. Da mesma forma, se existirem alterações nas regras do negócio, será o Model (e possivelmente também os Controllers) a sofrerem alterações, mas as Views manter-se-ão intactas, deixando a interface intacta. Se for necessário alterar a forma de persistência (por exemplo, trocar de motor de base de dados), será a parte de persistência do Model a única secção do projecto a sofrer alterações. Esta tarefa fica ainda mais simplificada se for utilizado um ORM com suporte a múltiplos motores de base de dados.

Fica um vídeo engraçado que ajuda a perceber a vantagem da organização MVC:

http://www.youtube.com/watch?v=p5EIrSM8dCA

Como funciona então o MVC?

O MVC funciona através de rotas. Mas o que é uma rota?

Com uma tecnologia de desenvolvimento web convencional (e por convencional leia-se, não MVC), o url www.dominio.com/pasta/ficheiro.extensao leva-nos até um ficheiro chamado "ficheiro.estensao", que se encontra realmente dentro de uma directoria chamada "pasta", no servidor indicado por aquele endereço. Em MVC isto não acontece assim. Os caminhos são "virtuais" e são resolvidos pelo componente de gestão de rotas da plataforma. Em Rails e em grande parte das Frameworks MVC, a rota padrão é: Domínio/Controller/Action/Id, também representado por / Controller/Action/Id.

Exemplo: www.dominio.com/Clientes/Detalhes/10

Como se deve ler url "à moda" do MVC?

Fazendo o paralelismo com o mapeamento acima, facilmente se percebe o seguinte:

Controller: Clientes
Action: Detalhes
Id: 10

Então temos que estamos a invocar a acção "Detalhes", que é um método do Controlador "Clientes" e estamos a passar-lhe o id 10.

O que esta Action provavelmente faz é aceder ao Model e trazer os dados do cliente com o id 10. Depois deve passar os dados, devidamente formatados, para uma View, que é invocada pela Action.

Não existe, necessariamente, nenhuma pasta chamada "clientes", ou "detalhes", nem um ficheiro com nome "10", mas sim uma rota que é capaz de pegar num url e de o transformar numa chamada a um método (Action) de uma classe (Controller) passando-lhe um (ou mais) parâmetros.

Este tipo de url's permite uma transposição directa da estrutura dos métodos (públicos) dos controllers, o que faz com que seja muito utilizada com webservices RESTfull.

É ainda possível criar rotas personalizadas, de modo implementar práticas SEO, mas não serão abordadas neste artigo.

Passemos então ao Rails, enquanto Framework e plataforma de desenvolvimento web.

Quickstart Rails

Como já foi referido anteriormente, Ruby On Rails é constituído (entre outras coisas) por Ruby (a linguagem) e Rails (a Framework).

A framework - actualmente na versão 3.0 - surgiu em 2004 pelas mãos de David Heinemeier Hansson (http://david.heinemeierhansson.com/).

Utiliza o padrão MVC e disponibiliza, *out-of-the-box*, funcionalidades como *scaffolding* (abordado mais à frente), um ORM (Active Record), um servidor web (webBrick), entre outras funcionalidades e ferramentas, que permitem aos *developers* criarem, de uma forma rápida, aplicações web.

Baseia-se em duas filosofias de funcionamento:

- "Convenção sobre a Configuração" (Convention over configuration) onde se assume um conjunto de convenções (algumas são abordadas de seguida) de forma a que o developer não tenha de se preocupar com a configuração de aspectos que, na maioria das vezes, são configurados sempre da mesma forma. Por exemplo, se existe uma classe no Model chamada "User", é muito provável que a tabela na base de dados se vá chamar "Users". E de facto é esta a tabela utilizada pelo Active Record, por convenção, para esta classe. Podemos sempre ultrapassar as convenções e configurar o que for necessário, pois "Convention over Configuration" não significa "Convention instead of Configuration".
- DRY – Significa Don't Repeat Yourself, mas também pode ser lido na forma literal, como código "seco". Código Seco (dry code) significa a arquitectura está tão bem estruturada que não existe repetição de código. A repetição de código leva a que por vezes um bug seja repetido com a replicação do código, o que torna a sua correcção mais difícil, e por vezes é corrigido num local e permanece em outro. Ao evitar a repetição de código (princípio DRY), toda a aplicação se torna mais legível, fácil de gerir e de modificar.

Convenções

De acordo com o que referido atrás, a Framework Rails utiliza uma filosofia de "Convention over Configuration". Por esta razão é importante conhecermos algumas das convenções (também conhecidas como sensible defaults), de modo a que melhor se possa perceber o comportamento da plataforma.

Existem diferentes tipos de convenções, e seria impossível abordá-las todas aqui. No entanto ficam algumas das mais importantes para a fase de introdução à plataforma:

Base de Dados:

- nomes de tabelas sempre no plural. Ex: customers, para a classe Customer
- chave primária é o campo id
- chave estrangeira é constituída pelo nome da entidade, seguida por underscore e "id". Ex: customer_id

Model:

Modelos com mais do que uma palavra usam Pascal Case no nome da classe, e undescore a separar as palavras (que devem utilizar apenas letras minúsculas) no nome do ficheiro. Ex:

Nome da classe: CustomerOrder

Nome do ficheiro: customer_order.rb

Existem bastantes mais convenções. Algumas serão ainda abordadas durante a parte prática do artigo, que pode ser vista de seguida.

Criação de uma aplicação em RoR

A criação de uma aplicação em Ruby on Rails implica um uso constante da linha de comandos. Para quem está habituado a que o IDE dê uma ajuda nas tarefas de desenvolvimento, este paradigma pode parecer um pouco "contra-natura". Contudo, e como em muitas outras situações, acaba por se tornar usual e bastante prática.

Tarefas como criar uma nova aplicação Rails, adicionar uma entidade ao Model, criar um novo Controller, criar um Scaffold (abordado de seguida) ou até iniciar e parar o servidor, implicam o uso da linha de comandos, e implicam também o conhecimento de comandos específicos, com uma sintaxe própria.

Nota: existem IDE's que abstraem o developer da utilização da linha de comandos, facultando interfaces gráficas mas que no fundo vão chamar os comandos que teriam de ser executados na consola.

A aplicação que será construída de seguida, como forma de demonstração da tecnologia, será construída apenas com um editor de texto e com a linha de comandos, sem recurso a qualquer IDE, de forma a que possa ser demonstrado, de uma forma mais limpa, o processo de criação. No dia-a-dia, é de todo, aconselhável a utilização de um IDE, com *auto-completion* e *syntax highlighting*.

Aptana Studio 3 – (grátis) http://www.aptana.com/

RubyMine – (Grátis para projectos OpenSource) - http://www.jetbrains.com/ruby/

Passemos então à criação da aplicação. O primeiro passo é definir o local onde a aplicação será armazenada. Neste caso a aplicação irá ficar na directoria "C:\Sites". De seguida é necessário abrir a linha de comandos.

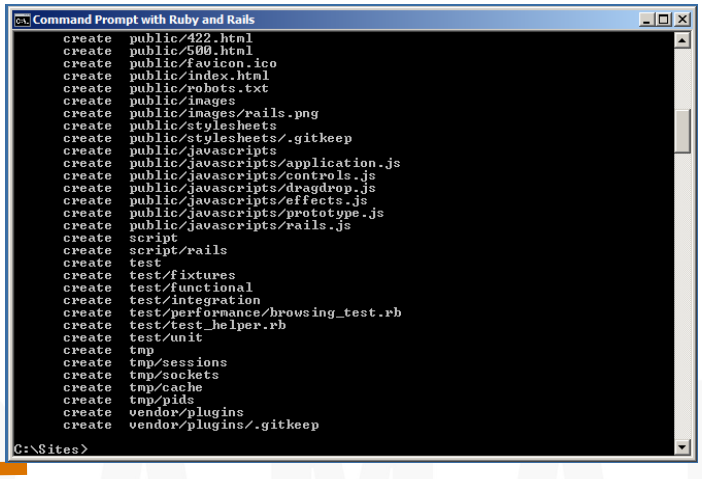
Uma vez dentro da directoria desejada, o comando responsável pela criação da aplicação é o comando:

rails new nome_da_aplicacao

Uma vez executado este comando, serão criados vários ficheiros e pastas dentro de uma pasta com o nome da aplicação. Neste caso vamos dar o nome "lista_de_compras" à nossa aplicação:

rails new lista_de_compras

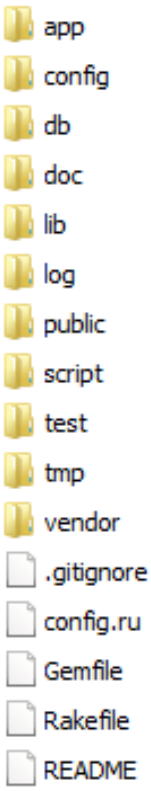
Após a execução deste comando, é possível ver que o script colocou algumas mensagens na consola, indicando a criação dos ficheiros e pastas da aplicação.



Abrindo a pasta com o nome da aplicação, podemos observar e analisar os ficheiros e pastas que foram utilizados:

Vamos então ver o que cada directoria e ficheiros significam e qual o seu propósito.

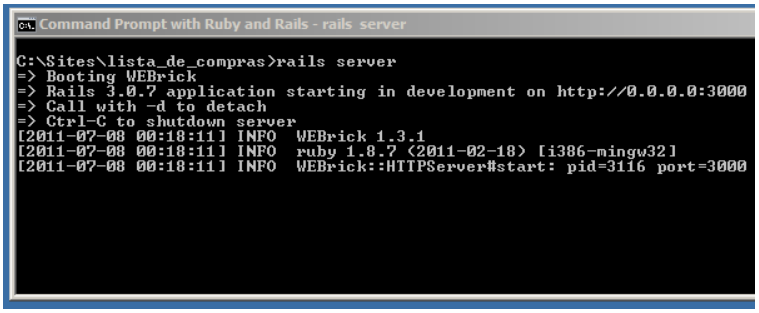
- Pasta App: É aqui que a aplicação reside. Todos os controllers, views, models, helpers residem aqui dentro. Falaremos dela mais à frente à medida que construímos a aplicação.
- Pasta Config: Como o próprio nome indica é aqui que residem as configurações da nossa aplicação. Iremos também abordar alguns ficheiros contidos nesta pasta de seguida.
- Pasta Db: Tipicamente uma aplicação Rails vai utilizar uma base de dados relacional. É aqui que ficam os scripts de migração/criação da base de dados, e no caso de a base de dados ser no formato de ficheiro, o ficheiro da base de dados ficará também aqui.
- Pasta Doc: Existe uma Framework chamada RubyDoc que permite gerar documentação a partir do nosso código. É aqui que a documentação fica armazenada.
- Pasta Lib: É onde as bibliotecas serão adicionadas (excepto os plugins/bibliotecas proprietárias, que serão armazenadas na pasta Vendor).
- Pasta Log: É aqui que as mensagens de Log serão guardadas.
- Pasta Public: Este é o local onde são guardadas as imagens, folhas de estilo (CSS) e scripts client-side (Javascript), bem como todas as páginas estáticas partilhadas pela aplicação: página 404 (not found), etc.
- Pasta Script: é onde reside o script responsável pela geração de itens (views, models, controllers, etc). Este script apenas chama outros scripts/executáveis, não sendo ele o verdadeiro responsável pela geração/destruição dos elementos.
- Pasta Test: A Framework anda de mão dada com Test Driven Development, o que significa que a criação de testes unitários é encorajada. É aqui que ficam os testes, bem como os dados fictícios (Mock).
- Pasta tmp: Pasta utilizada pelo Rails para armazenamento temporário, auxiliar ao processamento.
- Ficheiro .gitignore: Este ficheiro não está directamente relacionado com o Rails, mas sim com o GIT. Quando utilizamos um repositório de código existem certos ficheiros que não interessa enviar e guardar: ficheiros temporários, ficheiros gerados/compilados (não acontece aqui, pois Ruby é interpretado), entre outros. O ficheiro .gitignore indica quais os ficheiros / tipos de ficheiro devem ser ignorados nos commits e push's do GIT. Para mais detalhes, ver o artigo sobre GIT, na edição anterior da revista.
- Ficheiro config.ru: Juntamente com os ficheiros contidos na pasta config, este ficheiro é utilizado na configuração da aplicação.
- Ficheiro Gemfile: É aqui que são definidas as RubyGems utilizadas pela aplicação. Por exemplo, se a aplicação utiliza uma base de dados MySQL, é necessário adicionar a Gem do MySQL, para que o Rails consiga comunicar com o MySQL.
- Ficheiro Rakefile: Semelhante ao Makefile usado em Unix/C. É o ficheiro que contém informações sobre o "empacotamento" da aplicação. É consumido pelo utilitário rake, que vem contido na instalação.



Terminada a análise à estrutura da pasta da aplicação, é altura de iniciar o servidor e ver pela primeira vez a aplicação a correr. Para tal basta executar o comando:

rails server

Nota: este comando terá de ser executado dentro da pasta da aplicação, caso contrário não será iniciado o servidor.



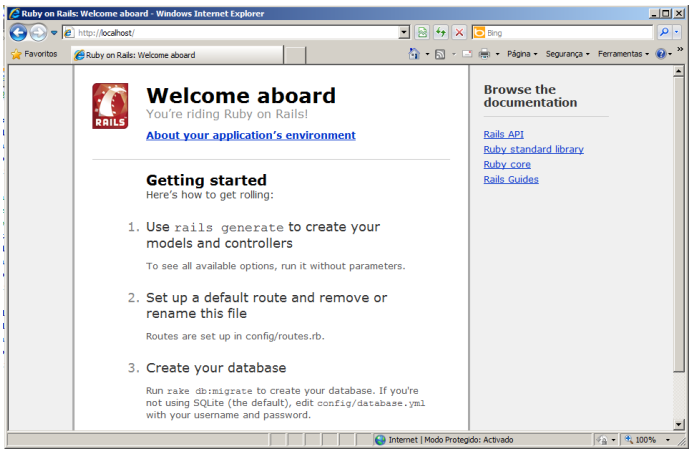
Para terminar o servidor, basta utilizar a combinação de teclas Ctrl+C.

Como é possível ver, o servidor webBrick é iniciado na porta 3000. Se quisermos utilizar a porta padrão para um servidor web (porta 80), basta adicionar "-p 80" ao comando:

rails server –p 80

Nota: Será esta a porta usada durante o resto do artigo.

Desta forma podemos aceder à nossa aplicação, simplesmente colocando o endereço http://localhost no browser:



É possível ver o ambiente da aplicação, clicando na ligação "About your application's environment". Aqui podemos conferir a versão dos "elementos" do nosso ambiente, bem como saber qual dos ambientes está actualmente activo (abordado com maior detalhe mais à frente).

Ainda na página é possível ver 3 passos para que a aplicação comece a funcionar como deve ser.

1 – Criar controllers e models com o comando "rails generate"
2 – Registar rotas
3 – Criar e configurar a base de dados

É, agora, altura de definir o que a aplicação vai fazer: será uma aplicação bastante simples que irá permitir a criação de várias listas de compras. Cada lista de compras terá um nome e uma prioridade. Uma lista poderá ter vários itens, sendo que cada item terá um nome, um preço estimado e uma quantidade.

O objectivo é construir uma aplicação simples que permita a criação, alteração e remoção de listas de compras e dos seus itens. Irá ser utilizado o motor de base de dados SQLite, dado que vem com a instalação da Framework.

Vamos começar pela criação de um Controller chamado "Inicio". A criação de controllers faz-se com o comando:

rails generate controller nome_do_controller

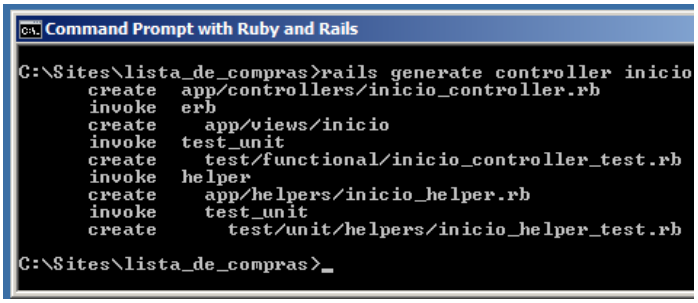
Ao correr este comando, são executadas várias acções, das quais interessa destacar a criação de um ficheiro dentro da pasta App/Controllers, que contém a classe do Controller.

É também criada uma pasta com o mesmo nome dentro da pasta das Views (App/Views).

Por convenção, todas as views de um controller, devem estar dentro de uma pasta com o nome desse controller (estando esta pasta dentro da pasta views).

Vamos então gerar o controller Inicio (se o servidor estiver em execução é necessário pará-lo):

rails generate controller inicio



Abrindo a pasta App/Controllers, é possível ver que foi criado um ficheiro chamado inicio_controller.rb, que é o ficheiro onde o controller está definido:

```
class InicioController < ApplicationControler

end
```

Este ficheiro contém uma classe chamada InicioController. Interessa relembrar a naming convention onde no nome dos ficheiros as palavras são separadas por underscore (inicio_controller.rb) e no nome das classes/módulos é utilizado Pascal Case (InicioController).

Esta classe estende (herda de) a classe ApplicationControlller.

Todos os controllers estendem esta classe, e por convenção, o nome utilizado na criação é sufixado pela palavra "Controller", gerando assim o nome da classe.

Como vimos anteriormente, um controller tem métodos, aos quais podemos chamar de "Actions".

Vamos definir um método vazio chamado “ola”:

```
class InicioController < ApplicationControler
     def ola
     end
end
```

Como vimos anteriormente, pelo padrão do MVC, conseguimos chegar a uma action utilizando o seguinte url: http://servidor/controller/action, logo ao abrir o url http://localhost/inicio/ola devemos chegar a esta Action. Vamos iniciar o servidor e tentar abrir este url no browser.

Routing Error

```
No route matches "/inicio/ola"
```

Pois é, não funciona. O erro diz-nos que não existe ainda nenhuma rota definida que coincida com /inicio/ola.

Vamos então definir a rota “standard” do MVC: /controller/action/id. Para isso, vamos até à pasta config, dentro da pasta da aplicação, e vamos abrir o ficheiro routes.rb. Vamos alterar o ficheiro de modo a adicionar este mapeamento. Possivelmente a rota está comentada no fim do ficheiro, pelo que basta descomentar (ou adicionar no caso de não existir) a seguinte linha:

```
ListaDeCompras::Application.routes.draw do
  match ':controller(/:action(/:id))'
end
```

É necessário reiniciar o servidor, de modo a que a nova rota tomada em conta. Abrindo o url anterior, agora o erro é diferente:

Template is missing

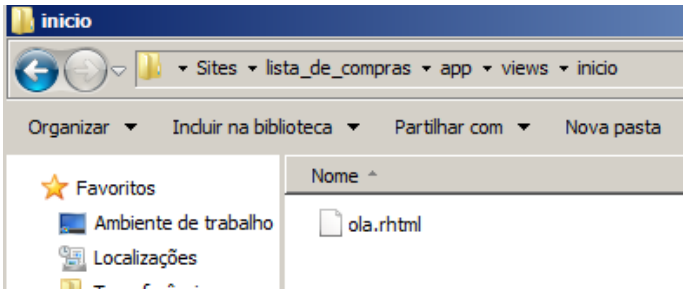
Missing template inicio/ola with {:locale=>[:en, :en], :formats=>[:html, :text, :js, :css, :ics, :csv, :xml, :rss, :atom, :yaml, :multipart_form, :url_encoded_form, :json], :handlers=>[:rhtml, :rxml, :builder, :erb, :rjs]} in view paths "C:/Sites/lista_de_compras/app/views"

O que aconteceu, é que o **controller** “inicio” foi encontrado, a **action** “ola” também foi encontrada, e por convenção, esta Acção tem uma Vista associada com o mesmo nome. Ainda por convenção, a vista estará dentro de uma pasta com o mesmo nome do **controller**, dentro da pasta das Views (neste caso o erro indica que esperava encontrar um ficheiro chamado “ola” dentro da pasta views/inicio).

A View não é criada automaticamente, nem existe um script “generate” para Views. Teremos, por isso, de a criar à mão.

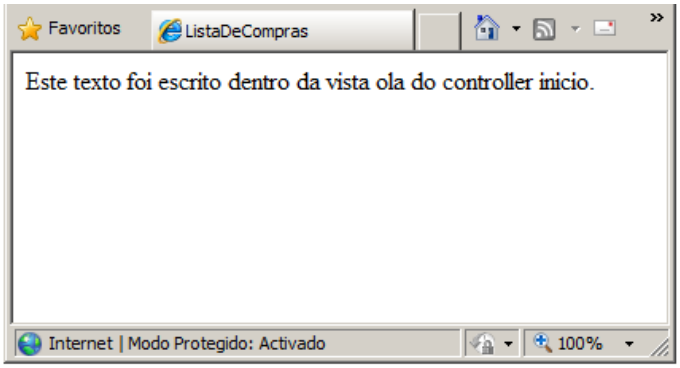
Uma view tem extensão .rhtml (ruby-html) e contém código Ruby e HTML. Uma vez que a Action se chama “ola”, a menos que indiquemos explicitamente na Action, que a View tem outro nome, o nome da View será “ola.rhtml”.

Vamos então criar um ficheiro vazio, chamado “ola.rhtml”, dentro da pasta inicio, que se encontra dentro da pasta **views**, dentro da pasta **app**:



Vamos agora abrir o ficheiro no editor de texto e adicionar a seguinte frase: *“Este texto foi escrito dentro da vista ola do controller inicio.”*

Depois de guardar o ficheiro, vamos recarregar a página que antes estava a mostrar o erro, e deverá ser possível ver o seguinte:



Já funciona :) !!

Vamos adicionar uma nova rota, de modo a que esta seja a action e controller a ser chamados quando apenas se coloca o endereço base (http://localhost) no browser (Rota de raíz ou pré-definida).

Para isso vamos novamente abrir o ficheiro routes.rb, que está na pasta config.

Vamos alterar o comportamento da rota de raiz (root) que, por convenção, aponta para o ficheiro "index.htm" que está na pasta public, e que vimos quando abrimos a aplicação pela primeira vez.

Para que tal aconteça vamos, primeiro, apagar ou renomear o ficheiro index.html, que está dentro da pasta public, pois tem precedência sobre a rota.

De seguida vamos modificar ligeiramente o ficheiro de rotas, passando para dentro do bloco de código uma instância do objecto de gestão de rotas, e vamos adicionar o mapeamento da rota "root":

```
ListaDeCompras::Application.routes.draw do |map|
  match ':controller(/:action(/:id))'
  map.root :controller=>'inicio', :action =>'ola'
end
```

Agora acedendo simplesmente a http://localhost, a Action "olá" do Controller "inicio" deverá ser invocada.

Vamos agora analisar o código fonte que foi gerado para a página:

```
<!DOCTYPE html>
<html>
<head>
    <title>ListaCompras</title>
    <link href="/stylesheets/scaffold.css?1310683055" media="screen" rel="stylesheet" type="text/css" />
    <script src="/javascripts/prototype.js?1310683025" type="text/javascript"></script>
    <script src="/javascripts/effects.js?1310683025" type="text/javascript"></script>
    <script src="/javascripts/dragdrop.js?1310683025" type="text/javascript"></script>
    <script src="/javascripts/controls.js?1310683025" type="text/javascript"></script>
    <script src="/javascripts/rails.js?1310683025" type="text/javascript"></script>
    <script src="/javascripts/application.js?1310683025" type="text/javascript"></script>
    <meta name="csrf-param" content="authenticity_token"/>
    <meta name="csrf-token" content="Wwl+qgnbSIXvi2eRmDY72z1oWAkbzPuKFKIy/tg5W4g="/>
</head>
<body>
    Este texto foi escrito pela action ola do controller inicio
</body>
</html>
```

É possível ver que existe bem mais conteúdo do que aquele que a nossa View contém. Isso deve-se à existência de um "Layout" ou "Master Page". Um "Layout" é uma página principal que fica "à volta" das Views, englobando-a. É no layout que devem ser definidos os pontos comuns do website, de modo a que fiquem definidos num só local, evitando a repetição (princípio DRY), como menus, barras de topo/fundo, etc.

Onde está então este layout?

Este layout encontra-se dentro da pasta layouts, que se encontra dentro da pasta Views. Um layout tem a extensão ".html.erb". Vamos abrir o ficheiro application.htm.erb no editor de texto:

```
<!DOCTYPE html>
<html>
<head>
  <title>ListaCompras</title>
  <%= stylesheet_link_tag :all %>
  <%= javascript_include_tag :defaults %>
  <%= csrf_meta_tag %>
</head>
<body>
   <%= yield %>
</body>
</html>
```

Antes de analisarmos o código, vamos adicionar um texto antes da instrução yield, para que melhor se perceba como funciona a interligação da View com o Layout.

Para dar algum ênfase, vamos utilizar um H3:

```
<body>
<h3>Este texto encontra-se no layout</h3>
<%= yield %>
</body>
```

Como é possível ver, tanto o texto que colocámos no layout como o que está contido na View são mostrados.

Passemos então à análise do código do template (ficheiro layout.htm.rb). Podemos ver algum código HTML (o mesmo que havíamos visto no código fonte da página que abrimos à pouco no browser) e algum código Ruby (delimitado pelas tags <% %>).

Existem duas formas de incluír Ruby dentro de uma página, seja ela um layout ou uma view:

- A forma com output <%= instrução %>
- A forma sem output <% instrução %>

A primeira é utilizada para escrever algo na página, e a segunda apenas para instruções sem geração de output. Vamos então analisar o código Ruby presente nesta página:

<%= stylesheet_link_tag :all %> - É uma tag Ruby com geração de output (vê-se pelo sinal de igual [=]). Recebe como parâmetro o símbolo :all. Este método "olha" para a pasta stylesheets (dentro da pasta public) e gera código HTML para que as folhas de estilo sejam adicionadas à página. É possível definir um ficheiro isolado, mas neste caso com o símbolo :all, todas as folhas de estilo contidas na pasta serão carregadas.

<%= javascript_include_tag :defaults %> - É um método em tudo semelhante ao anterior, com a diferença de que este não está relacionado com folhas de estilos (CSS) mas sim com ficheiros de script client-side (Javascript) contidos na pasta scripts, dentro da pasta public. O símbolo :defaults passado como parâmetro visa carregar alguns ficheiros de javascript, pertencentes à Biblioteca Script.acol.us (http://script.aculo.us/) que vêm junto com a aplicação. Existe quem defenda que esta instrução não deveria ser automaticamente adicionada a uma aplicação recém criada, pois implica o carregamento de cerca de 140KB de ficheiros Javascript, e como ficheiros de javascript são carregados em série, leva a um atraso que pode ser significativo e muitas vezes não se utiliza qualquer javascript.

<%= csrf_meta_tag %> - CSRF significa "Cross Site Request Forgery". Uma vulnerabilidade do Rails (e de outras tecnologias web) é a capacidade de se gerar um request de um website para outro, levando a que por vezes sejam servidos pedidos ilegítimos.

Esta instrução insere, como é possível ver no código gerado, um token de autenticidade sob a forma de meta tag. Sem este token acções como a própria submissão de um formulário poderá não funcionar, por falta de autenticidade do request. Desta forma é possível verificar se o request foi gerado dentro do mesmo domínio ou não.

Existem formas explícitas de "abrir" o request para outros domínios específicos.

Mais informação:

http://guides.rubyonrails.org/security.html#cross-site-request-forgery-csrf

Por último temos a instrução **<%=yield %>**. É esta instrução que "será substituída" pelo conteúdo da View. Como vimos anteriormente a View aparece com o layout "à sua volta".

É, porém, necessário, indicar qual o local, dentro do layout, onde a view será mostrada.

É possível ter mais do que uma instrução <%= yield %> no mesmo template, sendo que desta forma, a partir do segundo yield (inclusive), estes terão de ter um nome, para poderem ser referenciados de forma distinta.

Exemplo de um layout com dois yields:

```
.... html …
<%= yield 'barra_topo' %>
.... html ....
<%= yield %>
```

Para que a sidebar seja "preenchida" no layout, terá de ser usado método "content_for", seguido do nome do yield a que se refere.

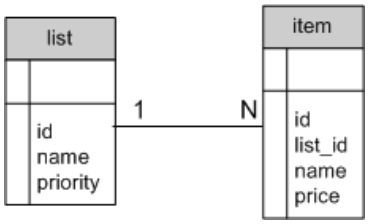
Posso ter vários layouts?

Sim. Imagine-se o cenário típico de um website com front-office e back-office. É possível ter dois (ou mais) layouts e depois, ao nível do controller ou ao nível da própria action definir qual ou template a ser usado, ou então dizer que não

se pretende que seja utilizado um template.

Criação de Modelos

Voltando ao objectivo de criar uma aplicação de lista de compras, vamos necessitar de criar o modelo de dados. Imagine -se o seguinte modelo de dados:



Temos que uma lista de compras tem um identificador (id), um nome e uma prioridade, e poderá ter vários itens. Cada item estará ligado a uma lista e terá um campo chave (id), o id da lista a que pertence, um nome e um preço.

Este modelo de dados segue as *naming conventions* de Rails: Utilizará nomes de tabelas no plural, o campo chave em ambas as tabelas chama-se "id". A chave estrangeira da list na tabela de itens chama-se "list_id": singular da tabela referenciada, seguido de "_id". Desta forma, deixamos as convenções fazerem o seu papel, e não é necessário configurar quais as chaves primárias, qual o nome das tabelas, qual o nome das chaves, etc.

É altura de criar o modelo de dados na nossa aplicação. A sintaxe é semelhante à criação de controllers, mas agora existem mais informações que necessitam de ser passadas no comando: os campos do modelo e os seus tipos de dados. Caso existisse já uma base de dados criada, os campos poderiam ser inferidos a partir da base de dados. No entanto vamos utilizar o paradigma "Code First" e vamos deixar que seja o Rails a criar a base de dados por nós.

A sintaxe para geração de uma entidade de model é:

rails generate Model nome_do_modelo [parametros]

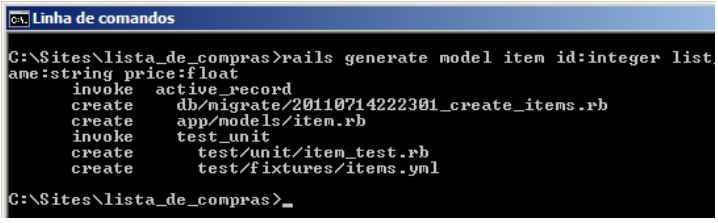
Os atributos são passados a seguir ao nome do Model no seguinte formato:

nome_do_campo:tipo_de_dados

A passagem de parâmetros é facultativa. Se não forem passados parâmetros, os atributos da classe gerada serão inferidos a partir da tabela da base de dados a que a entidade se referencia. Iremos ver como configurar a ligação à base de dados de seguida.

Não vamos para já gerar o modelo para a lista de compras, pois irá ser utilizada outra técnica (scaffolding) posteriormente.

Vamos então criar o model para o "item":



Ao observarmos o output do comando que acabámos de invocar, vemos que foram criados 4 ficheiros.

Os primeiros dois, após invocação do "active_record" (abordado um pouco mais à frente) são um ficheiro de migração, responsável pela criação da tabela na base de dados e o segundo é o ficheiro que contém o Model em si: a classe de Ruby que representa um item da lista de compras.

Os outros dois ficheiros, criados após invocação do "unit_test", dizem respeito aos testes unitários para a entidade "item". É aqui que serão especificados os testes unitários (não serão abordados neste artigo). Interessa notar quem um dos ficheiros é um ficheiro Ruby (extensão .rb) e o outro um ficheiro YAML (extensão .yml). Este tipo de ficheiros será abordado aquando da configuração da base de dados.~

Scaffolding

O conceito de Scaffold (esqueleto), neste contexto, significa a construção de um back-end, dentro de uma aplicação informática, a partir dos dados e meta-dados contidos na base de dados.

A partir da análise da estrutura da base de dados, gera código e interface para as operações CRUD (Create, Read, Update, Delete).

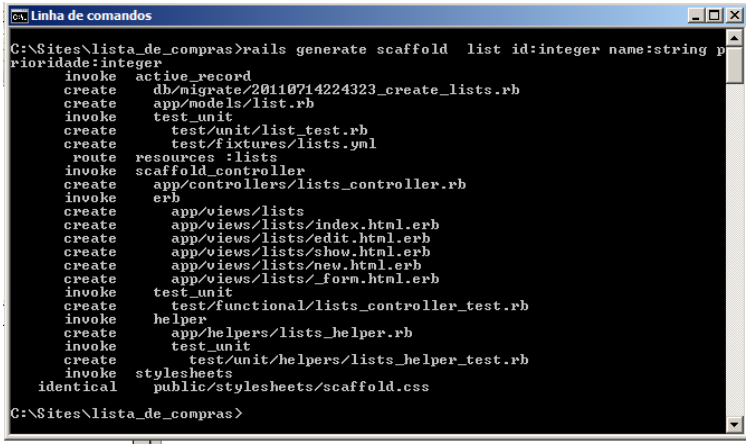
Em Rails, o scaffolding apresenta-se como uma mais-valia da tecnologia. Se por um lado o resultado nem sempre é o esperado, e em muitas das vezes existem vantagens em fazer manualmente a interface e código para as operações

de CRUD, outras vezes não existem grandes validações nem outras regras de negócio por trás, o que leva a que o scaffolding seja uma forma rápida de produzir resultados.

Vamos utilizar scaffolding para gerar uma interface para gerir as listas de compras. A gestão dos itens de cada lista será feita do modo "tradicional", ou seja, criando tudo à mão.

A sintaxe da criação de um scaffold é semelhante à da criação do model. A criação de um scaffold agrega a criação de uma entidade do Model, e a criação de um controller e das actions e views necessárias para que tudo funcione (daí não ter sido criado o Model "list" anteriormente).

Vamos então criar o scaffold para a entidade "list":



É importante observar, no output do comando, que o scaffold não faz nada por si só. O que faz é invocar o active_record, o scaffold_controller e o gerador de stylesheets para que o todo o scaffold possa ser criado.

Note-se, ainda, que foi criado um controller chamado "lists" e várias Views para este controller (index, edit, show, new e _form). Iremos voltar ao controller e às views mais à frente.

A partir da versão 2.0 do rails, deixou de ser possível gerar um scaffold para uma entidade já existente, sendo a entidade criada ao mesmo tempo que o scaffolding. É por isso necessário pensar bem se uma determinada entidade vai ser usada com scaffolding ou não.

O Rails encarrega-se de adicionar um "recurso" às rotas, de modo a que as ligações usadas pelo scaffold para navegação, funcionem:

map.resources :lists

Configuração da Base de Dados e Active Record

Falta apenas tratarmos da relação entre estes dois elementos do Model para que o modelo de dados esteja definido por completo.

Para isso é necessário dizer que uma lista pode ter vários itens, e que um item pertence a uma lista. Parece simples dizer isto desta forma, e de facto é simples colocar esta lógica no código.

Abrindo o ficheiro **list.rb** que se encontra dentro da pasta model, vamos então indicar que uma lista pode ter vários itens:

```
class List < ActiveRecord::Base

      has_many :itens

end
```

Como é possível ver, esta não é uma classe normal. É uma classe que deriva (herda) da classe Active Record - Base. Como tal contém alguns métodos especiais, como é o caso deste "has_many". Este método recebe um símbolo como parâmetro, e o símbolo (no plural neste caso) indica qual a entidade presente na relação.

Da mesma forma, no ficheiro **item.rb** vamos dizer que um item pertence a uma lista:

```
class Item < ActiveRecord::Base

      belongs_to :list

end
```

O método "belongs_to" indica que um item pertence a uma lista (repare-se que o símbolo neste caso está no singular).

Tendo o modelo de dados definido, é altura de fazer reflectir estas alterações na base de dados. Mas qual base de dados? Existe um ficheiro chamado database.yml dentro da pasta config. Este ficheiro utiliza um formato chamado YAML (YAML ain't markup language).

É um formato de serialização como o XML e JSON que no caso do Rails é muito utilizado para configurações. Basicamente, em YAML, um elemento é definido pelo seu nome seguido de dois pontos (:).

Depois os campos/atributos deste elemento são precedidos por um espaçamento (tab) e o valor de cada campo é precedido de dois pontos (:).

Por exemplo, um animal poderia serializar-se da seguinte forma:

animal:
especie: Cao
raca: Terrier
nome: Pintas

Voltando ao ficheiro database.yml, podemos ver que as configurações da base de dados se encontram divididas por vários ambientes:

```
# SQLite version 3.x
#   gem install sqlite3
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000
production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000
```

O ambiente predefinido, e que nos interessa neste caso, é o ambiente de desenvolvimento. É possível mais tarde alterar o ambiente de funcionamento (para produção, por exemplo). Podemos ver que está a ser utilizado o motor de base de dados "sqlite3", e que a base de dados irá residir dentro da pasta "db" e que se irá chamar "development.sqlite3". Está também definido o número máximo de ligações e o tempo máximo de espera.

Caso o motor de base de dados fosse diferente (por exemplo MySQL) os campos da parametrização seriam outros (host, username, password, ...). A separação por ambientes permite-nos facilmente mudar de ambiente, e fazer testes na base de dados de desenvolvimento, evitando fazer testes em base de dados de produção. Dado que não vamos, desta vez, alterar a configuração da base de dados, vamos fechar o ficheiro e vamos "migrar" o nosso modelo de dados para a base de dados.

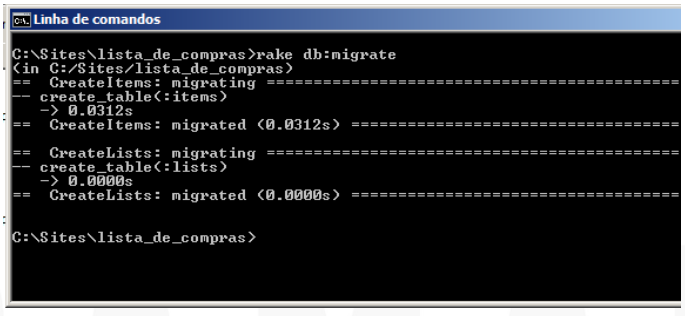
Antes de procedermos à migração, vamos ver os ficheiros de migração que foram gerados. Estes ficheiros (um por cada entidade) deverão estar na pasta db/migrate. Vamos abrir o ficheiro de migração para a entidade Item.

```
class CreateItens < ActiveRecord::Migration
  def self.up
    create_table :itens do |t|
      t.integer :id
      t.integer :list_id
      t.string :name
      t.float :price
      t.timestamps
    end
  end
  def self.down
    drop_table :itens
  end
end
```

É possível ver que o script de migração pertence ao Active Record - Migration, e que tem dois métodos: o up e o down. O método up serve para criar a tabela na base de dados. É possível identificar os campos do item, que serão traduzidos em colunas da tabela. O método down serve para reverter a migração, o que neste caso significa a destruição da tabela.

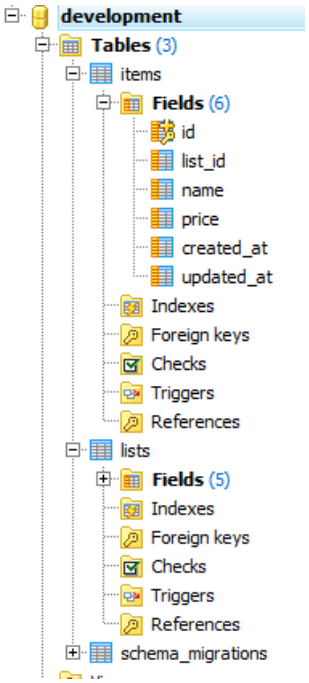
Vamos então proceder à migração. Para isso recorremos ao comando

rake db:migrate



Como é possível ver pelo output, tal como esperado, foram criadas duas tabelas.

Abrindo o ficheiro da base de dados de desenvolvimento num qualquer utilitário que trabalhe com SQLite (neste caso o SQLite Maestro) é possível ver que as tabelas e respectivas colunas foram criadas.



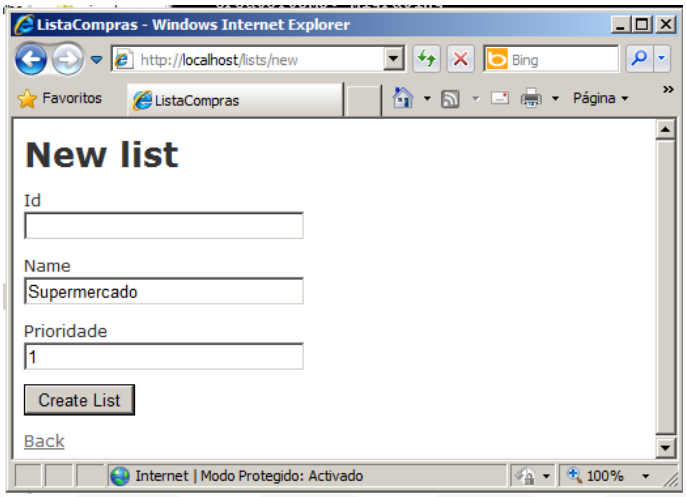
É de notar que existe uma tabela que não criámos, e que contém dados relacionados com a migração do modelo de dados.

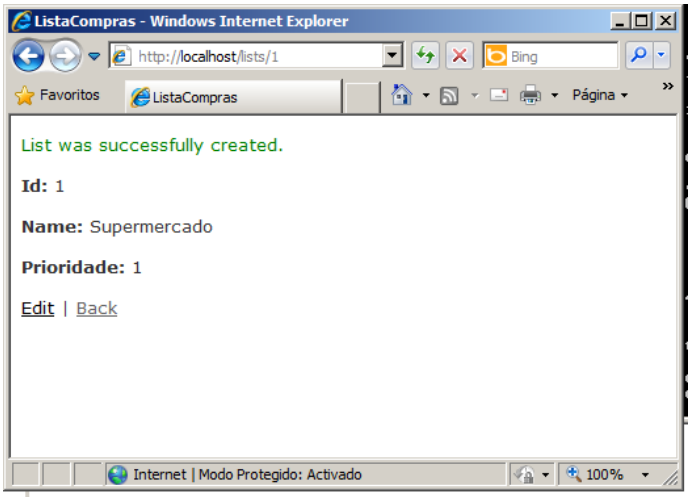
É também importante referir que foram adicionados dois campos a cada tabela: updated_at e created_at. Estes são dois campos do tipo datetime e "automagicamente" (graças ao active_record) guardam a data/hora de criação ou alteração do objecto.

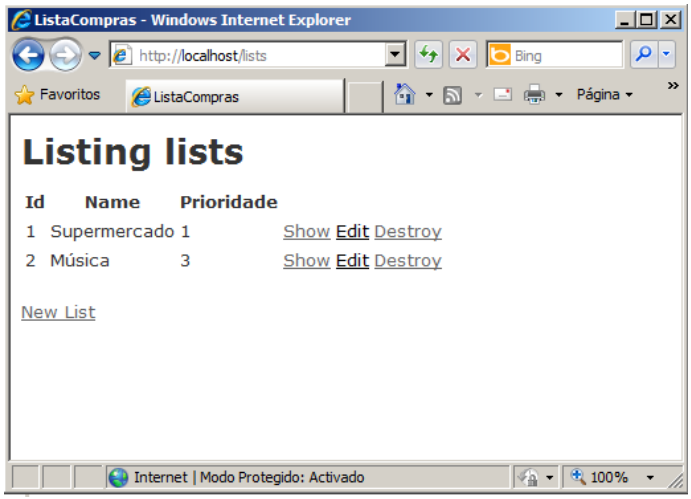
Estando o modelo de dados reflectido na base de dados, podemos iniciar o servidor e tentar abrir o scaffold da lista, através do controller "lista" que foi criado.

Vamos aceder ao seguinte url: http://localhost/lists, que nos dá acesso a uma tabela com as listas já existentes (inicialmente vazia) e uma ligação para o formulário que permite criar novas listas.

Nota: quando apenas o controller é passado no URL, a action chamada é a "index"







Todo o código gerado está disponível para consulta e alteração, tanto ao nível do controller como ao nível das Views, e é uma boa forma de perceber como o Rails funciona.

Vamos agora fazer o mesmo processo para os itens, mas sem usar scaffolding. Por questões de brevidade irão apenas ser demonstrados a criação e listagem de itens.

O primeiro passo será gerar o controller para os itens. Este controller irá conter as actions (e Views) para listar, criar, alterar e apagar itens de uma determinada lista.

generate controller itens

Vamos adicionar uma action chamada "new". Esta action e a respectiva view terão como responsabilidade permitir a criação de um novo item. É também necessário criar o ficheiro da View, dado que não é criado automaticamente.

Uma variável/objecto pode ser passado do controller para a

view, simplesmente colocando uma arroba antes do nome da variável.

Controller (ficheiro itens_controller.rb):

```
def index
      @title="Lista de Artigos"
      @itens=Item.find(:all,:order=>'list_id')
end
```

Nesta action é definida uma variável com o título. Esta é uma variável global, pois começa com uma arroba, e desta forma poderá ser acedida dentro da view.

É também definida uma variável global chamada lists. A classe "List", estende a classe "ActiveRecord", pelo que contém uma série de métodos relacionados com o acesso a dados.

O ActiveRecord é uma ferramenta ORM, e como tal permite que o developer se concentre no desenvolvimento, abstraindo de uma forma completa a base de dados. Não é sequer necessário escrever SQL, dado que através dos métodos disponíveis, o ActiveRecord encarrega-se de gerar o SQL necessário.

Um desses métodos é o "find". Este método recebe como parâmetros elementos como condições de pesquisa, o número de elementos e o offset (para paginação), e devolve um Array com elementos, trazidos directamente da base de dados.

Vamos ver mais alguns métodos do ActiveRecord mais à frente, à medida que vai sendo necessário utilizá-los.

Abrindo a janela da consola, onde o servidor está a ser executado, é possível ver o código SQL gerado.

View (ficheiro views/itens/new.rhtml):

```
<h3><%= @title %></h3>
<%= form_for :item, @item, :url =>
{ :action => "update" ,:id=>@item.id} do | f | %>

Id:</br>
  <%= f.text_field :id,:disabled => true %> <br/>
  Nome:<br/>
  <%= f.text_field :name %> <br/>
  Preço:<br/>
  <%= f.text_field :price %> <br/>
  Lista:<br />
  <%= f.select (:list_id, @lists.collect
 { | lista | [lista.name, lista.id]}) %> <br/>
  <%= submit_tag "Guardar" %>
<% end %>

<a href="/itens/">Voltar</a>
```

Passemos então à análise do código da View para a construção do formulário. Como é possível ver, para aceder à variável com o título, basta invocá-la, sem esquecer da arroba antes do nome da variável.

O Rails proporciona também Helpers para a construção do formulário. Ainda que este possa ser construído com HTML, a utilização dos Helpers ajuda a reduzir o código e a torná-lo mais fácil de ler.

A instrução form_tag, recebe a acção, que neste caso está direccionada para a Action "Create". Quando se omite o Controller, assume-se que a acção pertence ao controller actual.

Esta instrução termina com o end, no final da página e considera-se que tudo o que estiver entre estas duas instruções está dentro do formulário. Basicamente traduzem-se nas tags <form ..> e </form> do HTML, respectivamente. Os Helpers não se limitam à criação do formulário, mas também na criação dos campos deste.

A sintaxe do Helper para criação de inputs de texto é a seguinte:

text_field :nome_objecto, :nome_do_campo

Por exemplo, para criar uma caixa de texto, para o campo "name" de um objecto chamado "item" utiliza-se a instrução

text_field :name, :item

A criação de um "Select" (ou uma caixa de selecção) também beneficia de um Helper. Na action que invoca esta View, criámos uma variável global chamada @lists, onde passámos, através do método find do ActiveRecord, todas as listas da base de dados.

Em que formato vêm estas listas? Existe uma forma de sa-

ber qual o conteúdo de uma variável, e qual o seu formato. Vamos, por momentos, comentar todo o código da View e adicionar uma só linha:

```
<%= debug(@lists) %>
```

Esta instrução serializa (em formato YAML) o objecto ou a colecção de objectos. Neste caso, existem duas listas na base de dados e o resultado é este:

```
---
- !ruby/object:List
  attributes:
    name: Supermercado
    created_at: 2011-07-14 22:46:05.199146
    updated_at: 2011-07-14 22:46:05.199146
    id: 1
    prioridade: 1
  attributes_cache: {}

  changed_attributes: {}

  destroyed: false
  marked_for_destruction: false
  new_record: false
  previously_changed: {}

  readonly: false
- !ruby/object:List
  attributes:
    name: "M\xC3\xBAsica"
    created_at: 2011-07-14 22:46:34.065217
    updated_at: 2011-07-14 22:46:34.065217
    id: 2
    prioridade: 3
  attributes_cache: {}

  changed_attributes: {}

  destroyed: false
  marked_for_destruction: false
  new_record: false
  previously_changed: {}


  readonly: false
```

Este é o resultado da serialização, com o método debug. Este método torna-se especialmente útil quando algo corre mal e não se sabe muito bem de onde vem o problema.

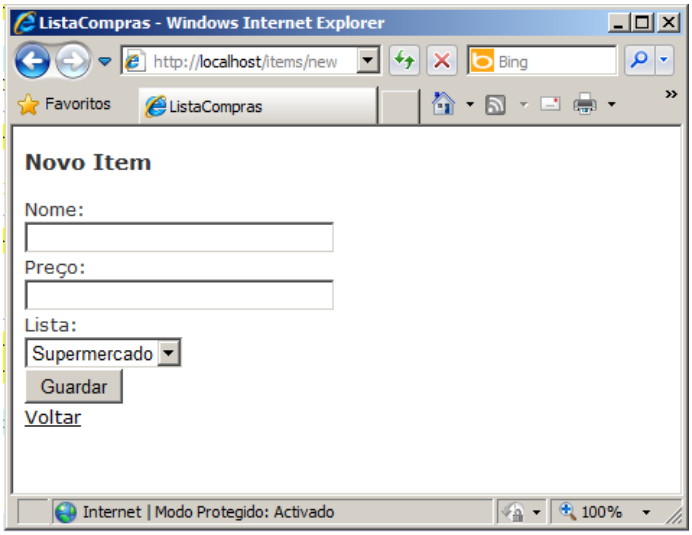
Interessa reter que cada elemento da lista tem um campo id, e um campo name, que serão utilizados na caixa de selecção. Podemos agora retirar a instrução de debug e descomentar o restante código.

Voltando à caixa de selecção do formulário, a instrução select recebe e o nome do objecto (:item) e o nome do campo (:list_id). De seguida recebe a colecção dos itens que vão compor a listagem. O método collect, vai pegar em cada item da colecção, temporariamente identificado por "lista" e vai construir um conjunto de dois valores: o nome e o id.

Esta lista de elementos com estes dois valores vai povoar a caixa de selecção.

Por último existe um Helper para o botão de Submit – submit_tag que recebe o valor a apresentar no botão.

O resultado desta view é o seguinte:



Vamos analisar o código HTML gerado pelos helpers.

```
<form accept-charset="UTF-8" action="/itens/create"
method="post">

<div style="margin:0;padding:0;display:inline">

<input name="utf8" type="hidden" value="&#x2713;"/>

<input name="authenticity_token" type="hidden"
value="Ww1+qgnbSIXvi2eRmDY72z1oWAkbzPuKFKIy/
tg5W4g="/>

</div>

      Nome:<br/>

      <input id="item_name" name="item[name]" si-
ze="30" type="text" /> <br/>

      Preço:<br/>

      <input id="item_price" name="item[price]"
size="30" type="text" /> <br/>
```

```
      Lista:<br />

      <select id="item_list_id" name="item
[list_id]">

<option value="1">Supermercado</option>

<option value="2">Música</option>

</select> <br/>

      <input name="commit" type="submit"
value="Guardar" />

</form>
```

É possível encontrar imediatamente a tags <form ... > com o método "post" e a action especificada no helper e o controller continua o "itens". No final do formulário está a tag </form> a terminá-lo.

É também possível ver que foi criado um campo escondido chamado "autenticity_token" que serve para que a autenticidade do request possa ser comprovada pelo Rails. Este token é comparado com o token que figura na secção <HEAD> da página e que falámos anteriormente (CSRF).

É ainda possível ver que o nome de todos os campos segue um padrão: nome_do_objecto[nome_do_campo]. Este padrão tem uma razão de ser, pois facilita muito a criação do objecto Ruby a partir dos valores do formulário, como veremos de seguida.

Passemos então à criação da Action responsável pela recolha dos dados passados pelo formulário e pelo registo do "item" na base de dados. Falamos da Action "Create" do Controller "Itens".

```
def create
      @item = Item.new(params[:item])
      @item.save
      redirect_to(:action=>'index')
end
```

Estas 3 linhas de código são suficientes para "pegar" nos dados do formulário e guardar o objecto na base de dados. De facto, as primeiras duas linhas bastam. A terceira linha apenas redirecciona o browser para View "Index", que irá, dentro de alguns momentos, conter a listagem de todos os itens, presentes em todas as listas de compras.

A instrução presente primeira linha do método obtêm a partir dos valores passados pelo formulário (neste caso via POST) o objecto "item". Daí aquela notação do tipo "item[name]" no nome dos campos do formulário.

A segunda linha chama o método save (herdado do ActiveRecord) e o objecto é automaticamente persistido na base de dados.

Não vamos testar ainda, pois falta o local onde ver os itens. Vamos passar à criação da Action "index", que irá trazer todos os itens da base de dados e mostrá-los numa tabela.

Action "index" do controller Itens (ficheiro itens_controller.rb):

```
def índex
      @title="Lista de Artigos"
      @itens=Item.find(:all,:order=>'list_id')
end
```

Da mesma forma que anteriormente, utiliza-se o ActiveRecord para trazer todos os itens, mas desta vez, ordenados pelo campo "list_id".

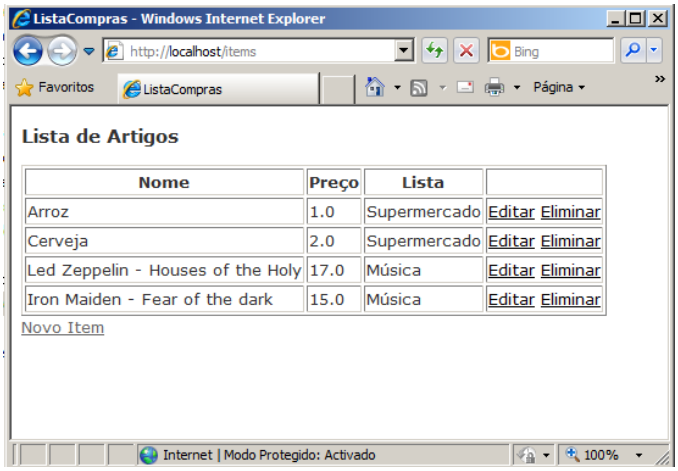
View Index (ficheiro itens/index.rhtml):

```
<h3><%= @title %></h3>
<table border="1">
<tr>
      <th>Nome</th>
      <th>Preço</th>
      <th>Lista</th>
      <th></th>
</tr>
<% @itens.each do |item| %>
<tr>
   <td>
      <%= item.name%>
   </td>
   <td>
      <%= item.price%>
   </td>
   <td>
      <%= item.list.name%>
   </td>
   <td>
   <a href="/itens/edit/<%=item.id%>">Editar</a>
<a href="/itens/destroy/<%=item.id%>">Eliminar</a>
  </td>
</tr>
<%end%>
</table>

<a href="/itens/new">Novo Item </a>
```

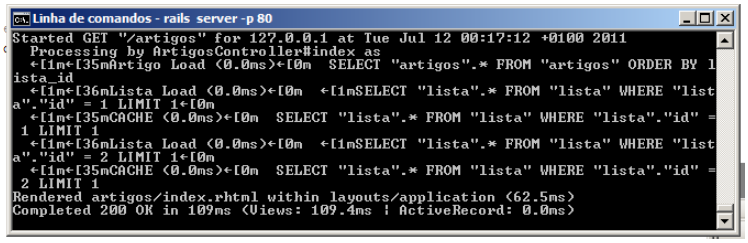
Basicamente iteramos sobre a colecção, mostrando os dados sob a forma de uma linha de tabela. Note-se que acedemos ao atributo "list" ele é trazido da base de dados, como objecto, neste caso, mostrando o campo "nome". O carregamento de objectos relacionados é feito com LazyLoad, ou seja, só em caso de o objecto ser requerido é que é trazido da base de dados, caso contrário, a relação não é feita. No final do ficheiro, está uma ligação para o formulário que permite a criação de mais itens.

Nota: Estão também a ser criados dois links para a alteração e destruição dos itens. Apesar de estes duas componentes da aplicação, por questões de brevidade, não serem abordadas no artigo, estão contidas no código da aplicação, cujo link para download está no final do artigo.

Vamos então ver o resultado:



É possível ver na consola o SQL gerado pelo ActiveRecord:



Repare-se que é feita uma query inicial à tabela de itens, e depois são feitas várias queries à tabela de listas. Isto acontece porque estamos, para cada item, a aceder à lista à qual ele pertence, e essa operação gera uma query.

As restantes Views e Actions seguem um raciocínio semelhante. O código fonte da aplicação completa, encontra-se para download (ver parte final do artigo).

Conclusão

Por agora é tudo. Apesar de ter sido um exemplo simples, espero ter demonstrado como o Ruby on Rails funciona e, pelo menos, ter despertado o vosso interesse para esta tecnologia.

O que foi abordado aqui foi apenas uma muito pequena parte desta vasta e rica tecnologia de desenvolvimento web.

Links Úteis

Para quem quiser aprofundar mais, deixo um conjunto de links que serão certamente muito interessantes.

Código Fonte da aplicação de demonstração:
https://github.com/CarlosRodrigues/DemoRailsRevistaProgramar

Webcasts sobre Rails: http://railscasts.com/
Site Oficial Ruby (PT): http://www.ruby-lang.org/pt/
Site Oficial Rails: http://rubyonrails.org/
Documentação Ruby: http://www.ruby-doc.org/
Documentação API Rails: http://api.rubyonrails.org/
Comunidade Ruby-pt: http://www.ruby-pt.org/
Alojamento Rails (gratuito no pacote mais básico): http://www.heroku.com/
Livro: Why's (poignant) guide to Ruby: http://www.rubyinside.com/media/poignant-guide.pdf

AUTOR

Escrito por Carlos Rodrigues
Licenciado em Engenharia Informática pelo Instituto Politécnico de Viseu, frequenta actualmente o Mestrado em Sistemas e Tecnologias de Informação para Organizações na Escola Superior de Tecnologia de Viseu. É actualmente consultor / programador na área da banca, por uma multinacional Portuguesa na região de Lisboa e formador em regime de freelance. Interessa-se pelo desenvolvimento para web e para dispositivos móveis.

Twitter: @crodriguesPT Blog: http://crodrigues.com

A PROGRAMAR

Lua – Linguagem de Programação (Parte 10)

Criar um sistema RSS no Sharepoint através de uma lista de páginas

Introdução ao Objective-C e à plataforma IOS

Atributos em C#

Lua – Linguagem de Programação – Parte 10

Neste artigo são apresentadas informações introdutórias e básicas sobre o uso da linguagem Lua embutida dentro de programas escritos em C e C++. Além deste tema apresenta-se também informações relacionadas ao uso de co-rotinas.

Para a execução dos programas codificados nas linguagens C e C++ integrados com a linguagem Lua deve-se levar em consideração o uso do compilador GCC.

Linguagem de Script

Em 12 de Maio de 2011 foi publicado na revista *ACM Queue Architecting Tomorrow's Computing* o artigo intitulado **Passing a Language through the Eye of a Needle (Passando uma Linguagem através do Buraco de uma Agulha)**, onde Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes arquitectos da linguagem Lua falam sobre o uso da linguagem Lua com outras linguagens.

Em http://www.lua.org/portugues.html, se lê que "*Lua é uma engine rápida e pequena que você pode facilmente embutir na sua aplicação. Lua tem uma API simples e bem documentada que permite uma integração forte com código escrito em outras linguagens. É simples estender Lua com bibliotecas escritas em outras linguagens. Também é simples estender programas escritos em outras linguagens com Lua. Lua é usada para estender programas escritos não só em C e C++, mas também em Java, C#, Smalltalk, Fortran, Ada, Erlang, e mesmo outras linguagens de script, como Perl and Ruby*".

Dentro da temática desta discussão que este artigo mostra como fazer a integração de código em linguagem Lua (linguagem de *script*) de forma embutida em códigos previamente escritos em C e C++.

A linguagem de programação Lua possui duas importantes características, pode ser usada como linguagem de extensão e como linguagem extensível, ou seja, o fluxo de comunicação com a linguagem Lua é bidireccional. Como linguagem de extensão, Lua pode ser usada juntamente com outra linguagem no sentido de estender alguma funcionalidade embutida para esta outra linguagem.

Como linguagem extensível, Lua pode ser usada para obter uso de recursos existentes em outra linguagem, a fim de se completar. A comunicação entre as linguagens C, C++ e Lua pode ocorrer sob duas esferas: na primeira esfera por meio da definição de uma instância de estado de operação e na segunda esfera por meio do uso do conceito de "pilha". O foco de estudo deste artigo concentra-se na primeira esfera.

Lua Embutida

Lua foi concebida para comunicar-se com a linguagem C, ou seja, foi preparada para ser uma biblioteca da linguagem C. No entanto, permite ser usada como linguagem independente como apresentado nas nove partes anteriores desta série de artigos.

Usar a integração de Lua com C++ foi uma ocorrência natural, devido à origem de C++ a partir de C. No caso de uso das linguagens C e C++, o uso de Lua é semelhante entre elas. Assim sendo, é necessário fazer uso dos arquivos de cabeçalho **lua.h**, **lauxlib.h** e **lualib.h**; integrantes da biblioteca Lua, de-nominadas **API C**, a qual é escrita em ANSI C. As funções do arquivo **lua.h** são iniciadas pelo prefixo **lua_**, as funções do arquivo auxiliar **lauxlib.h** são iniciadas pelo prefixo **_luaL** e o arquivo **lualib.h** dá acesso às bibliotecas padrão da linguagem Lua.

A identificação do uso das bibliotecas de **API C** em linguagem C é utilizada com a chamada dos arquivos de cabeçalho que são configuradas dentro da função **main** a partir da definição em negrito indicado a seguir:

```
#include <stdio.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
int main(void)
{
  // código com funções Lua
 return 0;
}
```

A identificação do uso das bibliotecas de **API C** em linguagem C++ é utilizada com a definição do bloco externo com a cláusula **extern** a partir da definição em negrito indicado a seguir:

```
#include <iostream>
extern "C"
{
  #include <lua.h>
  #include <lualib.h>
  #include <lauxlib.h>
}
using namespace std;
int main(void)
{
  // código com funções Lua
  return 0;
}
```

Lua – Linguagem de Programação – Parte 10

A partir da definição da chamada das bibliotecas de **API C** basta indicar junto à função **main()** tanto de um programa escrito em linguagem C como em C++ a definição de uso das funções Lua que farão as acções de processamento de acesso a recursos Lua de dentro do programa escrito em C ou C++.

A primeira definição a ser configurada na função **main()** é a indicação de uma variável ponteiro que dará acesso à instância da operação Lua, representado pelo ponteiro do tipo **lua_State** *. Após a definição da instância de estado para um ponteiro do tipo **lua_State** * torna-se necessário criar pelo menos um estado de uso para este ponteiro com a função **lua_open()**.

O valor retornado pela função **lua_open()** o ponteiro **lua_State** * necessita de ser passado como argumento (parâmetro) às demais funções da **API C** que venha a ser utilizado dentro do código do programa. Todo o estado da operação Lua definido dentro da função **main()** necessita de ser encerrado com o uso da função **lua_close()**.

Assim sendo, note o trecho em negrito da função **main()** seguinte:

```
int main(void)
{
  lua_State *ACESSO = lua_open();
  // acesso as funções Lua
  lua_close(ACESSO);
  return 0;
}
```

Note que a instância de acesso ao estado Lua é definido pelo ponteiro (variável de estado) ACESSO, inicializada com a função **lua_open()** e finalizada com a função **lua_close()** antes do uso da instrução **return 0;**.

```
// Exemplo 1
#include <stdio.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>

int main(void)
{
  lua_State *ACESSO = lua_open();
  luaL_openlibs(ACESSO);
  luaL_dostring(ACESSO, "print('Ola.')");
  lua_close(ACESSO);
  return 0;
}
```

O programa seguinte apresenta um programa escrito em linguagem C, o qual executa uma instrução em linguagem Lua. O mesmo bloco de código **main()** poderia estar sendo definido dentro de um programa C++.

Grave o programa com o nome **exemplo1.c**. Note que o programa faz uso das funções Lua: **luaL_oponlibs()** e **luaL_dostring()**.

A função **luaL_oponlibs()** estabelece para a variável de estado ACESSO o acesso as bibliotecas Lua que correspondem a:

"luaopen_base();";
"luaopen_table();";
"luaopen_io();"
"luaopen_string();";
"luaopen_math();".

A função **luaL_dostring()** atribui à variável de estado ACESSO a capacidade de efectuar a apresentação da *string* escrita em código Lua. Ao ser executado o programa será apresentada no ecrã a *string* "**Olá.**".

Note que com este pequeno exemplo se teve acesso à execução de recursos Lua a partir de um programa escrito em linguagem C por meio de funções para acesso à API Lua. Faça a compilação do código e execute-o.

O compilador GCC é uma ferramenta de trabalho similar à ferramenta CC do UNIX, a qual opera em vários sistemas operativos. Possui como forma de uso, uma certa diversidade na sua escrita, tal como pode ser observado junto a vários sistemas operativos:

S.O.	Comando
UNIX	cc –o exec exemplo1.c
Linux	gcc –o exemplo1.c exec
Linux	g++ -o exemplo1.c exec
Linux	gcc exemplo1.c –o exec
Linux	g++ exemplo1.c –o exec
Windows	gpp exemplo1.c -o exec

No entanto, é comum encontrar certas variações no uso da sintaxe de escrita do comando de compilação para o GCC. Há situações onde é necessário acrescentar junto à linha de compilação as *flags* **-llua –llualib** ou as *flags* **–llua5.1 –llualib5.1**, ou uma variação entre elas.Há situações onde é necessário indicar os locais onde se encontram as bibliotecas o que implica o uso de um complemento da linha de compilação com as opções **–I** indicando o uso do directório **include** e **–L** indicando o uso do directório **lib** do local onde se encontra a instalação da linguagem Lua.

Dependendo do sistema operativo em uso e da forma de instalação da linguagem Lua é necessário acrescentar junto a linha de compilação a opção **-Wall 'lua-config --include -libs'**.

Apesar das informações aqui expostas pode ocorrer a não execução de um código embutido em Lua que foge ao propósito aqui apresentado. Isto ocorre devido a factores diversos que não podem neste artigo serem tratados, sendo tema e tese de discussão encontrados nos diversos fóruns existentes na grande rede (*Internet*).

A **API C** fornece um grande conjunto de funções que podem ser utilizadas de forma embutida num programa codificado em C ou C++. Para uma visão mais ampla sobre este tema recomenda-se consultar o manual de referência que pode ser obtido no endereço: **http://www.lua.org/manual/5.1/pt/**. Consulte no índice do manual as funções da categoria **API C**.

O exemplo seguinte mostra como efectuar o carregamento de um *script* escrito em Lua e armazenado num programa Lua com o nome **teste.lua**. Assim sendo, escreva num editor de texto escreva o código seguinte, gravando-o com o nome **teste.lua**.

```
-- início do programa TESTE
  io.write("Início do script: Lua\n")
  for I = 2, 20, 2 do
    print(I)
  end
  io.write("Fim do script: Lua\n")
-- fim do programa TESTE
```

O programa Lua **teste.lua** quando executado apresentará os valores pares de 0 até 20 saltando a contagem de 2 em 2. Diferentemente de um programa executado pelo ambiente Lua este será executado de dentro de um programa escrito em linguagem C. Assim sendo, observe o código a seguir:

```
// Exemplo 2
#include <stdio.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
int main(void)
{
  lua_State *ACESSO = lua_open();
  luaL_openlibs(ACESSO);
  luaL_dofile(ACESSO, "teste.lua");
  lua_close(ACESSO);
  return 0;
}
```

Grave o programa com o nome **exemplo2.c**. Note o uso da função **luaL_dofile()** que carrega e executa o conteúdo do arquivo indicado, neste caso o arquivo **teste.lua** que possui a capacidade de apresentar os valores pares situados na faixa de 2 a 20.

Recomenda-se como ampliação de conhecimento, a leitura do artigo "**Lua Embarcada a Aplicativos que Podem Ter Script**" disponível em 06/2001 no endereço URL: http://www.ibm.com/developerworks/br/library/l-embed-lua/.

Co-Rotinas

Co-rotina é o nome dado a uma actividade operacional que permite a uma sub-rotina de programa, quer seja um procedimento ou uma função, ser suspenso temporariamente em determinado ponto de execução e venha a ser executada noutro momento. Este tipo de acção também é conhecido como *fluxo de execução colaborativo* (*threads*).

No caso da linguagem Lua, a execução de co-rotinas ocorrer por meio de uma linha de execução independente que faz uso de uma pilha própria de chamadas. Para fazer uso deste recurso Lua possui uma biblioteca (módulo) chamado **coroutine**, a qual disponibiliza para uso algumas funções operacionais, tais como: "**coroutine.create**()"; "**coroutine.status**()"; "**coroutine.resume**()";"**coroutine.wrap**()";"**coroutine.yield**()".

O funcionamento de co-rotinas na linguagem Lua difere do modo encontrado em sistemas convencionais *multithreading (multitarefas)*, pois não efectua qualquer acção que cause a mudança de processamento. Em Lua uma co-rotina só pode ser interrompida quando esta termina ou quando para ela é chamado explicitamente uma acção de suspensão de execução por intermédio da função **yield()**.

Para a criação de co-rotinas pode-se usar as funções **create()** ou **wrap()**. A função **create()** cria uma nova co-rotina mas não a coloca em execução, deixando-a em modo de suspensão e para ser executada necessita da execução da função **resume()**. Já a função **wrap()** cria uma co-rotina que possui como característica a capacidade de ser recomeçada automaticamente sempre que a co-rotina é chamada.A função **yield()** como comentado suspende a execução de uma co-rotina de forma temporária. A função **status()** detecta o estado operacional de uma co-rotina que pode ser:

- ***suspended*** (modo de suspensão quando uma co-rotina tem seu fluxo de execução interrompido);
- ***running*** (modo de execução quando uma co-rotina está em operação);
- ***normal*** (quando uma co-rotina retorna seu fluxo de execução a outra co-rotina);
- ***dead*** (modo de encerramento quando uma co-rotina é normalmente encerrada ou quando é retornado um erro de operação).

Em seguida escreva o código de programa num editor de texto, gravando-o com o nome **corotina1.lua** e execute-o com a linha de comando **lua 5.1 corotina1.lua**.

```
-- início do programa CO-ROTINA1
  rotina = coroutine.create(
             function ()
               print("Olá, Mundo!")
             end
           )
  print(coroutine.status(rotina))
  print(rotina)
  coroutine.resume(rotina)
  print(coroutine.status(rotina))
-- fim do programa CO-ROTINA1
```

Ao ser executado o programa ocorrerá a apresentação da mensagem da primeira linha de código que corresponde a **suspended**, pois assim que a co-rotina foi criada esta fica em estado de suspensão na memória, tanto que ao ser solicitada a acção **print(rotina)** é apresentada uma mensagem de erro identificada por **thread: 0052C388**, que poderá apresentar no seu computador um valor numérico hexadecimal diferente. Esta é a prova de que a co-rotina criada não pode ser executada. Na sequencia ocorre a execução da linha de código com a chamada da função **resume()** que faz a execução da co-rotina e apresenta a mensagem "**Olá, Mundo!**". Na última linha encontra-se novamente a execução da função **status()** que neste momento indica a mensagem **dead**, informando que após a execução da co-rotina, esta foi removida da memória.

O próximo exemplo de programa faz uso da função **yield()**. Observe o código seguinte:

```
-- início do programa CO-ROTINA2
  rotina = coroutine.create(
             function ()
               for I = 10, 16, 2 do
                 print(">> " .. I)
                 coroutine.yield()
               end
             end
           )
  print(coroutine.status(rotina))
  print(rotina)
  coroutine.resume(rotina)
  print(coroutine.status(rotina))
  coroutine.resume(rotina)
  print(coroutine.status(rotina))
  coroutine.resume(rotina)
  print(coroutine.status(rotina))
  coroutine.resume(rotina)
 print(coroutine.status(rotina))

-- fim do programa CO-ROTINA2
```

Em seguida escreva o código de programa num editor de texto, gravando-o com o nome **corotina2.lua** e execute-o com a linha de comando **lua 5.1 corotina2.lua**.

```
-- início do programa CO-ROTINA3
     rotina = coroutine.wrap(
                function (N)
                  R = N
                  print(R)
                  R = N * 2
                  print(R)
                end
              )
     N = 2
     rotina(N)
-- fim do programa CO-ROTINA3
```

Ao ser executado o programa ocorrerá a apresentação da mensagem **suspended** e ocorrerá também a apresentação da mensagem de erro. Em seguida a cada vez que é executada a função **resume()** ocorre a chamada da co-rotina e um valor da variável **I** do laço é apresentado, pois a cada vez que é chamada a co-rotina a função **yield()** faz a interrupção da co-rotina. O programa a seguir faz uso da co-rotina criada com a função **wrap()**. Observe o código sugerido.

```
-- início do programa CO-ROTINA4
     rotina = coroutine.wrap(
                function (N)
                  R = N
                  print(R)
                  coroutine.yield()
                  R = N * 2
                  print(R)
                end
              )
     N = 2
     rotina(N)
     rotina(N)
-- fim do programa CO-ROTINA4
```

Em seguida escreva o código de programa num editor de texto, gravando-o com o nome **corotina3.lua** e execute-o com a linha de comando **lua 5.1 corotina3.lua**. O programa ao ser executado apresenta os valores **2** e **4**. Note que neste exemplo a função **wrap()** possui sua funcionalidade semelhante à função **create()**.

Já no próximo programa a co-rotina criada com a função **wrap()** é interrompida com o uso da função **yield()**. A função **resume()** é usada para iniciar a execução de uma co-rotina criada por meio da função **create()**. Para efectuar um teste de uso de co-rotinas considere um programa que apresente a mensagem "**Olá, Mundo!**", como segue.

Em seguida escreva o código de programa num editor de texto, gravando-o com o nome **corotina4.lua** e execute-o com a linha de comando **lua 5.1 corotina4.lua**. Neste quarto exemplo a co-rotina é chamada de início e apresenta apenas o valor **2**, depois ocorre a segunda chamada que apresenta o valor **4**. Observe que a função **yield()** faz uma interrupção na execução da co-rotina retornando do ponto parado após a sua chamada. Quando se faz uso de co-rotina criada com a função **wrap()** não há a necessidade em usar a função **resume()** para dar continuidade na execução manual da co-rotina, pois a função **wrap()** permite que este avanço seja executado de forma automática.

O exemplo a seguir, é uma adaptação de um programa publicado no livro "Beginning Lua Programming" ("Iniciando Programação em Lua"), páginas 278 e 279 dos auto-res Kurt Jung e Aaron Brownque, da editora Wiley Publishing que mostra os modos de apresentação do *status* no uso de co-rotinas.

```
-- início do programa CO-ROTINA5
  local A, B, C
  local function Status(Str)
    io.write(string.format(" %-11s | ", Str))
    io.write(string.format("A:%-9s | ",
         coroutine.status(A)))
    io.write(string.format("C:%-9s | ",
         coroutine.status(C)))
    io.write(string.format("Estado: [%9s]\n",
         tostring(coroutine.running()
         or "thread: inicial ")))
  end
  function A()
    Status("Rotina(A)")
  end
  function B()
    Status("Rotina(B)")
  end
  function C()
    Status("Rotina(C) 1")
    coroutine.resume(A)
    Status("Rotina(C) 2")
    B()
    Status("Rotina(C) 3")
    coroutine.yield()
    Status("Rotina(C) 4")
  end
  A = coroutine.create(A)
  B = coroutine.wrap(B)
  C = coroutine.create(C)
  Status("Principal 1")
  coroutine.resume(C)
  Status("Principal 2")
  coroutine.resume(C)
  Status("Principal 3")
-- fim do programa CO-ROTINA5
```

Em seguida escreva o código de programa num editor de texto, gravando-o com o nome **corotina5.lua** e execute-o com a linha de comando **lua 5.1 corotina5.lua**. Ao ser o programa executado é apresentado os prognósticos de execução das co-rotinas existentes.

Conclusão

Ao longo deste tempo, desde o número 21 da revista "PROGRAMAR", foi fornecido aos leitores um "mini-curso" introdutório e básico de conhecimento da linguagem de programação Lua, para os falantes do idioma português. Há muita gente que utiliza esta linguagem, mas parece existir poucas pessoas com disposição a compartilhar as suas experiências e conhecimentos. Muitos dos tutoriais, disponíveis e encontrados na internet não atendem plenamente às necessidades dos leitores, mas são um bom começo.

É óbvio também que esta série de artigos sobre a linguagem Lua não é perfeita e nem se teve este intuito, seria muita heresia da nossa parte. Como material inicial esta série é um começo, e assim deve ser tratada pelos nossos amigos leitores. Há muito que percorrer. Há muito o que fazer para se popularizar o conhecimento, não só das tecnologias da informática, mas de diversas áreas do saber humano.

Este é, temporariamente, um último artigo desta série. Diz-se temporariamente, pois há muito a dizer em relação à linguagem Lua. No entanto, também muito foi dito nestes dez artigos ao longo dos últimos meses. Assim, a qualquer instante poder-se-á apresentar novos complementos de estudo e com certeza ter-se-á outras partes, como 11, 12, 13 e sabe-se lá, quantas mais. A diferença entre os próximos artigos sobre Lua e a série que aqui se encerra, é que nos dez capítulos que se completam com este foi feito um "mini-curso" para novatos em programação Lua, e os próximos artigos abordarão temáticas direccionadas a um público que a partir deste artigo já possui algum conhecimento desta linguagem.

AUTOR

Escrito por Augusto Manzano

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

Elege o melhor artigo desta edição

Revista PROGRAMAR

http://tiny.cc/ProgramarED30_V

Criar um sistema RSS no Sharepoint através de uma lista de páginas

Introdução

Quem já utilizou as facilidades RSS do SharePoint certamente já se deparou com alguns problemas de configuração ao nível do output da informação que sai com RSS Standard, como por exemplo a designação dos campos e os longos caminhos que temos de utilizar do tipo:

_layouts/listfeed.aspx?List=ab4a2784-8309-4905-88be-71449afcd9a3&View=6db1464a-f5a1-43de-9526-dc94b064052e

Neste artigo vamos abordar uma forma simples e eficaz de através de RSS 2.0 fazer um "Response" directamente numa página ASPX com um Controlo de Utilizador

Estrutura XML RSS 2.0

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
  <channel>
    <title>RSS Title</title>
    <description>This is an example of an RSS feed</description>
    <link>http://www.someexamplerssdomain.com/main.html</link>
    <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
    <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
    <item>
      <title>Example entry</title>
      <description>
        Here is some text containing
        an interesting description.
      </description>
      <link>http://www.wikipedia.org/</link>
      <guid>unique string per item</guid>
      <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
    </item>
  </channel>
</rss>
```

Fonte: http://en.wikipedia.org/wiki/Rss

Considerações Iniciais

A primeira consideração que temos de ter em atenção é que para efectuarmos o nosso *response* não podemos utilizar nenhum Layout de Página, temos de utilizar uma página que "saia" do SharePoint como acontece em todos os procedimentos deste tipo.

Neste exemplo vamos utiliza uma página ASPX e vamos adicionar um Controlo de Utilizador à mesma, visto a grande maioria das Aplicações SharePoint assentar em camadas, uma delas certamente a de estrutura do nosso site onde colocamos todas as nossas páginas ASPX, Controlos de Utilizador, Layouts e finalmente as nossas Features por forma a propagarmos a nossa estrutura na altura da Instalação do Portal activando-as.

Vamos separar o nosso CAML Query do método principal por forma a termos uma maior percepção do código.

Método CAML Query

Vamos criar um método Protected invocando o Object Model através do SPQuery.

```
protected SPQuery ONossoCamlQuery()
 {
     SPQuery qry = new SPQuery();
     qry.ViewAttributes += "Scope=\"Recursive\"";

     string camlQry = String.Empty;
     qry.Query = camlQry;
```

O próximo passo, a nossa String para a Query. Neste passo temos de modificar 'OSeuCampoData' para o nome do campo que queremos filtrar, em que sugiro a utilização do CAML QUERY BUILDER

(http://www.u2u.be/res/Tools/CamlQueryBuilder.aspx). Deverá funcionar em ambos MOSS e SharePoint 2010, pelo menos para elaborarmos o nosso filtro.

Vamos deixar "Ascending='False'" para ser apresentado por ordem decrescente.

```
    try
    {
        camlQry = "<OrderBy><FieldRef
        Name='OSeuCampoData' Ascending='False'/></
        OrderBy>";
        qry.Query = camlQry;
    }
    catch (Exception)
    {
        throw;
    }
```

Vamos limitar o nosso resultado a 10 páginas, caso contrário poderá ser lento e não funcionar correctamente visto que limitamos o nosso tempo limite de cache por forma a ser eficaz em todos os RSS Readers.

```
    qry.RowLimit = 10;
```

O método completo:

```
protected SPQuery ONossoCamlQuery()
 {
     SPQuery qry = new SPQuery();
     qry.ViewAttributes += "Scope=\"Recursive\"";

     string camlQry = String.Empty;
     qry.Query = camlQry;

     try
     {
         camlQry = "<OrderBy><FieldRef
         Name='OSeuCampoData' Ascending='False' /
         ></OrderBy>";
         qry.Query = camlQry;
     }
     catch (Exception)
     {
         throw;
     }
     qry.RowLimit = 10;
     return qry;
 }
```

Limite de tempo de Cache

Primeiro vamos definir no ASPX do nosso controlo a duração do Cache de Saída, que será a duração permitida do Output-Cache por forma a não ficarmos com pedidos longos. Mais informação em (ASP.NET Caching)

http://msdn.microsoft.com/pt-br/library/xsbfdd8c.aspx

<%@ OutputCache Duration="120" VaryByParam="*" %>

Método para construção do RSS

O método consiste em criarmos uma Response e dinamicamente renderizar o XML Standard enumerando uma Lista de Páginas de SharePoint filtrado pelo nosso CAML Query

```
readonly string NossoUrlNivelRoot =
         String.Format("{0}//{1}:{2}",
         SPContext.Current.Site.Protocol,
         SPContext.Current.Site.HostName,
         SPContext.Current.Site.Port);
```

Vamos buscar o URL de Root dinamicamente:

Dentro do URL de Root vamos buscar a área onde está a lista de páginas, exemplo “Noticias”

```
string URL = NossoUrlNivelRoot + "/
```

Limpamos a nossa Response e iniciamos uma nova baseada em XML.

```
Response.Clear();
Response.ContentType = "text/xml";
```

Definir o *encoding* para UTF8 e começar a escrever o XML na página.

```
XmlTextWriter TextWriter = new XmlTextWriter

       (Response.OutputStream, Encoding.UTF8);
TextWriter.WriteStartDocument();
```

TAGS mandatórias RSS 2.0

```
TextWriter.WriteStartElement("rss");
TextWriter.WriteAttributeString("version", "2.0");
```

O Channel TAG que vai conter a informação do FEED criando o nosso nó XML “channel”, dentro deste iremos criar a seguir outro nó chamado “item” com a informação das nossas páginas SharePoint.

```
TextWriter.WriteStartElement("channel");
TextWriter.WriteElementString("title",
   "JOAO TITO LIVIO");
TextWriter.WriteElementString("link",
  "http://msmvps.com/blogs/officept/default.aspx");
TextWriter.WriteElementString("description",
   "Espero que gostem deste RSS");
```

```
TextWriter.WriteElementString("copyright",
   "Copyright 2011 Joao Tito Livio. All rights re-
served.");
```

Abrir o nosso WEB usando o Object Model vamos utilizar Using por forma a não termos problemas de dispose dos objectos do SharePoint.

Abrir a nossa lista através do SPList e seguidamente enumerar as páginas através de uma SPListItemCollection. Nesta fase vamos chamar o nosso método em que definimos o nosso CAML Query. Vemos também que a nossa lista se chama "Pages", mude para "Paginas" ou o nome que deu à mesma.

```
SPList ListaDePaginas = web.Lists["Pages"];
SPListItemCollection ColecaoListaDePaginas = Lista-
DePaginas.GetItems(ONossoCamlQuery());
```

Agora dentro da nossa colecção de páginas vamos buscar a informação de cada uma através de um SPItem a que chamo elemento verificando sempre se dentro de cada elemento ele é nulo e começando a escrever um nó de XML chamado "item" citado anteriormente. Temos de ter em atenção a mudança dos nomes dos campos a que se refere cada elemento pelos definidos por si.

```
SPItem elemento = ColecaoListaDePaginas[z];
TextWriter.WriteStartElement("item");
if (elemento["CampoTituloPaginaSharepoint"]!= null)
      TextWriter.WriteElementString("title",
      elemento["CampoTituloPaginaSharepoint"]
      .ToString());
```

```
if (elemento["CampoDescricaoPaginaSharepoint"] !=
null)
      TextWriter.WriteElementString("description",
      elemento["CampoDescricaoPaginaSharepoint"]
      .ToString());

if (elemento["CampoLinkPaginaSharepoint"] != null)
```

Neste elemento string "link" estamos a construir dinamicamente o Link para a lista de páginas que abrimos anteriormente, novamente vemos que a nossa lista se chama "Pages", mude para "Paginas" ou o nome que deu à mesma.

```
TextWriter.WriteElementString("link",
       String.Format("{0}/Pages/{1}",
       URL,elemento["CampoLinkPaginaSharepoint"]));

 if (elemento"CampoDataPublicacaoPaginaSharepoint"]
  != null)

    TextWriter.WriteElementString("pubDate",
    elemento["CampoDataPublicacaoPaginaSharepoint"]
    .ToString());

TextWriter.WriteEndElement();
```

Vamos fechar todos os elementos e acabar a nossa Response.

```
TextWriter.WriteEndElement();
TextWriter.WriteEndElement();
TextWriter.WriteEndDocument();
TextWriter.Flush();
TextWriter.Close();
Response.End();
```

Método completo

```
protected void  GetYourRss()
{
  readonly string  NossoUrlNivelRoot = String.Format("{0}//{1}:{2}",
                                                     SPContext.Current.Site.Protocol,
                                                     SPContext.Current.Site.HostName,
                                                     SPContext.Current.Site.Port);
  try
   {
      string  URL = NossoUrlNivelRoot + "/NossaAreaDentroDaRoot";
      Response.Clear();
      Response.ContentType = "text/xml";
      XmlTextWriter  TextWriter = new  XmlTextWriter(Response.OutputStream, Encoding.UTF8);

      TextWriter.WriteStartDocument();
      TextWriter.WriteStartElement("rss");
      TextWriter.WriteAttributeString("version", "2.0");
      TextWriter.WriteStartElement("channel");
      TextWriter.WriteElementString("title", "JOAO TITO LIVIO");
      TextWriter.WriteElementString("link", " http://msmvps.com/blogs/officept/default.aspx");
      TextWriter.WriteElementString("description", "Enjoy your RSS");
```

```
        TextWriter.WriteElementString("copyright", "Copyright 2011 Joao Tito Livio. All rights reserved.");

    using (SPSite site = new SPSite(URL))
      {
          using (SPWeb web = site.OpenWeb())
          {
              SPList ListaDePaginas = web.Lists["Pages"];
              SPListItemCollection ColecaoListaDePaginas = ListaDePaginas.GetItems(ONossoCamlQuery());
              for (int z = 0; z < ColecaoListaDePaginas.Count; z++)
              {
                  SPItem elemento = ColecaoListaDePaginas[z];
                  TextWriter.WriteStartElement("item");
                  if (elemento["CampoTituloPaginaSharepoint"] != null)
                      TextWriter.WriteElementString("title",
                              elemento["CampoTituloPaginaSharepoint"].ToString());

                  if (elemento["CampoDescricaoPaginaSharepoint"] != null)
                      TextWriter.WriteElementString("description",
                              elemento["CampoDescricaoPaginaSharepoint"].ToString());

                  if (elemento["CampoLinkPaginaSharepoint"] != null)
                      TextWriter.WriteElementString("link",
                          String.Format("{0}/Pages/{1}", URL, elemento["CampoLinkPaginaSharepoint"]));

                  if (elemento["CampoDataPublicacaoPaginaSharepoint"] != null)
                      TextWriter.WriteElementString("pubDate",
                              elemento["CampoDataPublicacaoPaginaSharepoint"].ToString());

                  TextWriter.WriteEndElement();
              }
          }
      }
      TextWriter.WriteEndElement();
      TextWriter.WriteEndElement();
      TextWriter.WriteEndDocument();
      TextWriter.Flush();
      TextWriter.Close();
      Response.End();
  }
  catch (Exception ex)
  {
      throw new Exception(ex.ToString());
  }
}
```

Considerações Finais

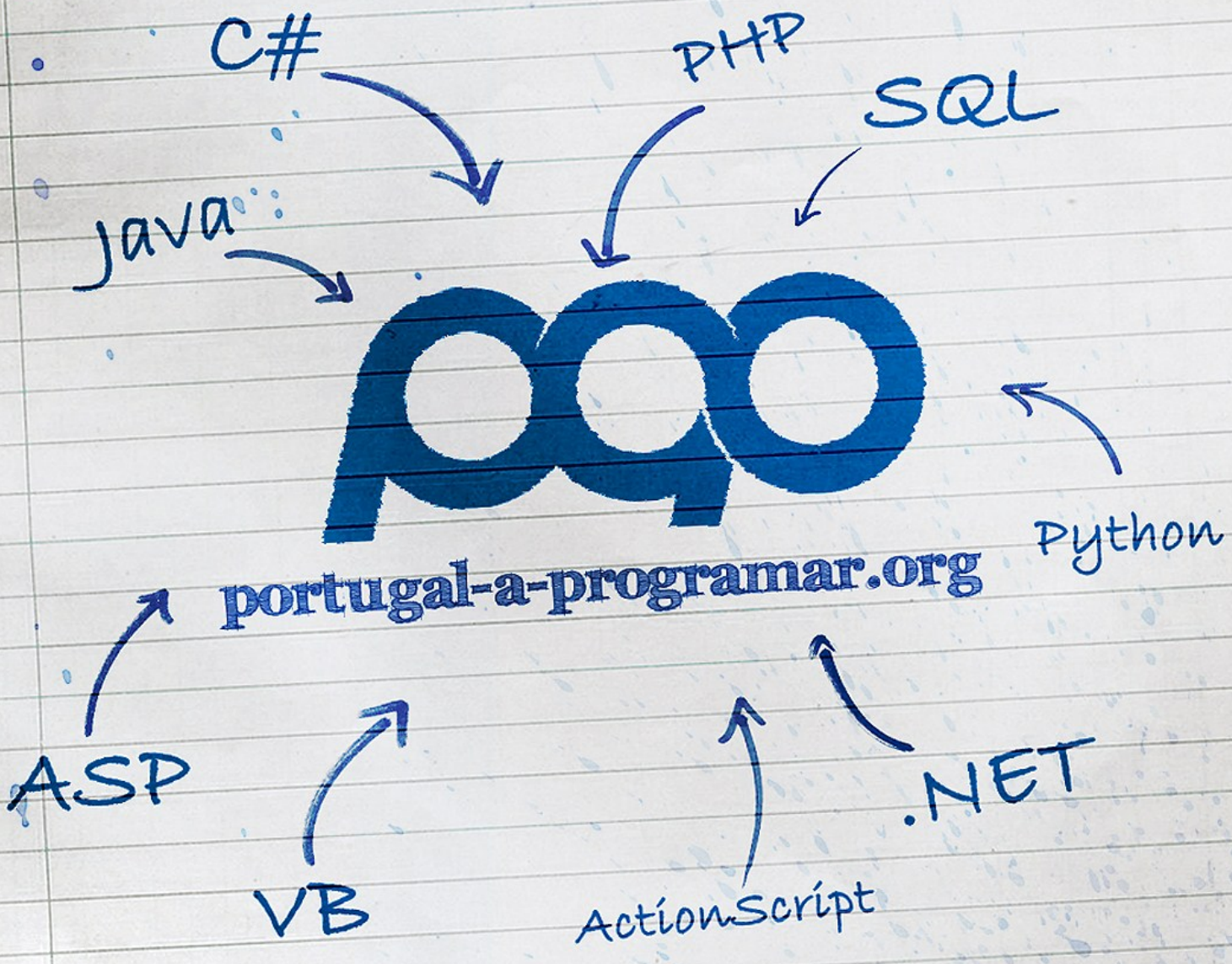
Desde que comecei a programar em SharePoint pela mão do MVP SharePoint André Lage, das primeiras coisas que me transmitiram foi que "já existe em SharePoint não se vai fazer de novo", mas realmente neste caso prefiro esta solução. Deixo ao vosso critério HAPPY CODING!

AUTOR

Escrito por João Tito Lívio
É MCP e MCTS em SharePoint 2010 Development, neste momento trabalha como Consultor e Programador na CIL – Centro de Informática, SA.

É Microsoft Most Valuable Professional (MVP) na categoria de Office Systems Development desde 2002

A maior comunidade portuguesa de programação!



Visite-nos!

Introdução ao Objective-C e à plataforma IOS

Objective C

O Objective-C é uma linguagem dinâmica baseada nas linguagens C e Smalltalk, uma procedimental e outra orientada a objectos. Na realidade o Objective-C não é nada mais do que a linguagem C com esteróides, adicionando à potência da linguagem C o sistema de mensagens do Smalltalk.

Como a linguagem C é parte integrante do Objective-C, grande parte da sintaxe é comum entre estas duas linguagens.

Os tipos primitivos, declaração de funções e chamada de funções da linguagem C são exactamente iguais em Objective-C e, por outro lado, todas as funcionalidades que são orientadas a objectos implementam o sistema de mensagens do Smalltalk. O leitor tem até a liberdade de integrar código desenvolvido na linguagem C/C++ no seu programa de Objective-C e utiliza-lo sem qualquer tipo constrangimento.

iOS

É o sistema operativo proprietário da Apple utilizado nos seus dispositivos móveis, como o iPhone e o iPad. O seu principal trunfo é o user interface, que permite uma utilização natural e agradável graças à utilização de gestos e acções multi-toque como meio de interacção com o dispositivo.

O iOS encontra-se divido em quatro camadas: Core OS, Core Services, Media Layer e Cocoa Touch. Para facilitar o desenvolvimento de aplicações para iOS, a Apple disponibiliza a iOS SDK, um kit de desenvolvimento, que apenas é possível instalar em computadores Apple Macintosh, e contém todas as ferramentas necessárias para desenvolver aplicações para a o iOS.

CocoaTouch

É uma das principais camadas do iOS, disponibiliza o acesso a um conjunto de API's que facilitam o desenvolvimento de aplicações e permite a utilização de algumas funcionalidades chave do iOS, como a câmara fotográfica, GPS, áudio e vídeo.

Sintaxe

Em Objective-C as classes são compostas por duas componentes distintas, interface e implementação. Encontram-se divididas em ficheiros onde a extensão que indica o seu propósito, quando se trata de um ficheiro com a extensão ".h" trata-se da declaração do interface da classe e quando o ficheiro tem como extensão ".m" trata-se da implementação da classe.

Na figura 1, apresentada em baixo, está definido o interface da classe Cliente. No caso de o leitor ter experiência nas linguagens orientadas a objectos mais comuns, como Java ou C#, é natural que cause alguma estranheza e até confusão a sintaxe do Objective-C, no entanto essa sensação é ultrapassada rapidamente depois de compreendidos os conceitos básicos da linguagem.

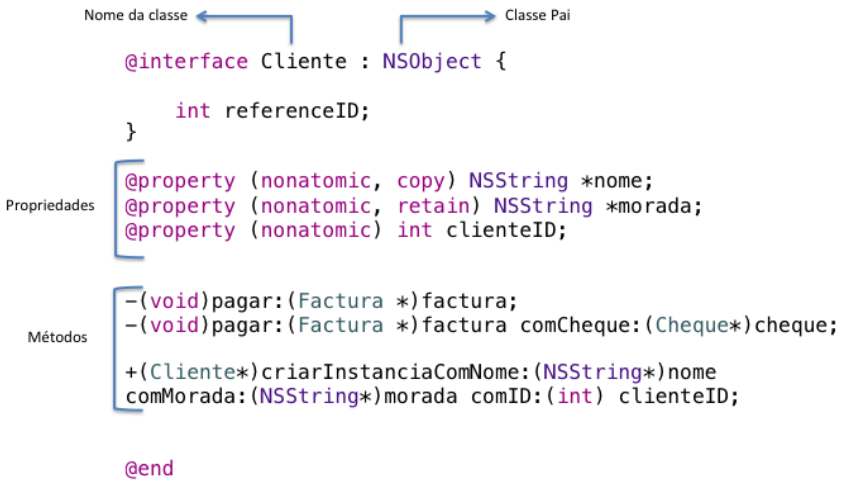


Figura 1- Interface da classe Cliente

A interface é o local apropriado para definir as propriedades e funcionalidades de uma classe em Objective-C.

Na acima apresentada, o leitor pode identificar alguns dos elementos principais de uma classe como variáveis, propriedades, métodos e herança. Existem algumas particularidades que são essenciais esclarecer nesta fase:

- NSString é uma classe de Objective-C que permite representar um conjunto de caracteres.
- Uma propriedade é definida pelo nome, tipo e atributos. Os atributos são utilizados para especificar o comporta-

mento da propriedade.

- Quando a assinatura de um método é identificado pelo carácter "+", trata-se de um método estático e quando o é identificado pelo carácter "-" trata-se de um método de instância.
- Na assinatura de um método, os parâmetros são definidos através da seguinte sintaxe: palavraChave: **tipoDoParametro nomeDoParametro**

```
#import "Cliente.h"

@implementation Cliente

@synthesize nome;
@synthesize morada;
@synthesize clienteID;


-(void)pagar:(Factura *)factura{

    NSLog(@"O cliente %@ vai pagar factura nº %@",
          nome, factura.facturaID);
}

-(void)pagar:(Factura *)factura comCheque:(Cheque*)
cheque{

    NSLog(@"O cliente %@ vai pagar factura nº %@
          com o cheque nº %@",
          nome, factura.facturaID, cheque.numero);
}

+(Cliente*)criarInstanciaComNome:(NSString*)nome
comMorada:(NSString*)morada comID:(int) clienteID{

    Cliente *cli = [[Cliente alloc] init];
    cli.clienteID = clienteID;
    cli.morada=morada;
    cli.nome =nome;

    return [cli autorelease];
}

@end
```

Figura 2- Implementação da classe Cliente

É no ficheiro de implementação que, como o próprio nome refere, é implementado tudo o que foi definido no ficheiro de interface da classe Cliente. É importante lembrar que é necessário fazer referência ao ficheiro onde está declarado o interface da classe.

É necessário definir através da palavra-chave @synthetize, as propriedades no ficheiro de implementação para que o compilador gere de forma automática os assessores das propriedades.

O Objective-C encarrega o programador das tarefas de gestão de memória, logo sempre que se cria uma instância de uma classe, é necessário alocar e inicializar espaço de memória para o objecto que é instanciado, bem como se torna necessário proceder à sua libertação o objecto não é mais necessário.

```
Cliente *cli = [[Cliente alloc] init];
cli.clienteID = 20;
cli.morada=@"1 Infinite Loop Cupertino, CA 95014";
cli.nome =@"Apple Computer, Inc.";

Factura *fact = [[Factura alloc] init];
fact.facturaID = @"NSOF556323";

Cheque *cq = [[Cheque alloc] init];
cq.numero = @"AB656561354546562 65";

[cli pagar:fact];
[cli pagar:fact comCheque:cq];
```

Figura 3 - Criação e destruição de objectos em Objective-C

Na figura 3, o leitor pode observar como se instância um objecto, atribui valores às suas propriedades, executa um método e, quando o propósito do objecto foi cumprido, se procede à sua destruição.

Em Objective-C, os objectos são criados com a sintaxe Classe *apontador = [[Classe alloc]init]; os métodos são executados com a através de [objecto nomeDoMetodo]; e os objectos são libertados quando se executa o método [objecto release];

Assim sendo, o resultado da execução do código da figura 3, com base na implementação executada na figura 2 e definida na figura 1 é o seguinte:

> *"O cliente Apple Computer, Inc. vai pagar factura nº NSOF556323"*
>
> *"O cliente Apple Computer, Inc. vai pagar factura nº NSOF556323 com o cheque nº AB656561354546562 65"*

Tap Counter

Depois de o leitor instalar o iOS SDK, tem à sua disposição um conjunto de ferramentas que lhe permite desenvolver aplicações para iOS. Entre as ferramentas incluídas no iOS SDK está o XCode IDE e o iOS Simulator.

O Tap Counter é uma aplicação cujo único objectivo é manter um contador do número que vezes que o utilizador pressiona um botão.

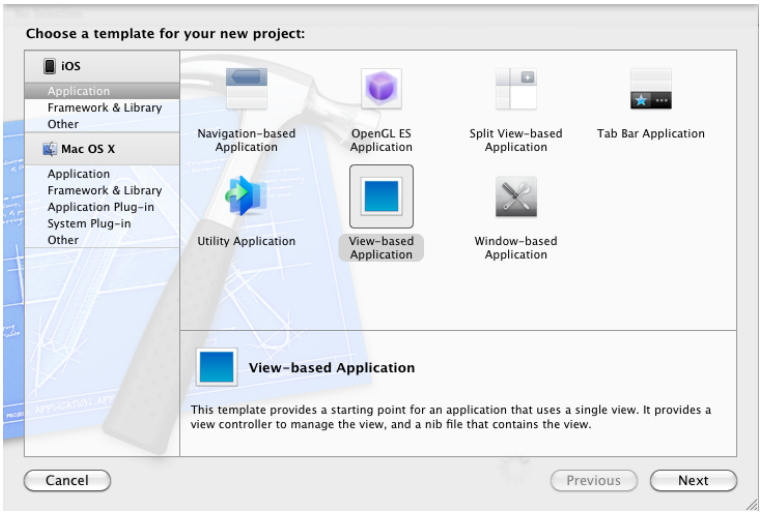


Figura 4 - Criar um projecto no XCode

Execute o XCode4 IDE e crie um novo projecto do tipo "View -based Application" como exemplificado na figura 4 e atribua o nome PT_AT_PROG ao projecto.

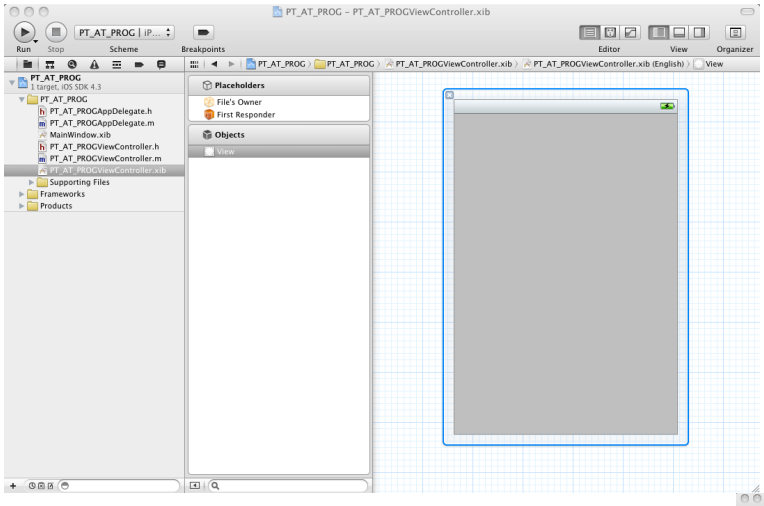


Figura 5 - XCode4 IDE

Na barra lateral esquerda, seleccione o ficheiro PT_AT_PROGViewController.h" para definir a interface a utilizar na aplicação.

```
@interface PT_AT_PROGViewController : UIViewCon-
troller {

    IBOutlet UILabel *countLabel;
    IBOutlet UIButton *tapButton;
    int count;
}
@end
```

Figura 6 – Definição da interface da aplicação Tap Counter

Como se trata de uma aplicação extremamente simples, o leitor apenas necessita de definir como está indicado na figura 6, um campo de texto (UILabel) para colocar o texto com o resultado da contagem, um botão (UIbutton) e uma variável para guardar a contagem das vezes que o botão foi pressionado.

O prefixo IBOutlet serve para anunciar ao Interface Builder, que é a ferramenta que permite desenhar o UI, que estas variáveis necessitam de estar referenciadas.

Na barra lateral esquerda, seleccione o ficheiro "PT_AT_PROGViewController.m" para implementar a interface definida.

```
- (void)viewDidLoad
{

    [super viewDidLoad];
    count = 0;
}


-(IBAction) buttonTouchDown:(id)sender{
  count++;
  [countLabel setText:[[NSString alloc]
  initWithFormat:@"TapCount:%d",count]];
}
```

Figura 7 – Implementação da interface definida

No ficheiro de implementação, o leitor apenas necessita de inicializar a variável count e implementar o evento buttonTouchDown.

Agora que o leitor tem concluído o código em Objective-C necessário é necessário desenvolver um User Interface que permita aos utilizadores interagirem com a aplicação.

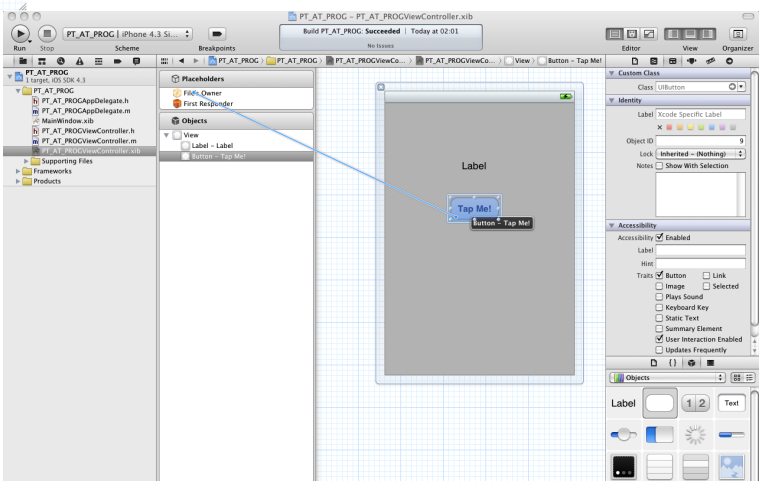


Figura 8 – Implementação da UI

No XCode selectione um UIButton e uma UILabel e construa um User Interface conforme a figura 8.

De seguida seleccione no inspector, na secção Outlets o tapButton que o leitor definiu no ficheiro de interface, e pressionando o botão do rato, arraste o cursor até ao Button que se encontra na barra vertical Objects, e repita o mesmo exercício para o countLabel e Label.

Isto serve apenas para estabelecer uma relação entre objectos de UI e objectos definidos na interface.

Falta apenas estabelecer a relação entre o evento buttonTouchDown e o tapButton declarado no ficheiro de interface. Para estabelecer essa relação, é necessário seleccionar no inspector, na secção Received Actions o evento buttonTouchDown, seleccionar com o rato, arrastar o cursor até à barra Objects e, quando o leitor largar o cursor sobre o Buttom e seleccionar Touch Down para assim associar o evento buttonTouchDown ao UIButton.

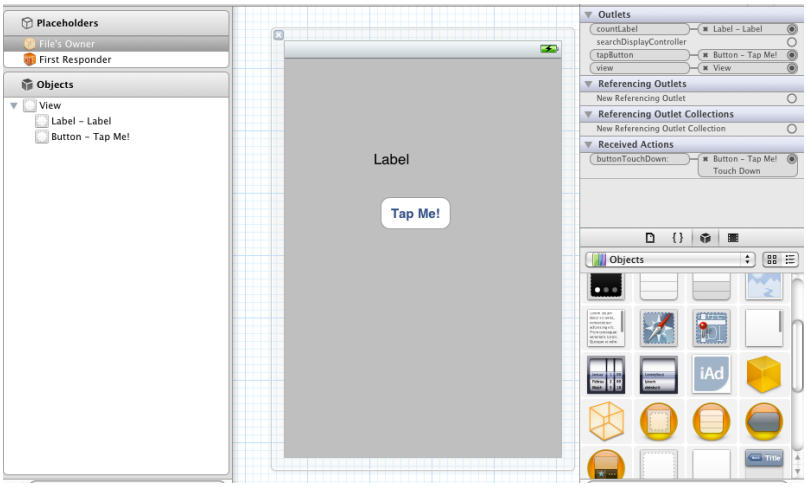
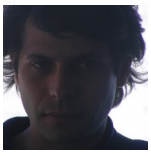


Figura 9 – Aspecto final da barra de inspector, Outlets e Received Actions

Para concluir, apenas é necessário executar o projecto ou CMD+R e vai-se inicializar o Tap Couter no iOS Simulator.

Figura 10 – Tap Counter a ser executado no iOS Simulator

AUTOR



Escrito por Bruno Pires

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

Atributos em C#

"An attribute is a piece of additional declarative information that is specified for a declaration." [1]

"Um atributo é um pedaço de informação declarativa adicional que é especificado para uma declaração.[1]

Introdução

Um programa em C#, para além do código e dos dados, contém também metadados. Informação acerca do programa em si. Esses metadados são colocados na aplicação através de Atributos.

Esses atributos podem existir ao nível da aplicação, classe, propriedade, método, etc.. Alguns exemplos podem ser vistos dentro do ficheiro AssemblyInfo.cs:

```
[assembly: AssemblyTitle("MyApplication")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("MyApplication")]
[assembly: AssemblyCopyright("Copyright ©  2009")]
```

Os atributos, definidos ao nível da aplicação, contêm informação da empresa ou pessoa que a desenvolve. Podem depois ser visualizados acedendo às propriedades do ficheiro executável, após compilação.

Um atributo que já deverá ter usado é o atributo Serializable. Indica que todos os campos, públicos e privados, deste tipo podem ser serializados.

Criar um Atributo

Para criar um novo atributo basta criar uma nova classe que derive, directa ou indirectamente, da classe abstracta Attribute:

```
public class DescriptionAttribute : Attribute
{
```

Aqui temos um atributo válido em C# que pode ser usado por qualquer classe, property, etc., em qualquer aplicação. No entanto não tem grande utilidade.

Vamos adicionar um bocadinho mais de informação a esta classe:

```
[AttributeUsage(AttributeTargets.Class,
 AllowMultiple = true,Inherited = false)]
public class DescriptionAttribute : Attribute
{
  private String author;
  private String description;
  public double version;

  public DescriptionAttribute(String author, String
description)
  {
    this.author = author;
    this.description = description;
    this.version = 1.0;
  }
}
```

Para controlar o uso deste atributo usamos um outro, AttributeUsage. Aqui dizemos que este atributo vai ser apenas usado em classes passando-lhe o Enum AttributeTargets. Podemos validar a utilização do atributo em Properties, Métodos, etc.. Podemos também permitir combinações destes, ou em todos.

O campo AllowMultiple indica, que na mesma classe, se pode usar este atributo várias vezes. O campo Inherited a false refere que, classes que derivem da classe que usa este atributo não têm a mesma informação, ou seja, se necessitarem de o usar têm de o fazer explicitamente.

NOTA: É uma convenção que se use o sufixo "Attribute" nos nomes das classes. Tal não é obrigatório pois o compilador procura por classes que herdem de System.Attribute com o nome da classe. Caso não encontre, adiciona a palavra Attribute e repete a pesquisa. Isso é considerado no entanto uma boa prática.

Usar um atributo

Este atributo que acabámos de usar guarda um histórico das classes na nossa aplicação:

```
[Description("Flavio", "Attributes example.")]
[Description("Miguel", "Updating version", ver-
sion=1.1)]
public class MyClass
{
  // Codigo do Flavio
  // Codigo do Miguel
}
```

Neste exemplo, numa classe em que já dois programadores participaram podemos ver os dois tipos de parâmetros que

os Atributos suportam: posicionais ou com nome.

Como parâmetros posicionais temos o autor e a descrição. São dois parâmetros privados da classe cujo valor é passado no construtor. São chamados posicionais pois, no construtor, são definidos pela posição em que estão, primeiro o autor e depois a descrição.

O campo versão é um campo que é referido por nome. Não é usado na definição do construtor mas é passado através deste na mesma. Visto ser uma variável pública o compilador relaciona o nome da variável com aquele que é passado no construtor.

Tornar o atributo útil

Até agora já criamos o atributo e colocámo-lo numa classe. No entanto ainda não nos serve de muito. Temos de ter uma maneira de aceder à informação que lá está. Para isso começamos por criar propriedades que nos permitam aceder à informação guardada:

```
public String Author { get { return
this.author; } }
public String Description { get { return
this.description; } }
public Double Version { get { return
this.version; } }
```

Visto que apenas queremos aceder ao que está no atributo, e não alterar o seu valor, podemos implementar apenas os métodos *get*.

Em seguida precisamos de um método que seja capaz de receber uma instância de um objecto e imprimir a sua informação. Para tal precisamos de usar reflexão para saber qual o tipo de objecto com que estamos a lidar.

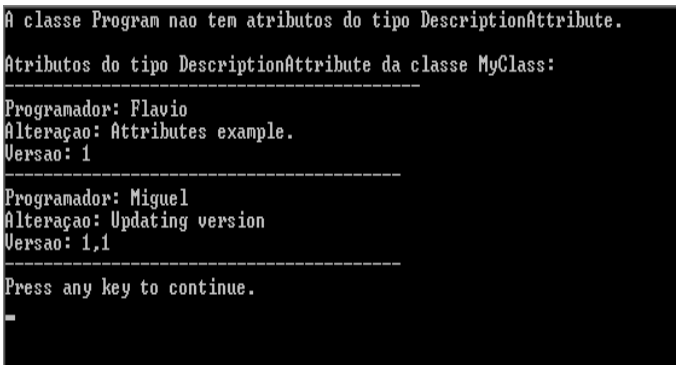
De seguida vamos buscar os atributos que queremos para poder enviar a informação para o ecrã. A explicação do código encontra-se em comentário:

```
private static void PrintDevelopersInformation
(Object myClass)
 {
  // Precisamos de saber qual o tipo
  // do objecto que recebemos
  Type type = myClass.GetType();

 /* Ao tipo de objecto vamos buscar os atributos
  * DescriptionAttribute este parâmetro é opcional.
  * Se não o colocarmos são devolvidos todos os
  * atributos. O parâmetro "false" indica que não
  * queremos subir pela cadeia de herança, ou seja,
  * apenas queremos os atributos desta classe em
  * particular. Como estamos a pedir uma lista
  * de um tipo de atributos especifico podemos
  * fazer o cast
  */
     DescriptionAttribute[] attributeArray =
       (DescriptionAttribute[])type.
       GetCustomAttributes(
       typeof(DescriptionAttribute),false);

     // Se o nosso array estiver vazio é porque
     // esta classe não tem atributos.
     if (attributeArray.Length == 0)
     {
       Console.WriteLine(
        "A classe {0} não tem atributos do tipo " +
        "DescriptionAttribute.", type.Name);
        Console.WriteLine();
     }
     else
     {
       // Caso o array tenha objectos vamos
       // percorre-los e imprimi-los.
       Console.WriteLine("Atributos do tipo Des-
criptionAttribute da classe {0}:",type.Name);

       Console.WriteLine(new string('-', 45));

       foreach (DescriptionAttribute description
                              in attributeArray)
       {
          / Podemos aceder directamente às
         // properties que criámos anteriormente.
         Console.WriteLine(
             "Programador: " +
            description.Author + "\n" +
             "Alteração: " +
            description.Description + "\n" +
             "Versão: " + description.Version);

          Console.WriteLine(new string('-', 45));
       }
     }
 }
```

Agora que temos um método que nos imprime as informações dos objectos que lhe passámos, vamos usá-lo:

```
class Program
{
  static void Main(string[] args)
  {
    Program program = new Program();
    PrintDevelopersInformation(program);
    MyClass myClass = new MyClass();
    PrintDevelopersInformation(myClass);
    WaitForKeyPressAndExit();
  }

  private static void PrintDevelopersInformation
(Object myClass)
  {
    // O nosso código
  }

  private static void WaitForKeyPressAndExit()
  {
   Console.WriteLine("Press any key to continue.");
   Console.ReadKey();
  }
}
```

Como se pode ver, instanciamos primeiro um objecto do tipo Program, que não tem o nosso atributo, e chamamos o nosso método. Seguidamente criamos um objecto MyClass, onde colocámos o atributo DescriptionAttribute, e voltamos a chamar o método com este novo objecto. Executando esta aplicação obtemos o seguinte resultado:



Podemos agora manter e mostrar um histórico das alterações feitas nas classes dos nossos projectos.

E para que serve isto?

Convenhamos que guardar o histórico de alterações de uma classe dentro do código da própria classe não é particularmente útil. Ainda mais quando temos bastante software de controlo de versões que faz isso muito melhor. Para que servem então estes atributos? Como foi dito no início um atributo é uma classe como qualquer outra que tem a particularidade de herdar de Attribute. Sendo uma classe pode ter, para além de variáveis e properties, métodos que implementem funcionalidades específicas.

Quem já desenvolveu Web Applications em .Net já terá usado validadores: objectos da Framework que verificam o valor de certos campos ou controlos e devolvem um erro, caso o seu valor não seja válido.

No entanto estes objectos não podem ser usados em Windows Forms ou na validação de qualquer outro objecto que desejemos. Vamos então fazer nós um conjunto de validadores que podemos usar no nosso código.

Como queremos ter vários tipos de validadores precisamos de definir uma estrutura comum. Desta forma podemos tratar qualquer validador da mesma forma. Criamos então uma classe que nos vai dizer o que é necessário para ser um validador:

```
[AttributeUsage(AttributeTargets.Property,
  AllowMultiple = true, Inherited = true)]
public abstract class ValidatorAttribute : Attrib-
ute
{
  public abstract Boolean Validate(Object obj);
}
```

Visto que vamos validar campos, este validador vai ser usado nas propriedades das classes, permitimos, ainda, que cada propriedade tenha vários validadores. Estas características vão ser utilizadas pelas classes que herdem desta. Esta classe é abstracta pois não define comportamento, apenas estrutura. As classes que irão fazer a validação irão herdar desta e serão forçadas a implementar o método Validate.

Vamos então criar uma classe que nos informa se um determinado campo é um valor inteiro:

De seguida precisamos de criar uma classe que verifique a

```
public class IntegerValidatorAttribute : ValidatorAt-
tribute
{
  public override bool Validate(object obj)
  {
   int value;
   if (obj == null) //verificamos se é nulo
     return false;
   return Int32.TryParse(obj.ToString(),out value);
  }
}
```

validade de um objecto que lhe seja passado. Para simplificar, vamos criar uma classe estática apenas com o método de validação. Como anteriormente, a explicação do código encontra-se nos comentários:

```
public static class Validator
{
  public static Boolean checkIfIsValid(Object obj)
  {
  // Precisamos de saber qual o
  // tipo do objecto que recebemos
  Type type = obj.GetType();

  // Com essa informação vamos buscar
  // as properties do objecto
  PropertyInfo[] properties = type.GetProperties();

  foreach(PropertyInfo prop in properties){

      // Para cada property vamos buscar
      // os objectos de validação
      ValidatorAttribute[] attributesList =
          ValidatorAttribute[])
      prop.GetCustomAttributes(
          typeof(ValidatorAttribute), false);

      foreach (ValidatorAttribute attribute in
          attributesList)
      {
        // Para cada atributo verificamos
       // se o valor é válido
        if (!attribute.Validate(prop.GetValue(
            obj, null)))
          return false;
      }
    }
    return true;
  }}
```

Em seguida chamamos a validação do objecto no nosso código:

```
static void Main(string[] args)
  {
   ClassTobeValidated classe =
       new ClassTobeValidated();

    Console.WriteLine("Is class valid (" +
       classe.field+")?" + Validator.checkIfIsValid
       (classe).ToString());

    classe.field = "texto";
    Console.WriteLine("Is class valid (" +
       classe.field+")? "+ Validator.checkIfIsValid
       (classe).ToString());

    classe.field = "12";
    Console.WriteLine("Is class valid (" +
       classe.field+")?" + Validator.checkIfIsValid
       (classe).ToString());

    WaitForKeyPressAndExit();
}
```

Correndo este código obtemos o seguinte resultado:

```
Is class valid ()? False
Is class valid (texto)? False
Is class valid (12)? True
Press any key to continue.
```

Como podemos ver, quando passamos a variável a vazia ou com um texto temos um objecto inválido. Quando passamos uma string que representa um inteiro temos uma resposta válida. Apesar de ser uma validação relativamente simples (fica a cargo do leitor encontrar formas mais eficientes de verificar a validade do campo), se existissem muitos campos na mesma classe, seria necessário efectuar várias vezes a mesma validação. Com um validador basta indicar em cada variável que validação se pretende fazer.

Vamos agora experimentar fazer um validador um pouco mais complexo, que receba um parâmetro que indique a validação a ser feita. Vamos criar um validador que recebe uma expressão regular e verifica se o campo no qual é usado, é válido segundo essa expressão regular:

```
public class RegularExpresionValidatorAttribute :
ValidatorAttribute
{
  // A nossa expressao regular
  private Regex expression;

  public RegularExpresionValidatorAttribute(String
expression)
  {
    // Criamos uma expressão regular com
    // base na string que recebemos
    this.expression = new Regex(expression);
  }

  public override bool Validate(object obj)
  {
    // verificamos se é null
    if (obj == null)
      return false;
    // Verificamos se o nosso objecto é valido
    return this.expression.IsMatch(obj.ToString());
  }
}
```

Agora podemos usar este validador num objecto. Como teste vamos validar um código postal português, sem a localidade:

```
  public class ClassTobeValidated
  {
  [RegularExpresionValidator(@"^\d\d\d\d-\d\d\d$")]
    public Object field{ get; set; }
  }
```

Como sabemos, um código postal em Portugal é composto por 4 dígitos mais 3, separados por um hífen. Se tentarmos validar um objecto com vários valores temos o seguinte resultado:

```
static void Main(string[] args)
{

  ClassTobeValidated classe =
                    new ClassTobeValidated();

  Console.WriteLine("Is class valid (" +
    classe.field+")? " +  Validator.checkIfIsValid
    (classe).ToString());
  classe.field = "texto";
  Console.WriteLine("Is class valid (" +
    classe.field +")? " + Validator.checkIfIsValid
    (classe).ToString());
  classe.field = "12";
  Console.WriteLine("Is class valid (" +
    classe.field + ")? " + Validator.checkIfIsValid
    (classe).ToString());
  classe.field = "1234";
  Console.WriteLine("Is class valid (" +
    classe.field + ")? " + Validator.checkIfIsValid
    (classe).ToString());
  classe.field = "2s34-awd";
  Console.WriteLine("Is class valid (" +
    classe.field + ")? " + Validator.checkIfIsValid
    (classe).ToString());
  classe.field = " 1234-123";
  Console.WriteLine("Is class valid (" +
     classe.field +")? " + Validator.checkIfIsValid
     (classe).ToString());
  classe.field = "1234-123 ";
  Console.WriteLine("Is class valid (" +
     classe.field +")? " + Validator.checkIfIsValid
     (classe).ToString());
  classe.field = "1234-123";
  Console.WriteLine("Is class valid (" +
    classe.field + ")? " + Validator.checkIfIsValid
    (classe).ToString());

 WaitForKeyPressAndExit();

}
```

```
Is class valid ()? False
Is class valid (texto)? False
Is class valid (12)? False
Is class valid (1234)? False
Is class valid (2s34-awd)? False
Is class valid ( 1234-123)? False
Is class valid (1234-123 )? False
Is class valid (1234-123)? True
Press any key to continue.
```

Atributos em C#

Agora que temos um validador mais versátil vamos tentar usar um caso que possa ser usado no mundo real. Será que conseguimos usar estes objectos para criarmos um objecto Pessoa e garantirmos que os inputs de um utilizador estão correctos?

Vamos então criar um objecto Pessoa com os validadores necessários. Neste caso vamos usar apenas expressões regulares mas podíamos usar qualquer outra que fosse necessária para algum caso mais particular:

```
public class Pessoa
{
 [RegularExpresionValidator(@"^\d{8}$")]
 public String BI { get; set; }

 [RegularExpresionValidator(@"(\d+)/(\d+)/(\d{4})")]
 public String DataNascimento { get; set; }

 [RegularExpresionValidator(@"^([0-9a-zA-Z]([-.\w]*
[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA
-Z]{2,9})$")]
 public String Email { get; set; }

 [RegularExpresionValidator(@"[A-Z]{1}[a-z\.]+")]
 public String Nome { get; set; }

}
```

Agora que temos um sítio onde guardar a informação podemos pedir ao utilizador que a insira para podermos fazer a validação:

```
static void Main(string[] args)
  {
    Pessoa p = new Pessoa();
    Boolean valid;
    do
    {
      Console.Write("Nome: ");
      p.Nome = Console.ReadLine();
      Console.Write("BI: ");
      p.BI = Console.ReadLine();
      Console.Write(
            "Data de Nascimento (dd/mm/yyyy): ");
      p.DataNascimento = Console.ReadLine();
      Console.Write("Email: ");
      p.Email = Console.ReadLine();
      valid = Validator.checkIfIsValid(p);
      Console.WriteLine("Os dados desta pessoa são
válidos: " + valid);
    }while(!valid);

    WaitForKeyPressAndExit();
```

O resultado será alguma coisa semelhante a isto:

```
Nome: manel
BI: 123
Data de Nascimento (dd/mm/yyyy): 2/Mar/79
Email: manel£mail,com
Os dados desta pessoa sao válidos: False
Nome: Manuel
BI: 12345678
Data de Nascimento (dd/mm/yyyy): 2/3/1979
Email: manel@mail.com
Os dados desta pessoa sao válidos: True
Press any key to continue.
```

Fica ao cargo do leitor perceber qual, ou quais, os motivos que fizeram a primeira tentativa falhar.

Conclusão

Os atributos são um recurso extremamente útil para evitar que tenhamos de repetir muitas vezes o mesmo código. Podendo ser usado em classes, propriedades, métodos, etc, ficam associadas ao elemento onde são utilizadas podendo ser facilmente acedidas. Podem ser usadas tanto por motivos informativos, bem como para ter um determinado comportamento, o que permite abstrair da implementação e focar apenas na sua funcionalidade.

Referências

[1] - MSDN (ms-help://MS.MSDNQTR.2002APR.1033/csspec/html/vclrfcsharpspec_17_2.htm)

http://www.radsoftware.com.au/articles/regexlearnsyntax.aspx

http://oreilly.com/windows/archive/csharp-regular-expressions.html

AUTOR

Escrito por Flávio Geraldes
Licenciou-se em Engenharia Informática e Computadores no Instituto Superior Técnico tendo-se especializado na área de Programação e Sistemas de Informação. Após a finalização do curso juntou-se à Sybase SBS Software onde teve oportunidade de trabalhar com várias tecnologias focando-se particularmente em .NET. Actualmente é consultor da Sybase SBS Software numa empresa de telecomunicações onde é responsável pelo desenho e desenvolvimento de várias aplicações.

COLUNAS

VISUAL (NOT) BASIC – Tipos Genéricos

Tipos Genéricos

Introdução

Se nunca usou tipos genéricos, provavelmente utiliza um array para guardar dados/objectos, no entanto a utilização desta estrutura tem algumas desvantagens relativamente aos genéricos, como o tamanho fixo, ao contrário dos tipos genéricos que podem ser expandidos de forma dinâmica, a falta de métodos especializados para trabalhar com objectos, são strongly typed não permitindo colocar valores de tipos diferentes, etc.

Com este artigo, pretendo através da utilização uma classe "Automovel" e através da elaboração de Console Applications e de várias analogias mostrar como poderá utilizar alguns dos tipos genéricos da linguagem Visual Basic 2010, da forma mais simples e acessível possível.

Passando desde já à "acção", deverá então criar a classe base sobre a qual iremos criar os objectos para os diferentes tipos genéricos que iremos utilizar neste artigo.

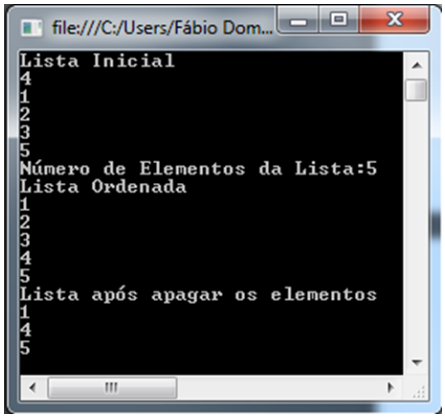
```
Public Class Automovel

    Public Property Cilindrada() As Integer
    Public Property Marca() As String
    Public Property Modelo() As String

    Public Sub New()
      'Permite não definir valores no construtor
    End Sub

    Public Sub New(ByVal cilindrada As Integer,
                   ByVal marca As String,
                   ByVal modelo As String)
        Me.Cilindrada = cilindrada
        Me.Marca = marca
        Me.Modelo = modelo

    End Sub

    Public Overrides Function ToString() As String

         Return String.Format(
              "{0} {1} com {2} cc de cilindrada",
              Marca, Modelo, Cilindrada)

    End Function

End Class
```

List(Of T)

A lista, é provavelmente o tipo mais básico que podemos aplicar, pois é apenas… uma lista. Existem vários métodos que podemos aplicar à lista, no exemplo seguinte, iremos ver alguns:

```
Sub ListaNumeros()
  Dim lista As New List(Of Integer) From
      {4, 1, 2, 3}

  'Adiciona um novo elemento à lista
  lista.Add(5)

  'Mostra o conteúdo da lista
  Console.WriteLine("Lista Inicial")

  For Each a As Integer In lista
    Console.WriteLine(a.ToString())
  Next

  'Escreve o número de elementos da lista
  Console.WriteLine(vbLf &
              "Numero de Elementos da Lista")

  Console.WriteLine(lista.Count)

  'Ordenar a lista
  lista.Sort()

  'Mostra a lista ordenada
  Console.WriteLine(vbLf & "Lista Ordenada" &
                                        vbLf)

  For Each a As Integer In lista
    Console.WriteLine(a.ToString())
  Next

  'Remove a primeira instância do item indicado
  'para apagar que se encontra na 3ª posição
  lista.Remove(2)

  'Remove o número numa determinada posição
  'para apagar o valor que se encontra no
  'índice 1 (2ª casa do array)
  lista.RemoveAt(1)
  Console.WriteLine(vbLf &
       "Lista após apagar os elementos" & vbLf)
  For Each a As Integer In lista
    Console.WriteLine(a.ToString())
  Next

  Console.Read()
End Sub
```

O output obtido é o seguinte:



No início, temos a lista com os elementos pela ordem que foram inseridos através das seguintes instruções:

```
Dim lista As New List(Of Integer) From
                                {4, 1, 2, 3}
lista.Add(5)
```

Através da primeira instrução, criámos a lista e inicializámo-la com os valores inteiros 4, 1, 2 e 3, e de seguida com a segunda linha, acrescentámos mais um valor, o valor 5. Neste momento, podemos esquematizar a lista da seguinte forma:

Posição	0	1	2	3	4
Elemento	4	1	2	3	5

É importante compreender que o primeiro elemento da lista, encontra-se na posição 0, e não na 1. Este é um erro comum e neste momento qualquer tentativa de referência à posição 5 por exemplo, resultará no lançamento de uma excepção e se tentar apagar o elemento "4" enviando como parâmetro o índice (posição) 1, irá apagar o elemento 1.

Depois tempoa seguinte linha de código:

```
For Each a As Integer In lista
     Console.WriteLine(a.ToString())
Next
```

Este é o iterador For Each, que tem como objectivo percorrer uma estrutura de dados (array ou genérico), sem ser necessário indicar o tamanho da mesma, foi também declarada uma variável "a" do tipo dos elementos da lista (inteiro neste caso), para que possa guardar o valor do elemento que se encontra na posição actual.

Mais à frente, fizemos uma ordenação à lista, através da seguinte instrução:

```
lista.Sort()
```

Uma nota muito importante, é que neste caso, como estamos a trabalhar com a classe Integer, a ordenação é bastante simples e lógica (ordem crescente).

Lista com Objectos

Agora, vamos criar uma lista de objectos, utilizando para isso a classe **Automovel** anteriormente definida

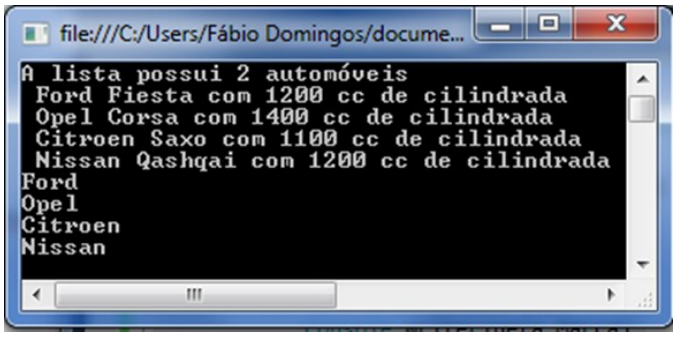
```
Sub Lista()

    'Criar a lista e inserir alguns automóveis
    Dim automoveis As New List(Of Automovel) From
        {
          New Automovel() With
              {.Marca = "Ford",
               .Modelo = "Fiesta",
               .Cilindrada = 1200},
          New Automovel() With
              {.Marca = "Opel",
               .Modelo = "Corsa",
               .Cilindrada = 1400}
         }

    'Mostra o total de carros na lista
    Console.WriteLine(
                  "A lista possui {0} automóveis",
                  automoveis.Count)

    'Adiciona um novo automóvel no final da lista
    automoveis.Insert(automoveis.Count,
                      New Automovel() With
                      {
                       .Marca = "Citroen",
                       .Modelo = "Saxo",
                       .Cilindrada = 1100
                      }
                     )

    'Insere o objecto na última posição
    automoveis.Add(New Automovel() With
                     {
                       .Marca = "Nissan",
                       .Modelo = "Qashqai",
                       .Cilindrada = 1200
                     }
                   )

    'Mostra o conteúdo da lista
    For Each a In automoveis
      Console.WriteLine(a.ToString())
    Next

    'Mostra apenas a marca
    For Each a In automoveis
      Console.WriteLine(a.Marca)
    Next

    Console.Read()
End Sub
```

Tipos Genéricos

Vamos então ver, um pouco mais em detalhe, alguns métodos que utilizámos:

- **Count()** - Retorna o número de objectos que a lista possui
- **Insert(Indice As Integer, Objecto)** - Adiciona o objecto especificado no segundo parâmetro na posição indicada pelo índice, para adicionar ao fim da lista, podemos utilizar automoveis.count
- **Add(Objecto)** - Adiciona o objecto no final da lista

O output da aplicação na consola é o seguinte:



Na primeira linha, podemos ver o resultado de ter corrido a instrução “nomedalista.Count”, que retorna o número de elementos que possui, neste caso retornou 2, porque era o número de “automóveis” que a lista possuía aquando da sua inicialização.

Depois foram inseridos mais dois automóveis, e através do iterador *For Each*, percorreu-se a lista, gerando as 4 próximas linhas, que contêm o retorno do método *toString* dos objectos, mostrando assim informações sobre os mesmos.

Finalmente temos as marcas dos automóveis, resultado de termos feito um *Console.Writeline* às várias posições do array.

Ordenar a Lista

Se desejarmos ordenar a lista, poderemos usar o método Sort e como parâmetro enviar uma classe que implemente a interface iComparer.

Para isso, e para que possamos ordenar os automóveis por cilindrada, vamos então criar uma classe “OrdenarPorCilindrada”, para que possamos utilizar como parâmetro do Sort, e desta forma ordenar os objectos:

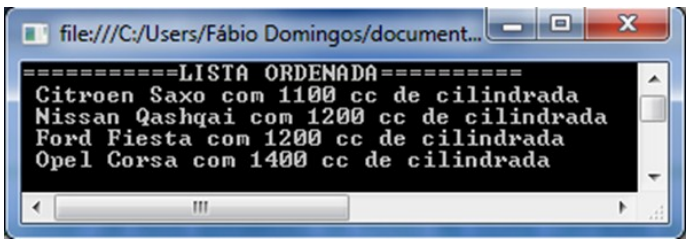
```
Public Class OrdenarporCilindrada
    Implements IComparer(Of Automovel)


     Public Function Compare(ByVal x As Automovel,
ByVal y As Automovel) As Integer Implements Sys-
tem.Collections.Generic.IComparer(Of Automo-
vel).Compare

        If x.Cilindrada > y.Cilindrada Then
            Return 1
        End If

        If x.Cilindrada < y.Cilindrada Then
            Return -1
        Else
            Return 0
        End If
    End Function


End Class
```

Se não entendeu o código da função Compare, não fique preocupado! Estamos apenas a informar a interface como é que queremos que a comparação seja feita. Se retornarmos 0, isso significa que ambos os objectos são iguais, no caso de retornarmos 1 ou -1, isso significa que o primeiro objecto é maior ou menor que o segundo, respectivamente.

Seguidamente, no *Sub* em que utilizamos a lista, devemos acrescentar o seguinte código:

```
    'Ordenar os objectos por cilindrada
    Automoveis.Sort(New OrdenarporCilindrada)


    Console.WriteLine(
            "==========LISTA ORDENADA==========")

    'Mostra o conteúdo da lista
    For Each a In Automoveis
      Console.WriteLine(a.ToString())
    Next


    Console.Read()
```

Teremos então como output, a lista ordenada por cilindrada:



Poderemos também, utilizar a interface IComparable, que ao contrário da IComparer que se define numa classe à parte e é depois enviada como parâmetro da função Sort, a IComparable é definida directamente na própria classe dos objectos da lista (neste caso a classe automóvel), um exemplo de implementação é o seguinte:

```
Public Class Automovel
    Implements IComparable

    Public Property Cilindrada() As Integer
    Public Property Marca() As String
    Public Property Modelo() As String

    Public Sub New()
      'Permite não definir valores no construtor
    End Sub

    Public Sub New(ByVal cilindrada As Integer,
     ByVal marca As String,ByVal modelo As String)
      Me.Cilindrada = cilindrada
      Me.Marca = marca
      Me.Modelo = modelo
    End Sub

    Public Overrides Function ToString() As String
      Return String.Format(
            " {0} {1} com {2} cc de cilindrada",
            Marca, Modelo, Cilindrada)
    End Function

    Public Function CompareTo(ByVal obj As Object)
As Integer Implements System.IComparable.CompareTo
      Dim temp As Automovel =
                        TryCast(obj, Automovel)

      If temp IsNot Nothing Then

       If Me.Cilindrada > temp.Cilindrada Then
          Return 1
       ElseIf Me.Cilindrada < temp.Cilindrada Then
          Return -1
       Else
          Return 0
       End If

      Else
        Throw New ArgumentException(
            "O parâmetro não é um automóvel")
      End If

    End Function
  End Class
```

Basicamente, adicionámos o Inherits IComparable para implementar a classe, e depois definimos a função CompareTo, para explicitar a forma como queremos que os objectos da classe sejam ordenados, neste caso e como, mais uma vez, queremos que sejam ordenados por cilindrada, a implementação da função CompareTo é bastante semelhante à compare que utilizámos na interface IComparer, com a diferença que em vez de variáveis como a x e a y, utilizámos o próprio objecto (Me) e o objecto a comparar, para a qual definimos uma variável temp. Para fazer a ordenação da lista, basta executar a seguinte instrução:

```
automoveis.Sort()
```

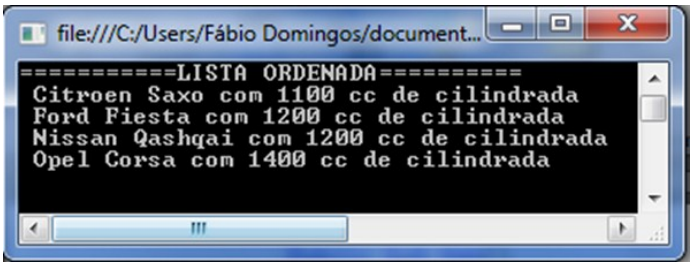
Esta instrução funciona, porque o mecanismo de ordenação está definido na própria classe.

Ordenação Alfabética

Se desejarmos, também podemos ordenar alfabeticamente a lista, neste caso, poderíamos implementar o Compare da interface IComparer, criando uma classe “OrdenarporMarca” com o seguinte código de modo a ordenar por marca:

```
Public Class OrdenarporMarca
    Implements IComparer(Of Automovel)

    Public Function Compare(ByVal x As Automovel,
ByVal y As Automovel) As Integer Implements Sys-
tem.Collections.Generic.IComparer(Of Au-
tomovel).Compare
        If x.Marca > y.Marca Then
            Return 1
        End If

        If x.Marca < y.Marca Then
            Return -1
        Else
            Return 0
        End If
    End Function
End Class
```

Poderá então concluir, que o processo de ordenação por ordem alfabética, é bastante semelhante ao dos números. Após correr o código, terá então o output desejado:



A Classe Stack(Of T)

Uma stack, em português pilha, é uma estrutura LIFO (Last in First Out), em que o último objecto colocado, é o primeiro a sair. Um exemplo da vida real, é uma série de livros colocados uns em cima dos outros, o primeiro a entrar, fica em baixo, logo se o formos retirar da pilha, este será o último, pois teremos de remover primeiro todos os que estão em cima deste, ou seja, os últimos que foram colocados, ficam em cima da pilha, e desta forma são os primeiros a sair.

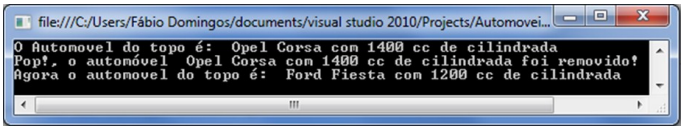
Tipos Genéricos

Esta classe, usa os métodos Push() e Pop() para carregar itens para cima da pilha ou removê-los respectivamente, poderá utilizar também o método Peek(), para visualizar o objecto do "topo", sem o remover.

Um exemplo de utilização desta estrutura é o seguinte

```
Sub usarStack()
  Dim stackAutomoveis As New Stack(Of Automovel)

  'Adicionar os Carros
  stackAutomoveis.Push(New Automovel With {
    .Marca = "Ford", .Modelo = "Fiesta",
    .Cilindrada = 1200})
  stackAutomoveis.Push(New Automovel With {
    .Marca = "Opel",.Modelo = "Corsa",
    .Cilindrada = 1400})

  'Mostra o item do topo da pilha sem o remover
  Console.WriteLine("O Automovel do topo é: {0}",
    stackAutomoveis.Peek())

  'Remove o automóvel do topo da pilha
  Console.WriteLine(
      "Pop!, o automóvel {0} foi removido!",
      stackAutomoveis.Pop())

  'Fazendo o peek de novo
  Console.WriteLine(
      "Agora o automovel do topo é: {0}",
       stackAutomoveis.Peek())

  Console.Read()

End Sub
```

A consola irá então mostrar o seguinte:



Na primeira linha, vemos o último objecto que adicionámos, que é o Opel Corsa, no entanto este não foi removido, pois foi utilizado o método peek() que apenas permite ver qual é o objecto. Seguidamente, usámos o método pop, que para além de remover o objecto, retorna o mesmo e por último, vemos o primeiro objecto que adicionámos, o Ford Fiesta.

Na próxima estrutura, a queue, vamos ver como se pode mostrar os objectos pela ordem em que os adicionámos.

A Classe Queue(Of T)

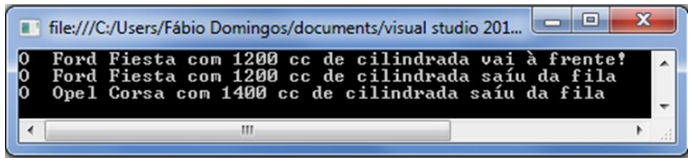
Quando estamos na fila de um supermercado, o primeiro a chegar, é o primeiro a ser atendido e consequentemente, o último a chegar, é o último a ser atendido. Este é o funcionamento da Queue (fila em português), que em vez de LIFO, funciona como FIFO (First In First Out), ou seja, o primeiro a entrar é o primeiro a sair.

Os principais métodos desta classe são o Dequeue(), que remove e retorna o objecto que se encontra na frente da fila (o primeiro que adicionámos), o Enqueue() que adiciona um objecto para a parte de trás da fila (na vida real, é o que acontece quando entramos numa fila, ficamos na última posição), e finalmente, o bem conhecido do Stack, o Peek() que retorna o primeiro objecto que adicionámos sem o apagar.

Passemos então à acção, e vamos pôr a fila a "trabalhar":

```
Sub Fila()
        'Adicionar 2 automóveis à fila
        Dim filadeAutomoveis As New Queue _
            (Of Automovel)()

        filadeAutomoveis.Enqueue(
            New Automovel With {
             .Marca = "Ford",
             .Modelo = "Fiesta",
             .Cilindrada = 1200})
        filadeAutomoveis.Enqueue(
            New Automovel With {
             .Marca = "Opel",
             .Modelo = "Corsa",
             .Cilindrada = 1400})
        'Temos então uma fila de 2 automóveis,
        'o Fiesta, encontra-se à frente do Corsa

        'Verifica, sem remover,
```

Correndo o código, obtemos o seguinte output:



Para este código, imaginemos que o Ford Fiesta e o Opel Corsa encontram-se numa fila para uma bomba de gasolina, o Ford Fiesta vai à frente, pois foi o primeiro a ser adicionado.

Primeiro, através do peek, vimos qual era o automóvel que ia à frente, depois com o dequeue, fizemos com que o objecto fosse retornado e removido da lista, ou seja, o automóvel abasteceu e saiu da bomba. Seguidamente, o Corsa que estava atrás do Fiesta, passa para a frente da fila, é abastecido e também sai da mesma.

A Classe Dictionary(Of TKey, TValue)

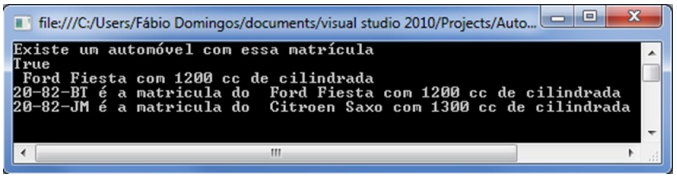
O dicionário é um genérico que possui uma sintaxe um pouco diferente dos outros, no entanto a forma de compreender, não é difícil. Todos nós, temos vários números que nos identificam, como o número do bilhete de identidade ou cartão do cidadão, o número do cartão de contribuinte e, no caso dos automóveis, o número da chapa de matrícula, que iremos usar para este exemplo, a este valor, em .net, chamamos de "Chave" (Key), enquanto que o valor dessa referencia, como os nossos dados no caso do BI, ou os dados do automóvel no caso da chapa de matrícula, chamamos isso mesmo, valor (value). Quando adicionamos um objecto a uma colecção do tipo Dictionary, temos então de indicar para além do objecto que vamos adicionar, a chave que o vai referenciar, a sintaxe é a seguinte:

```
Nomedodicionario.Add(123456789,"João Silva")
```

Desta forma, adicionámos o objecto, neste caso a String "João Silva", com o identificador 123456789. Mas como é de automóveis que temos falado, vamos então ver o exemplo:

```
Sub Dicionario()
    Dim setAutomoveis As New Dictionary _
    (Of String, Automovel)

    setAutomoveis.Add("20-82-BT",
       New Automovel With {.Marca = "Ford",
       .Modelo = "Fiesta", .Cilindrada = 1200})
    setAutomoveis.Add("20-82-JM",
       New Automovel With {.Marca = "Citroen",
       .Modelo = "Saxo",.Cilindrada = 1300})

    'Verifica se o objecto com uma
    'determinada chave existe
    If setAutomoveis.ContainsKey("20-82-BT") Then
        Console.WriteLine("Existe")
    Else
        Console.WriteLine("Não Existe")
    End If

    Console.WriteLine _
    (setAutomoveis.ContainsKey("20-82-BT"))

    'Escreve o objecto com a chave("20-82-BT")
    Console.WriteLine _
    (setAutomoveis.Item("20-82-BT"))

    'Mostra os objectos do dicionário e chaves
    Dim par As KeyValuePair _
        (Of String, Automovel)
    For Each par In setAutomoveis
        Console.WriteLine(
            "{0} é a matricula do {1}",
            par.Key, par.Value)
    Next

    Console.Read()
End Sub
```

O output obtido é o seguinte:



Vamos agora analisar o output linha a linha.

Na 1º linha, aparece a palavra "True", isto deve-se ao facto de termos invocado a função "ContainsKey", esta função recebe como parâmetro uma chave, e quando executada, verifica se existe na colecção um objecto com essa chave, retornando um valor booleano (true caso exista, false se não existir). Neste caso como retornou true, existe um automóvel com essa matrícula, poderíamos também formatar a resposta através de uma condição deste género:

```
If setAutomoveis.ContainsKey("20-82-BT") Then
    Console.WriteLine("Existe um automóvel com essa matrícula")
Else
```

Desta forma, se o objecto existir irá ser escrita uma mensagem bastante mais amigável para o utilizador.

Passando para a segunda linha, aqui utilizámos a função ContainsKey, que recebeu como parâmetro a matrícula (chave) do automóvel (objecto) e escreveu a descrição deste, utilizando para isso a função ToString definida na sua classe (Automóvel no caso deste exemplo).

De seguida, as duas últimas linhas são o resultado de, utilizando o KeyValuePair que define um par valor-chave do tipo String, para que possa ser depois escrito através do iterador For Each utilizado na linha seguinte que irá percorrer a colecção e, recorrendo ao método ToString, escrever a descrição dos objectos.

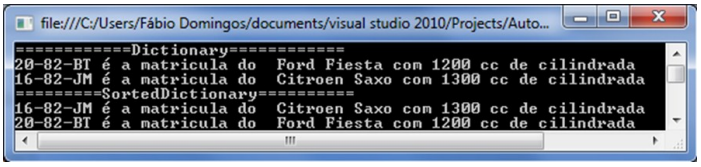
A Classe SortedDictionary(Of T)

Este genérico é bastante semelhante ao anterior, no entanto aqui os objectos serão ordenados por ordem crescente chave. Por isso, aqui irei apenas apresentar um pequeno exemplo para que compreendam a diferença entre estes dois genéricos, para isso iremos adicionar dois objectos a cada uma das colecções, e depois correr um iterador, tal como exemplificado no exemplo anterior de modo a percorrer ambas as colecções.

Tipos Genéricos

```
Sub SortedDictionary()
     Dim setAutomoveisOrd As New  _
     SortedDictionary(Of String, Automovel)()
     Dim setAutomoveis As New  _
     Dictionary(Of String, Automovel)

     setAutomoveisOrd.Add(
         "20-82-BT", New Automovel With
         {.Marca = "Ford",
          .Modelo = "Fiesta",
          .Cilindrada = 1200})
     setAutomoveisOrd.Add(
         "16-82-JM", New Automovel With
         {.Marca = "Citroen",
          .Modelo = "Saxo",
          .Cilindrada = 1300})
     setAutomoveis.Add(
         "20-82-BT", New Automovel With
         {.Marca = "Ford",
          .Modelo = "Fiesta",
          .Cilindrada = 1200})
     setAutomoveis.Add(
         "16-82-JM", New Automovel With
         {.Marca = "Citroen",
          .Modelo = "Saxo",
          .Cilindrada = 1300})

     Console.WriteLine("=Dictionary=")

     Dim par As KeyValuePair(Of String,Automovel)
     For Each par In setAutomoveis
         Console.WriteLine(
             "{0} é a matricula do {1}",
             par.Key, par.Value)
     Next

     Console.WriteLine("=SortedDictionary=")

     Dim par2 As KeyValuePair(Of String,Automovel)

     For Each par2 In setAutomoveisOrd
         Console.WriteLine(
            "{0} é a matricula do {1}",
               par2.Key, par2.Value)
     Next

     Console.Read()
End Sub
```

Após correr este código, iremos então obter o seguinte output:



De modo a percebermos melhor o que aconteceu aqui, repararemos na ordem em que os objectos foram inseridos, primeiro um com matrícula (chave) iniciada por 20 e de seguinte outro com matrícula iniciada por 16, estando assim desordenados.

Quando corremos o iterador sobre o dicionário, estes foram impressos na ordem em que foram inseridos, enquanto que no caso do SortedDictionary, estes foram impressos por ordem crescente de chave (matrícula). É esta a diferença entre estes dois genéricos.

Conclusão

Depois de ler este artigo, e se nunca utilizou estas estruturas nos seus programas, espero que agora sejam uma espécie de "amigos" inseparáveis e os utilize nos seus próximos programas que necessitem de guardar objectos, pois com isso vai ter imensas vantagens.

Neste artigo, foram abordados apenas alguns dos inúmeros métodos que poderá aplicar a cada um dos genéricos, poderá ver mais consultando a secção do namespace System.Collections.Generic que se encontra no website http://msdn.microsoft.com/en-us/library/0sbxh9x2.aspx . Neste endereço, poderá para além das funções, ver também um conjunto de interfaces úteis que poderá aplicar nas suas colecções, para além da iComparer que foi utilizada no exemplo do SortedSet.

Apesar de existirem mais tipos genéricos, penso que os básicos e mais utilizados foram abordados.

AUTOR

Escrito por **Fábio Domingos**

Estudante de Gestão de Sistemas de Informação na Escola Superior de Ciências Empresariais do Instituto Politécnico de Setúbal, tem como principal hobbie a informática, nomeadamente as áreas de programação, bases de dados e IT Support. É moderador global do fórum Portugal-a-Programar, membro do staff desta revista e por vezes contribui com soluções para a comunidade Experts-Exchange.

COMUNIDADES

AzurePT – Windows Azure Traffic Manager

SQLPort – Ferramentas gratuitas de trabalho com SQL Server

NetPonto – NHibernate - Do Import-Package à primeira iteração

SharepointPT – Sandboxed Solutions em SharePoint Online 2010

Windows Azure Traffic Manager

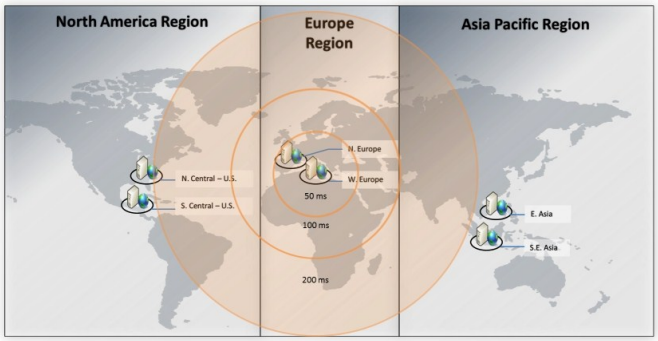
Introdução

Ultimamente tem-se falado muito sobre o Cloud Computing e sobre as diversas ofertas que se encontram disponíveis, e ainda mais sobre como esta nova forma de computação permite que os nossos negócios passem a ser cada vez mais globais. Com isso conseguimos atingir novos mercados, e novos utilizadores. Sem dúvida que todas estas questões são verdadeiras, pois uma vez na Internet as nossas soluções passam imediatamente a dispor de uma presença global e utilizável por qualquer utilizador de qualquer local do mundo, mas a questão que muitas vezes se coloca é, a que preço e como que qualidade estão esses utilizadores a ser servidos? Sem dúvida que esta é uma questão importante, e que pode tornar as nossas soluções mais ou menos conhecidas.

Porque nos devemos preocupar?

Estas são preocupações que deveremos ter seriamente em consideração quando disponibilizamos uma solução no mercado gobal, uma vez que a Experiência de Utilização (UX) é um dos pontos mais importantes para que a nossa solução seja mais ou menos utilizada, e um dos pontos importantes na UX é sem dúvida a performance da mesma, uma vez sabemos que os utilizadores não gostam de esperar demasiado tempo por uma página, ou por fazer uma determinada operação, pois é completamente diferente uma solução que responde num período de 1 a 3 segundos, de uma que responde entre 10 a 30 segundos, pois a produtividade vai decrescer significativamente, e esse aspecto será tido em conta no momento da aquisição ou não dessa mesma solução. Por isso mesmo um ponto muito importante a reter é que “A PERFORMANCE é muito Importante”.

Passemos então a ilustrar estes aspectos olhando para o mapa da dispersão dos Data Centers de Windows Azure da Microsoft , e analisando como a performance será afectada com base nas decisões que efectuamos sobre em que Data Center disponibilizar a solução.



6 Datacenters em 3 continentes

Figura 1 – Disponibilização da solução no Data Center de Western Europe.

Com base nesta figura vamos analisar o que acontecerá no caso de disponibilizar-mos a nossa solução no Data Center de Windows Azure identificado como Western Europe. O que este gráfico indica é que a região mais próxima irá obter tempos de latência de cerca de 50 milissegundos, o que significa que terão uma performance bastante boa no acesso à nossa solução, mas à medida que nos afastamos do local em que disponibilizamos a nossa solução o tempo de latência vai duplicando, para 100 ms, 200ms, e assim sucessivamente. O que significa que neste caso se estivermos a tentar abordar o mercado dos Estados Unidos teremos certamente questões de performance bastante importantes que deveremos ter em conta.

Também importante no momento da escolha do Data Center a utilizar será o site http://latency.cloudapp.net ou o http://metricstest.cloudapp.net/ que são dois sites de estatísticas e gráficos de tempos de latência produzido por Matthew Rosoff. Aqui poderemos o tempo de latência médio entre zonas.

From\To (ms)	North-central US	South-central US	North Europe	West Europe	East Asia	South-East Asia
North-central US	3	36.2	104.3	105	220.3	236.8
South-central US	40	2.8	120.5	118.3	201.4	220.3
North Europe	101.6	110.7	3.1	22.8	310.5	324.6
West Europe	112.2	117.8	22.5	2.8	320.3	335.5
East Asia	224.4	201.9	311.4	322.3	2.3	32.8
South-East Asia	242.8	224.6	325.8	336.1	33.3	3.7

Figura 2 – Tempos de Latência entre Zonas

Mas se explorarmos um pouco mais o site conseguimos ter informação sobre o histórico dos tempos de latência entre dois Data Centers, como por exemplo Western Europe e North-Central US.

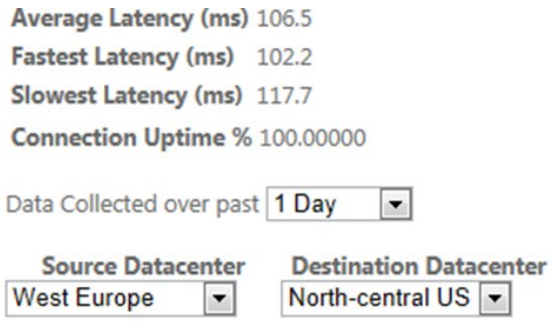


Figura 3 – Histórico do Tempo de latência entre Western Europe e North-Central US para as últimas 24 horas

Com estes dados conseguimos compreender que o tempo de latência médio das últimas 24 horas foi de 106.5 ms, e que o mais lento foi de 117.7 ms, o que perfaz uma performance interessante dada a dispersão geográfica. Também com todas estas informações torna-se mais simples compreender a Experiência de Utilização (UX) que os utilizadores finais irão obter.

O Problema

Na base de todas estas questões está uma bastante simples de compreender, que é, "Quanto melhor a Experiência de Utilização, mais €€€€", e é esta a razão pela qual nós todos lutamos. Mas para que isto seja uma realidade quais as soluções que podermos utilizar?

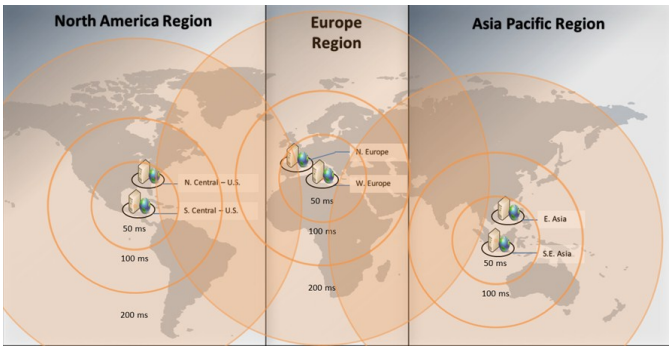


Figura 4 – Dispersão geográfica da solução

Provavelmente a resolução mais óbvia para este problema seria a de disponibilizar a nossa solução nos diversos Data Centers disponíveis e a partir daí, efectuar o redireccionamento dos clientes para cada um dos Data Centers dependendo da sua localização. Sendo uma solução mais ou menos óbvia levanta alguns problemas, como por exemplo:

- Como será efectuado o redireccionamento dos pedidos?
- Como deverei efectuar o balanceamento dos pedidos?
- Como deverei redireccionar os pedidos em caso de problemas num dos Data Centers?
- Como resolver o problema do endereço, visto que cada uma das disponibilizações em cada Data Center tem um endereço único?

Estas são sem dúvida questões complexas que necessitam de resposta de forma a que possamos ter uma solução efectiva para o problema.

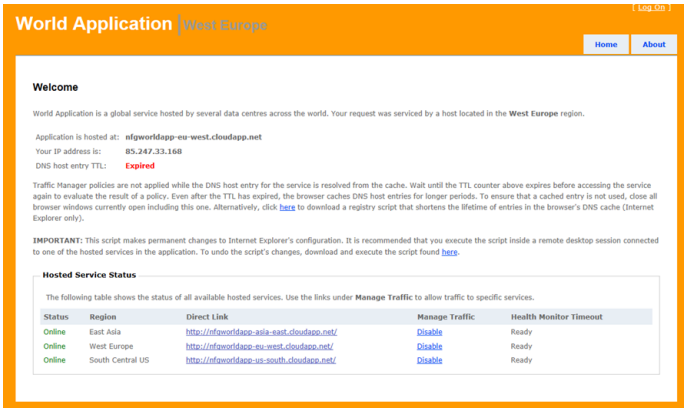
Para demonstrar esta solução iremos olhar para o Laboratório presente no Windows Azure Traning Kit em http://bitly.com/WATrainingKit, ou que poderemos obter o código fonte aqui http://bit.ly/WATrafficManagerLabSource e o guia de acompanhamento do laboratório aqui http://bit.ly/WATrafficManagerLabGuide.

Abrindo a solução de Visual Studio 2010 denominada WorldApp que se encontra dentro do ficheiro Source.Zip que fizemos download, e iremos ter todo o código-fonte necessário para efectuar os testes ao Windows Azure Traffic Manager. A solução é composta por 2 projectos denominados WorldApp, projecto de ASP.NET MVC que representa a solução que queremos disponibilizar globalmente, e o WorldAppService que representa o Serviço que iremos disponibilizar no Windows Azure.

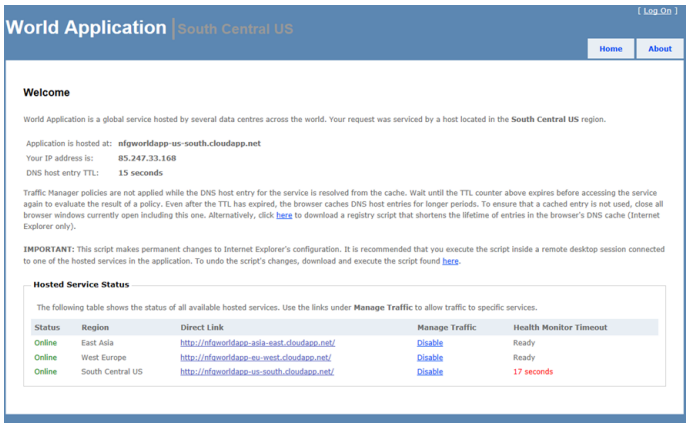
Os passos utilizados para efectuar a publicação da solução para os diversos Data Centers encontra-se disponível no guia de acompanhamento do laboratório.

Após a publicação passamos a ter a solução a correr nos 3 (três) continentes, ou seja, publicamos a solução para o Centro Sul dos Estados Unidos, Oeste da Europa e Este da Ásia. Cada uma das soluções têm configurações diferentes, nomeadamente em termos visuais para que consigamos idenficá-las mais facilmente. Mas para acedermos a cada uma delas temos que colocar o url relativo ao ambiente específico, que neste caso são:

http://nfgworldapp-eu-west.cloudapp.net/ - Oeste da Europa

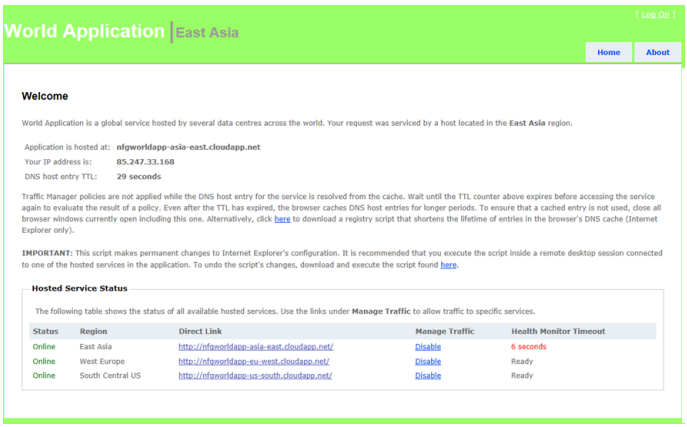


http://nfgworldapp-us-south.cloudapp.net/ - Centro Sul dos Estados Unidos



http://nfgworldapp-asia-east.cloudapp.net/ - Este da Ásia

Windows Azure Traffic Manager



Agora que já temos as soluções a funcionar nos 3 Data Centers embora os problemas mantêm-se.

- Como será efectuado o redireccionamento dos pedidos?
- Como deverei efectuar o balanceamento dos pedidos?
- Como deverei redireccionar os pedidos em caso de problemas num dos Data Centers?
- Como resolver o problema do endereço, visto que cada uma das disponibilizações em cada Data Center tem um endereço único?

Windows Azure Traffic Manager

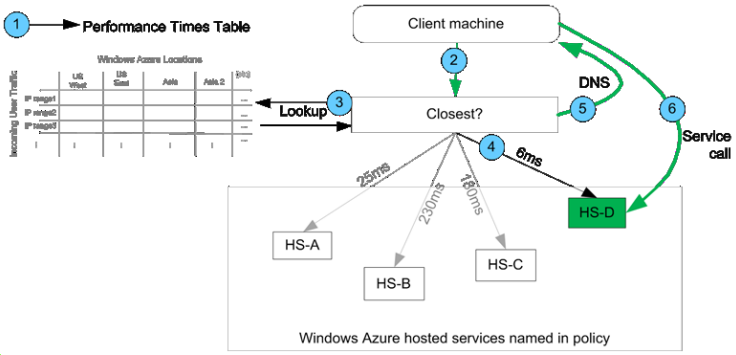
Tendo estes problemas em consideração a equipa do Windows Azure da Microsoft decidiu criar um produto que os conseguisse solucionar de forma rápida e simples, e esse produto chama-se Windows Azure Traffic Manager. Este produto encontra-se actualmente em CTP mas deverá ser lançada a versão final dentro de muito pouco tempo.

A solução apresentada é pela equipa com o Windows Azure Traffic Manager foi a disponibilização de um Load Balancer Global sobre o qual poderemos criar políticas que irão definir como esse balanceamento de tráfego será efectuado. Para isso existem neste momento 3 opções que poderemos utilizar, que são Performance, Round Robin e Failover. Vamos então olhar para cada uma delas em maior detalhe.

Performance

Balanceamento com base no Data Center "melhor"/"mais próximo" (de acordo com os cálculos que são efectuados em termos de tempos de latência, conforme apresentado nos endereços apresentados anteriormente). Um exemplo disto será, um utilizador de Portugal será redireccionado para a versão disponível no Oeste da Europa, e não para os restantes, uma vez que é o que se encontra mais próximo em termos de tempo de latência.

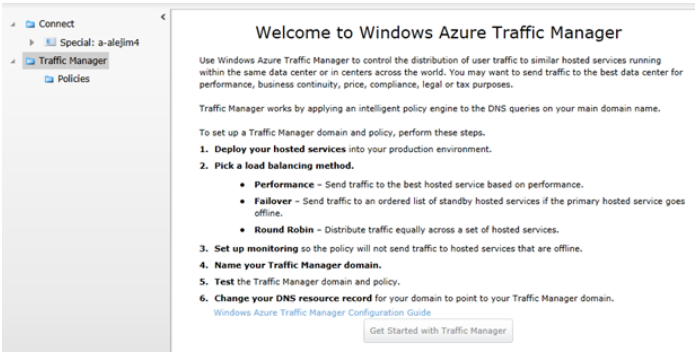
A Teoria



1. Internamente é preenchida a tabela de tempos de latência entre Data Centers
2. Quando o cliente efectua um pedido para o endereço do Windows Azure Traffic Manager, a política a utilizar é identificada como sendo de performance.
3. O Traffic Manager consulta a tabela de tempos de latência e define qual o "melhor"/ "mais próximo" para o cliente em questão.
4. Com base nessa definição o cliente é redireccionado para o Data Center escolhido
5. No momento da selecção é reservado para o cliente, durante um período de tempo definido no momento da criação da política de balanceamento, o DNS que foi seleccionado.
6. Todos os próximos pedidos serão efectuados directamente ao DNS escolhido durante o tempo definido para expiração da reserva.

Criação da Política

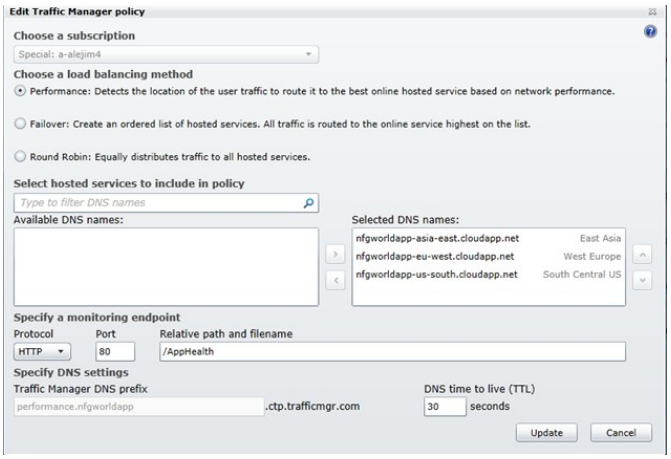
Em primeiro lugar deveremos abrir o Windows Azure Management Portal, e seleccionar a opção Virtual Network



Em seguida seleccionamos o Traffic Manager -> Policies, e seleccionamos o Create, onde iremos criar uma política de balanceamento, e para isso iremos definir:

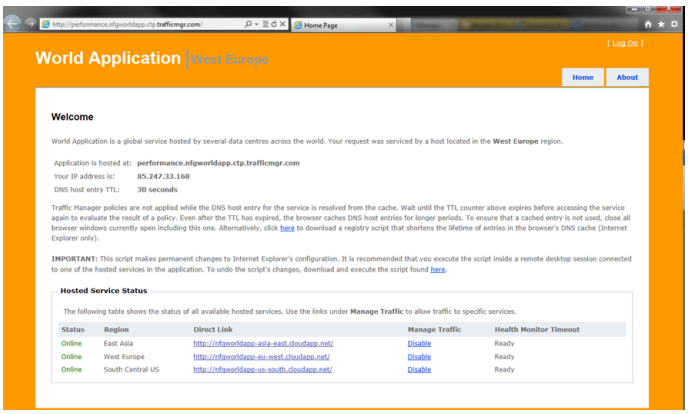
- Método de Balanceamento: Performance

- Hosted Services disponíveis na subscrição que deverão ser considerados
- Endereço de Monitorização: /AppHealth (irá auxiliar a aplicação a conhecer qual o estado de cada uma das politicas).
- DNS do Traffic Manager: performance.nfgworldapp.ctp.trafficmgr.com (este será o endereço que iremos utilizar para obtermos o balanceamento por performance).
- e finalmente o DNS TTL: 30 segundos (define o tempo em que o DNS é reservado para o cliente, e neste caso é curto para facilitar os testes).

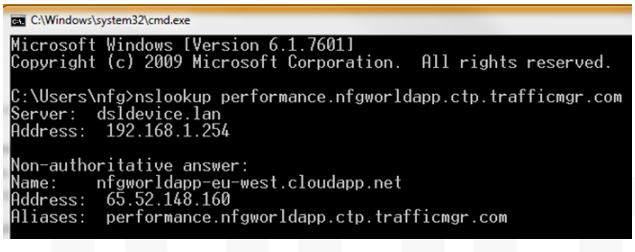


O Resultado

Tendo em conta a configuração que acabamos de efectuar, se efectuarmos o pedido de Portugal, obtemos a solução que se encontra disponível no Oeste da Europa.

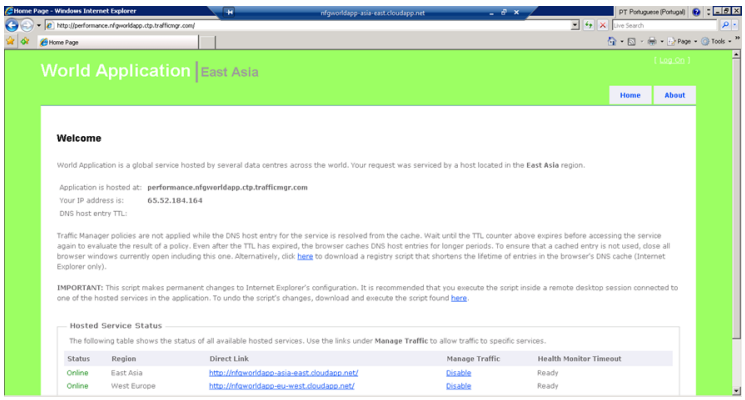


Se procurarmos compreender quem nos está a responder utilizando o comando nslookup, obtemos o seguinte resultado:

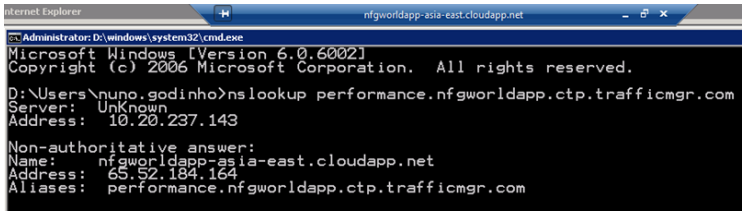


Isto significa que o endereço que está a responder quando o cliente pede o http://performance.nfgworldapp.ctp.trafficmgr.com (endereço da política de balanceamento do Windows Azure Traffic Manager) obtemos a resposta da publicação do Data Center do Oeste da Europa, pois é ele que definiu o endereço http://nfgworldapp-eu-west.cloudapp.net.

Mas se por outro lado tentarmos aceder através de uma máquina na Ásia, o resultado será a solução que se encontra publicada no Data Center do Este da Ásia.

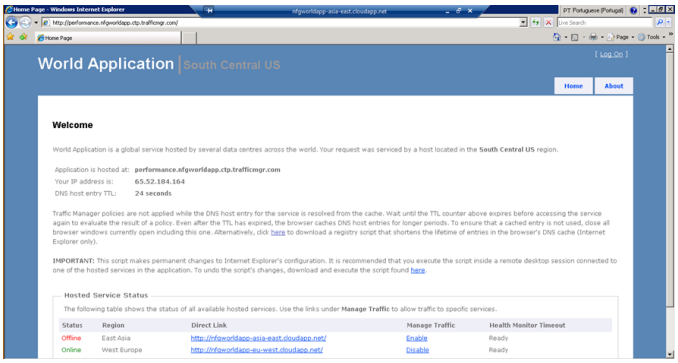


E com o nslookup, o resultado é o seguinte:

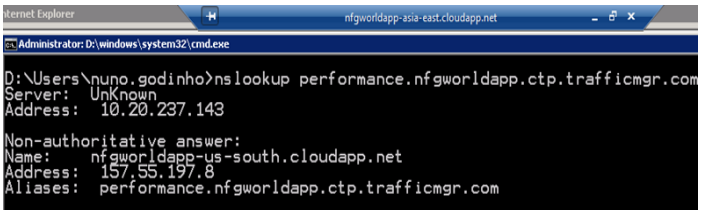


A resposta está a ser servida pelo Data Center mais próximo. E se agora, por algum motivo, uma das publicações não estiver disponível? O que acontece? Para isso vamos desactivar o balanceamento de tráfego do Este da Asia.

Se agora tentarmos aceder novamente ao nosso endereço do Traffic Manager, o resultado será que o trafego será imediatamente redireccionado para o Data Center do Centro Sul dos Estados Unidos.



E utilizando o nslookup, o resultado será:



Ou seja o DNS que é servido ao cliente, e reservado para o mesmo durante o período de tempo definido na politica de balanceamento, será o http://nfgworldapp-us-south.cloudapp.net.

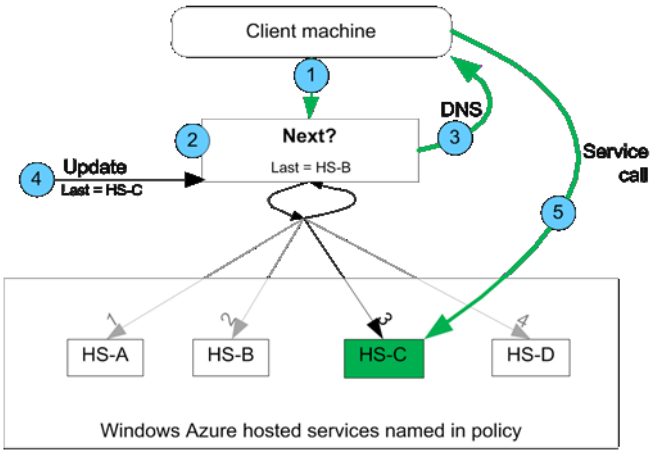
Conclusão

Com base nestes testes, poderemos com bastante certeza dizer que, o Windows Azure Traffic Manager resolve o problema do Balanceamento de Tráfego Global em termos de performance, identificando qual o melhor Data Center, para que a experiência de utilização seja a melhor.

Round Robin

Uma outra opção para o balanceamento do tráfego disponibilizado pelo Windows Azure Traffic Manager, é o Round-Robin, sendo que este é o mais simples uma vez que vai balanceando os pedidos para cada um dos Data Centers com as publicações da nossa solução sequencialmente.

A Teoria

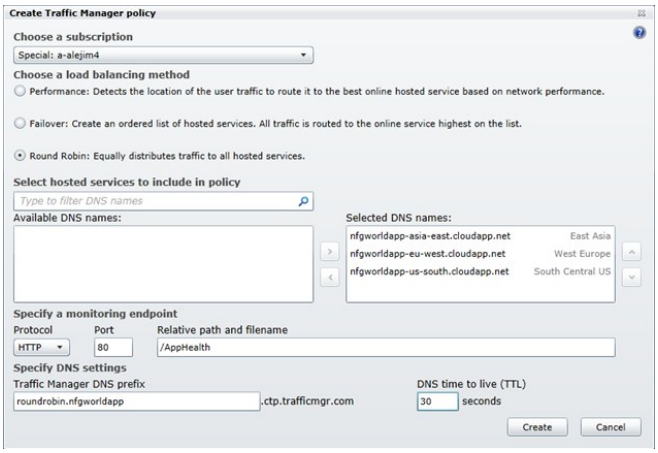


1. O cliente efectua um pedido para o endereço do Windows Azure Traffic Manager, a política a utilizar é identificada como sendo de round robin.
2. Verifica qual o DNS que foi servido por último.
3. Serve o próximo DNS da lista efectuado a reserva para o cliente, durante um período de tempo definido no momento da criação da política de balanceamento, o DNS que foi seleccionado.
4. Actualiza a propriedade que identifica qual o ultimo DNS servido a clientes.
5. Todos os próximos pedidos serão efectuados directamente ao DNS escolhido durante o tempo definido para expiração da reserva.

Criação da Politica

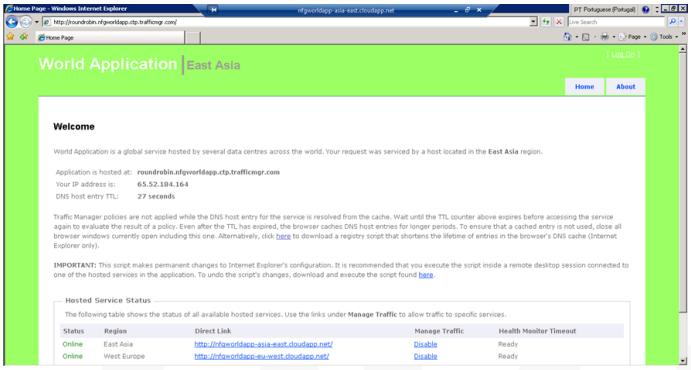
Em primeiro lugar deveremos abrir novamente o Windows Azure Management Portal, e seleccionar a opção Virtual Network. Seguidamente seleccionamos o Traffic Manager -> Policies, e seleccionamos o Create, onde iremos criar uma politica de balanceamento, e para isso iremos definir:

- Método de Balanceamento: Round-Robin
- Hosted Services disponíveis na subscrição que deverão ser considerados, aqui a ordem será importante uma vez que será pela mesma que o balanceamento irá girar.
- Endereço de Monitorização: /AppHealth (irá auxiliar a aplicação a conhecer qual o estado de cada uma das politicas).
- DNS do Traffic Manager: roundrobin.nfgworldapp.ctp.trafficmgr.com (este será o endereço que iremos utilizar para obtermos o balanceamento por round robin).
- e finalmente o DNS TTL: 30 segundos (define o tempo em que o DNS é reservado para o cliente, e neste caso é curto para facilitar os testes).

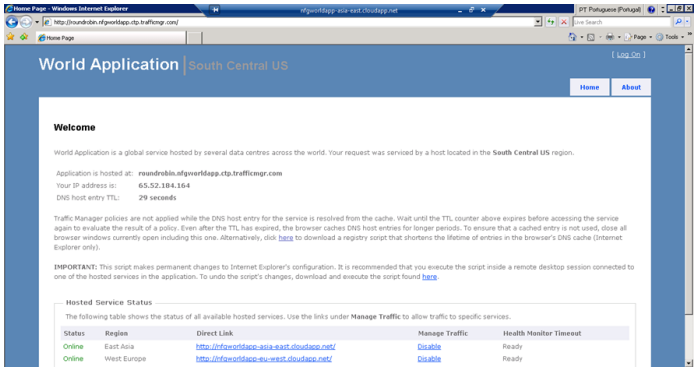


O Resultado

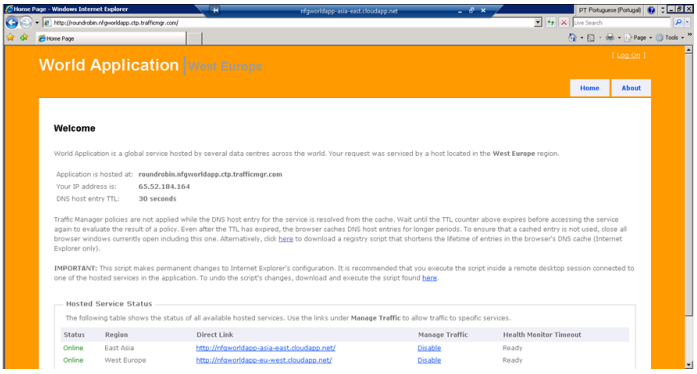
Agora que temos a política de Round Robin criada, vamos passar a verificar se a mesma funciona efectivamente, e para isso vamos efectuar um pedido e recebemos a primeira resposta a partir do Data Center de Este da Ásia.



Refrescando a página após os 30 segundos definidos de reserva do DNS, obtemos o resultado como sendo o Data Center do Centro Sul dos Estados Unidos.



E se efectuarmos mais um refrescamento da página obtemos o Data Center do Oeste da Europa.



Se por alguma razão um dos Data Centers, ou uma das soluções deixar de responder, então iremos imediatamente deixar de ser balanceados para esse mesmo Data Center.

Conclusão

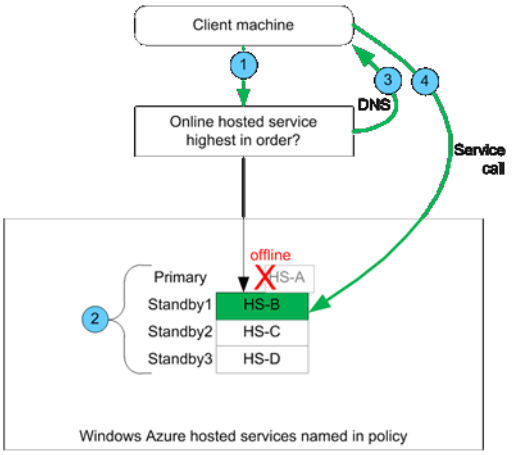
Com base nos testes efectuados anteriormente poderemos afirmar que, o Windows Azure Traffic Manager, resolve também o problema de balanceamento de carga, se objectivo for apenas distribuir uniformemente os pedidos, independentemente dos tempos de latência.

Failover

Por fim existe ainda um outro mecanismo de balanceamento que é denominado de Failover, uma vez que define a ordem pela qual o balanceamento deverá ser efectuado em caso de a publicação principal da solução não se encontrar disponível.

A Teoria

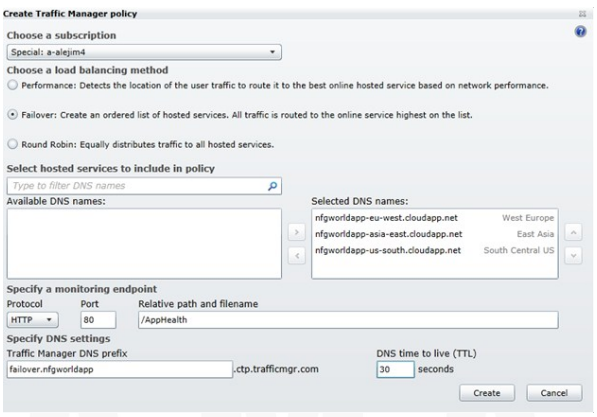
1. O cliente efectua um pedido para o endereço do Windows Azure Traffic Manager, a política a utilizar é identificada como sendo de failover.
2. O sistema verifica se o DNS principal (definido como sendo o que se encontra em primeiro na lista da definição da politica) se encontra disponível.
3. Se o mesmo se encontrar disponível, será esse o DNS servido, senão será servido o imediatamente seguinte da lista efectuando a reserva para o cliente, durante um período de tempo definido no momento da criação da política de balanceamento, o DNS que foi seleccionado.
4. Todos os próximos pedidos serão efectuados directamente ao DNS escolhido durante o tempo definido para expiração da reserva.



Criação da Política

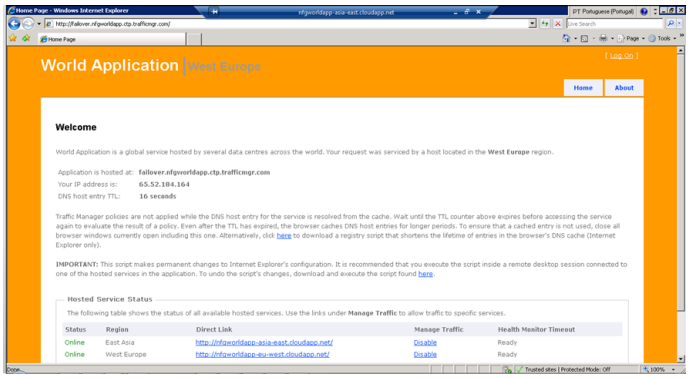
Em primeiro lugar deveremos abrir novamente o Windows Azure Management Portal, e seleccionar a opção Virtual Network. Seguidamente seleccionamos o Traffic Manager -> Policies, e seleccionamos o Create, onde iremos criar uma política de balanceamento, e para isso iremos definir:

- Método de Balanceamento: Failover
- Hosted Services disponíveis na subscrição que deverão ser considerados, aqui a ordem será importante pois será a forma de definir qual o DNS principal e a ordem pela qual será utilizado em caso de falha.
- Endereço de Monitorização: /AppHealth (irá auxiliar a aplicação a conhecer qual o estado de cada uma das políticas).
- DNS do Traffic Manager: failover.nfgworldapp.ctp.trafficmgr.com (este será o endereço que iremos utilizar para obtermos o balanceamento por failover).
- e finalmente o DNS TTL: 30 segundos (define o tempo em que o DNS é reservado para o cliente, e neste caso é curto para facilitar os testes).

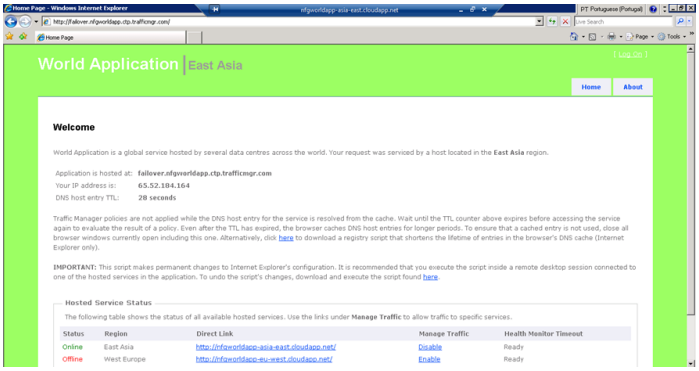


O Resultado

Agora que temos a política de Failover criada vamos passar a verificar se a mesma funciona efectivamente, e para isso vamos efectuar um pedido e somos redireccionados para o Data Center de Oeste da Europa, independentemente da localização da qual nos encontramos a efectuar o pedido, pois foi o que foi definido como sendo o DNS principal. Todos os pedidos subsequentes serão também servidos pelo mesmo DNS.



Mas se por algum motivo o DNS principal não se encontrar disponível seremos imediatamente redireccionados para o DNS seguinte da lista, e assim sucessivamente. Neste caso o DNS secundário refere-se ao DNS do Este da Ásia.



Conclusão

Com base nos testes efectuados anteriormente poderemos afirmar que, o Windows Azure Traffic Manager, resolve ainda o problema de balanceamento de carga, se objectivo for a definição da ordem que o balanceamento deverá seguir em caso de falha ou indisponibilidade do DNS principal.

Mas porque utilizar?

Mas mesmo depois de termos identificado e testado as diversas capacidades que poderemos utilizar como Windows Azure Traffic Manager, poderão surgir algumas dúvidas como por exemplo, que requisitos são que a sua utilização vai resolver. Alguns deles são os seguintes:

- Continuidade e fiabilidade do negócio (Failover)
- Diminuição do Tempo de latência e aumento da performance (Performance)
- Escalabilidade das soluções (Performance)
- Distribuição de Carga (Round Robin)
- Manutenção (Desligar o Tráfego para um DNS – Disable Traffic)
- Suspensão da utilização de uma das politicas (Disable Policy)

Limitações e Funcionalidades da CTP

Agora que já conhecemos melhor o Windows Azure Traffic Manager, quer relativamente à forma como funciona, quer em termos do tipo de requisitos que resolve, será importante analisar quais as limitações e funcionalidades que a versão actual, ainda em CTP, nos disponibiliza.

Limitações

Pensando nas limitações, as principais serão as seguintes:

- Não existe SLA para este serviço
- Não é pago
- Não é neste momento recomendado para utilização em produção, uma vez que não existe um SLA
- Também o domínio será certamente alterado, uma vez que neste momento o mesmo contém referência à versão ctp em que se encontra. (*.ctp.trafficmgr.com)

Funcionalidades

Em termos de funcionalidades disponíveis neste momento, o resumo será:

- Apenas configurável via Management Portal, uma vez que neste momento não existe uma API ou SDK
- Redireccionamento para Hosted Services em Produção apenas
- Métodos de balanceamento
  - ◊ Performance
  - ◊ Round Robin
  - ◊ Failover
- Configurabilidade do TTL (Time to Live) do DNS
- Monitorização via HTTP ou HTTPs em qualquer porta
- Criação, Visualização, Actualização e Remoção de políticas de balanceamento
- Capacidade de Ligar e Desligar políticas de Tráfego
- Capacidade de Ligar e Desligar balanceamento de tráfego para um determinado DNS.

Futuro

Um dos pontos muito importante é compreender qual será o futuro deste produto, e o que o mesmo nos reserva em termos de funcionalidades. Para isso aqui fica uma pequena lista de funcionalidades que poderemos esperar num futuro próximo:

- Relatórios, estatísticas e histórico da monitorização de estado
- Histórico de alterações das políticas de tráfego
- Balanceamento Geográfico - este balanceamento será efectuado com base na definição que de regiões que efectuarmos, sendo que poderemos definir que DNS deverá responder a que região ou país de origem.
- Balanceamento por Rácio - com este balanceamento poderemos definir o rácio de tráfego que cada um dos DNS's irá receber, sendo que a partir desse momento será da responsabilidade do Windows Azure Traffic Manager a gestão e cumprimento dos mesmos.
- Hierarquia de políticas
- Verificação do estado das políticas
- Criação de regiões à medida
- Criação de regras de monitorização
- Alertas

Agora que já sabemos tudo sobre o que existe actualmente na versão CTP, o que poderemos esperar no futuro, como funciona, etc, uma das questões que poderá estar a surgir é sem dúvida, Quando poderemos ter a versão final?

Esta é uma excelente questão pois não existe para já uma data fixa assumida pela Microsoft, sendo que a única coisa que sabemos efectivamente é que esta versão CTP foi disponibilizada no MIX no dia 12 de Abril de 2011, e que terminará no final do Verão. Sendo que estamos no verão e o mesmo termina a dia 23 de Setembro, estamos realmente muito perto de saber alguma novidade sobre este assunto. Se pensarmos ainda que durante o mês de Setembro vai existir uma grande conferência da Microsoft denominada Build, provavelmente teremos uma resposta nessa altura. Mas mais uma vez, não existe uma data assumida pela Microsoft.

Resumo

Em resumo, o Windows Azure Traffic Manager é um Balanceador de Tráfego Global, que se encontra actualmente na sua versão CTP, e que permite resolver diversos problemas em termos de melhoramento da Experiência de Utilização (UX), das nossas soluções, uma vez que nos permite que através de políticas como as de Performance, Round Robin ou Failover definir a forma como esse balanceamento é efectuado, resolvendo desta forma simples e eficaz um problema que muito nos preocupa quando queremos globalizar as nossas soluções e os nossos negócios.

Ainda para mais o processo é bastante simples, pois apenas necessitamos de 3 passos para o conseguir:

1. Publicar a solução para os vários Data Centers a utilizar.
2. Criar as políticas de balanceamento de tráfego.
3. Utilizar

Com estes 3 passos bastante simples conseguimos ter uma solução global, fiável e ao mesmo tempo de elevada Experiência de Utilização, permitindo-nos por isso mesmo chegar mais e melhor aos nossos potenciais clientes, e com isso melhorar os nossos negócios.

Espero que com este artigo vos tenha ajudado a compreender melhor o que é afinal o Windows Azure Traffic Manager e ao mesmo tempo desmistificar a forma de funcionamento, os resultados esperados, e acima de tudo as vantagens para o negócio. Claro que muito ficou por dizer em termos de monitorização das políticas, boas práticas, etc, mas ficou aqui o que me pareceu mais essencial, e em futuros artigos poderemos então falar em maior detalhe sobre esses assuntos.

***Nota**: Não vale a pena tentarem aceder aos endereços dos testes, pois os mesmos já se encontram inválidos, uma vez que a solução foi disponibilizada e utilizada apenas para os testes relativos à escrita deste artigo.*

AUTOR

Escrito por Nuno Godinho

Consultor Independente com 10 anos de Experiência e principal responsabilidade de ajudar os clientes a identificar, planear, gerir e desenvolver soluções e produtos de software. Especialista em Tecnologias Microsoft. Orador em alguns dos maiores eventos de desenvolvimento da Microsoft Portugal como MSDN, TechDays, DevDays, além de eventos internacionais como TechEd Europa, TechEd Online Worldwide, MVP Nation e CloudViews.Org. Microsoft MVP há 4 anos, inicialmente em ASP.NET e a partir do início deste ano em Windows Azure com blogs em http://www.pontonetpt.com/blogs/nunogodinho (Português e Inglês) e http://www.msmvps.org/blogs/nunogodinho (Inglês), INETA Country Leader por Portugal, e parte da equipa de gestão de por diversas comunidades Portuguesas como PontoNetPT, XAMLPT e Fundador da AzurePT (Windows Azure em Português)."

Ferramentas gratuitas de trabalho com SQL Server

A Microsoft SQL Server é uma das bases de dados mais populares do mundo, e qualquer developer profissional ou amador já se deve ter deparado uma ou mais vezes com ela. Independentemente do tamanho da empresa - seja pequena ou grande, trabalhando com C#, VB.NET ou JAVA, uma vasta maioria dos profissionais, já escreveu algumas linhas de código para esta base de dados que não pára de ganhar adeptos por todo o mundo.

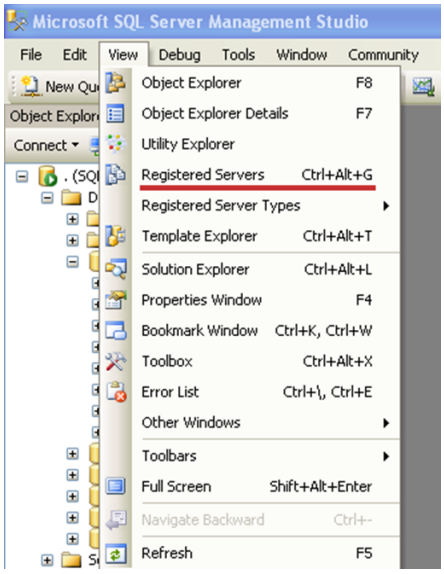
Neste artigo eu pretendo referir algumas opções e ferramentas gratuitas que podem facilitar e acelerar o trabalho com SQL Server.

A SQL Server Management Studio (SSMS) está instalado junto com SQL Server e pode ser instalado tanto em conjunto com o servidor, como em separado. Mesmo a versão Express de SQL Server tem a sua versão de SSMS, que é um pouco limitada em relação à versão paga (não se pode trabalhar com Analysis Services, Reporting Services, Notification Services ou SQL Server Agent, por exemplo) mas mesmo assim permite trabalhar com todas as versões de bases de dados SQL Server desde 2000. Muitos profissionais que trabalham com SQL Server 2000 escolhem instalar o SSMS para ter algumas das funcionalidade avançadas que esta ferramenta oferece. Eu próprio enquanto trabalho com SQL Server 2005, utilizo a SSMS de 2008 se for possível instalar num computador.

A vasta maioria de profissionais e amadores utilizam SSMS nos seus trabalhos, e por isso eu queria partilhar algumas das funcionalidades de SSMS que eu aprecio, mas que considero serem um pouco desconhecidas do utilizador menos avançado.

Para além da maneira tradicional de efectuar uma ligação com a BD de SQL Server através da opção "Connect" do Object Explorer e introduzir o login e a password para cada servidor em separado para cada ligação, existe uma opção mais avançada que se chama "Registered Servers" e que permite configurar uma ligação apenas uma vez para utilizar mais tarde apenas com um "double click" e que inclui algumas opções avançadas que permitem facilitar o trabalho.

No SSMS os "Registered Serveres" tem agrupamento natural pelo tipo de servidores isto é Database Engine, Analysis Services, Reporting Services, SQL Server Compact e Integration Services, o que permite distinguir com facilidade os tipos de trabalho associado com cada um destes engines. Os "Registered Servers" também permitem fazer agrupamento de servidores (Server Group), que pode servir para agrupar os servidores pelo ambiente (Desenvolvimento, Qualidade, Testes e Produção) ou pela aplicação, tendo todos os ambientes respectivos como registos (SuperCalc, MegaMonitor, etc).

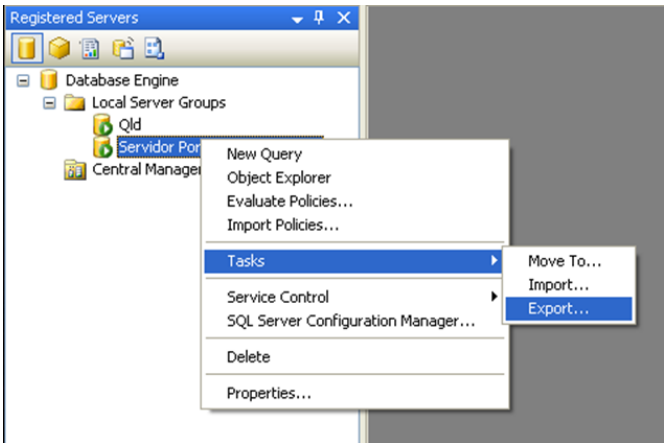


Uma funcionalidade muito avançada, mas ao mesmo tempo perigosa é a capacidade de executar um query em todos os servidores registados de um agrupamento ou em todos os servidores registados naquele computador na totalidade. Essa funcionalidade deve ser utilizada pelos utilizadores muito experientes e sobretudo para tirar alguma informação diagnóstica (e não pesada) sobre os respectivos servidores.

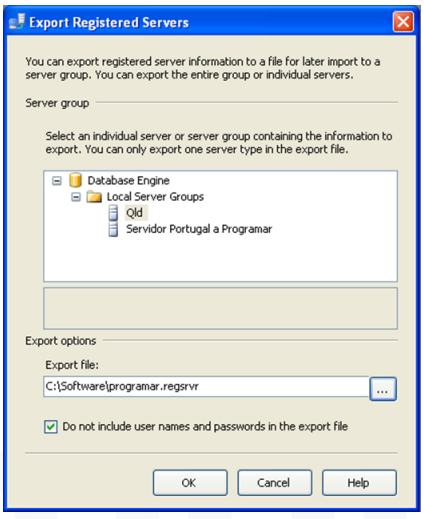
Para activar os "Registered Servers" basta abrir o menú principal "View" e escolher a opção "Registered Servers" ou como alternativa utilizar a combinação de teclado "CTRL-ALT-G".

Logo após a escolha desta opção vamos ter o seguinte view no ecrã do nosso SSMS, onde poderá consultar a lista de todos os servidores registados agrupados pelo “Server Groups”.

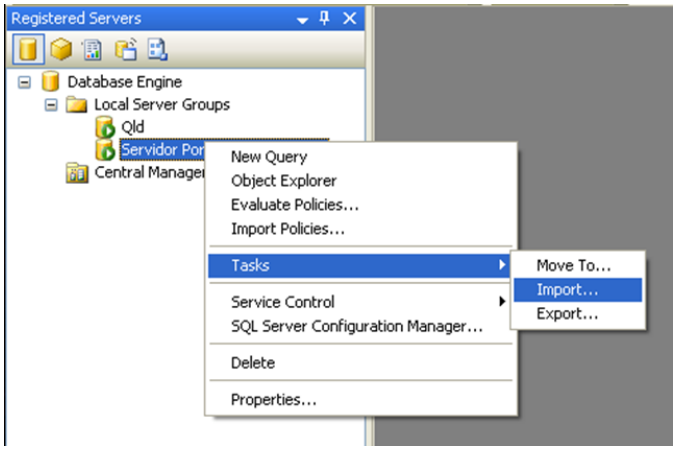
Sempre podemos extrair a lista dos servidores registados para partilhar com novos colegas do nosso projecto, que conseguem importar o XML gerado pela ferramenta de exportação com muita facilidade.



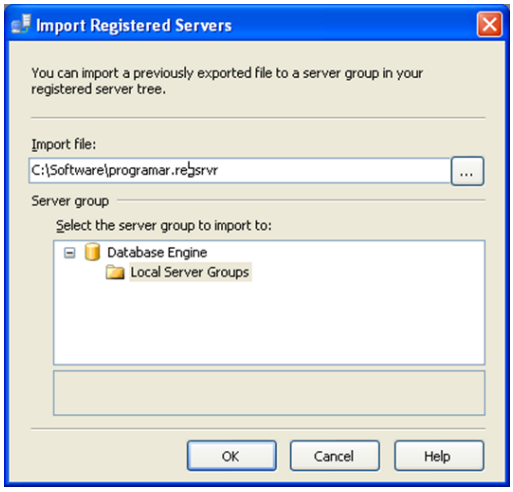
No ecrã de exportação de definições dos servidores registados existe uma opção para exclusão do nome do utilizador e da password, para evitar os problemas de segurança, pois é de uma boa prática no geral manter utilizadores separados para cada membro da equipa. Mas por outro lado na altura de mudança de lugar de trabalho, a possibilidade de "copiar" os nossos servidores de trabalho é uma funcionalidade que pode poupar imenso tempo. Nesta altura a migração fica muito facilitada, pois não vamos precisar de criar todos os servidores e grupos de novo introduzindo as credenciais.



Importação do servidor registado



A importação do ficheiro XML com servidores registados é um processo intuitivo: escolhe-se um grupo de servidores e faz-se click na tecla direita do rato e a seguinte escolha de opções "Task->Import" no menu contextual do SSMS. Basta escolher o ficheiro em questão e simplesmente clicar no OK para poder finalizar o processo tal como pode ver nas imagens ao lado deste texto.



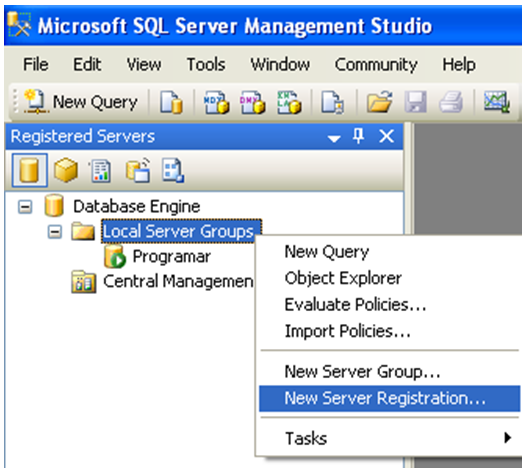
Criar um novo grupo de servidores registados

Para definição de um novo grupo de servidores registados basta clicar com a tecla direita do rato e escolher a opção com "New Server Group" que vai mostrar um novo diálogo para definição do nome e da descrição do grupo de servidores que estamos a definir.

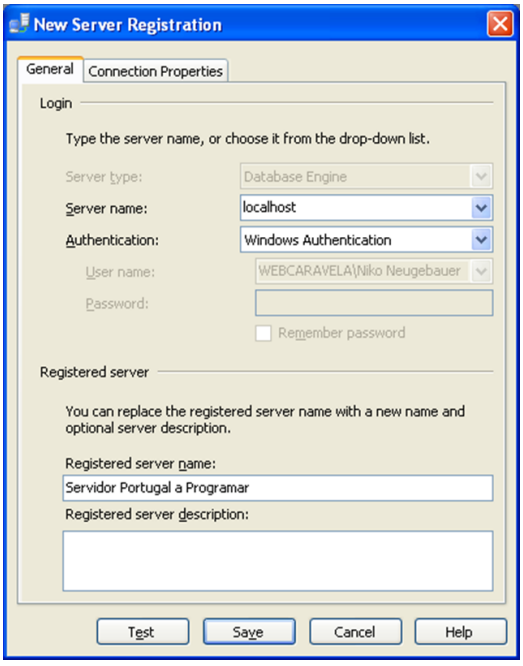
O processo por si é fácil e não deve criar dificuldades a ninguém que trabalha com uma ferramenta como SQL Server. A única coisa estranha é que para a edição do nome de grupo não existe nenhuma opção no menu contextual, pois a edição poderá ser efectuada apenas através dos métodos tradicionais de edição, como em Windows Explorer por exemplo – 2 cliques com intervalo da tecla esquerda do rato ou como alternativa através da tecla F2.

Criação de um novo servidor registado:

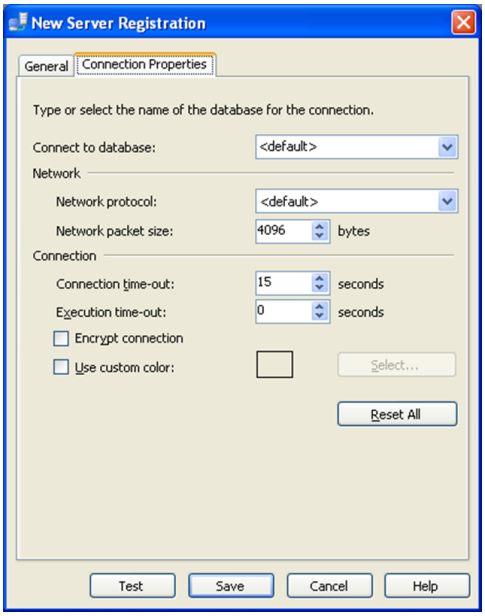
A criação de um novo servidor registado é uma operação bastante trivial – após a escolha de um grupo dos servidores onde pretendemos criar um novo servidor, clicamos na tecla direita do rato e avançamos com a opção "New Server Registration" para avançar com um dialogo de definição das propriedades de um novo servidor registado.



No ecrã de definição de um novo servidor registado temos as opções habituais para escolha do nome de servidor, do tipo de autenticação, do nome de utilizador e da password se for necessário e podemos definir o nome sobre qual o servidor vai aparecer na nossa lista, mais a descrição opcional que serve como um explicativo desta ligação.



Na definição de um novo servidor, para além de opções "General", temos um tab nomeado "Connection Properties", que permite definir diversas opções de configuração, que podem ser muito úteis – "encrypt connection" e "custom color".



A opção "encrypt connection" é auto-explicatória e deverá ser utilizada para aumentar a segurança, mas a opção "custom color" ainda não tem o reconhecimento que ela realmente merece: para definir uma cor customizada nós podemos assegurar o reconhecimento do tipo do ambiente onde estamos a trabalhar. Eu costumo definir 3 cores distintas para cada um dos ambientes de trabalho – o verde para o ambiente de desenvolvimento, o amarelo para o ambiente de qualidade e o vermelho para produção. Ainda antes de gravar o nosso novo servidor registado, podemos sempre testar a nossa ligação através do botão "Test" para ter a certeza que todas as credenciais foram introduzidas correctamente.

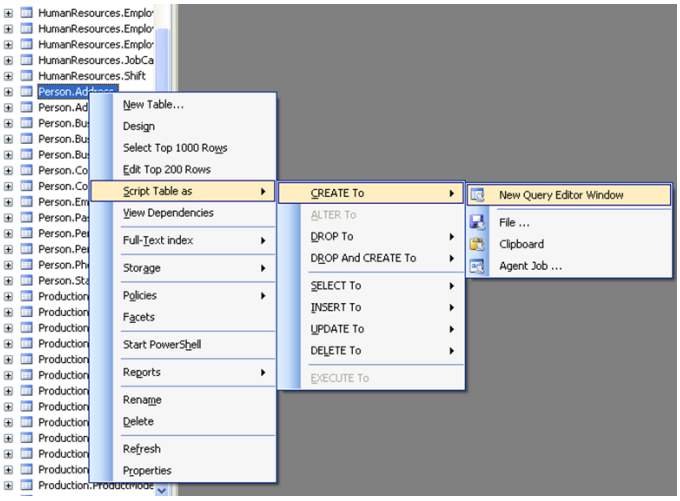
Assim, após estabelecimento de uma ligação a um servidor na parte baixa de janela dos queries teremos uma barra de cor definida para esta ligação, que assim nos vai servir de referência e de aviso sobre o servidor com o qual estamos a trabalhar.

Tendo uma barra vermelha em baixo qualquer um prestará mais atenção àquilo que está a fazer, o que é extremamente importante para o trabalho com servidores de ambiente de produção.

Atenção: se abrir uma nova ligação através do botão "New Query", mesmo tendo uma ligação com o servidor estabelecida através dos "Registered Servers" a cor da barra no painel de queries ficará perdida.

Script Object As

Por vezes, durante o trabalho nós precisamos de consultar ou extrair um script completo de criação de um objecto de base de dados, como uma tabela por exemplo. Depois de criar uma tabela, se queremos passá-la para outro ambiente como QLD(Qualidade) ou PRD(Produção), nós podemos criar a tabela com o T-SQL e depois disponibilizar os ficheiros para os Database Administrators (DBA's). Mas o que é que deverá ser feito se a tabela em questão foi criada por uma outra pessoa que se esqueceu de guardar o script durante o procedimento de criação da Tabela? Acabamos sempre por precisar de "scriptar" a nossa Tabela, e para isso dentro de SSMS existe uma opção de criação de um script de T-SQL para um objecto.

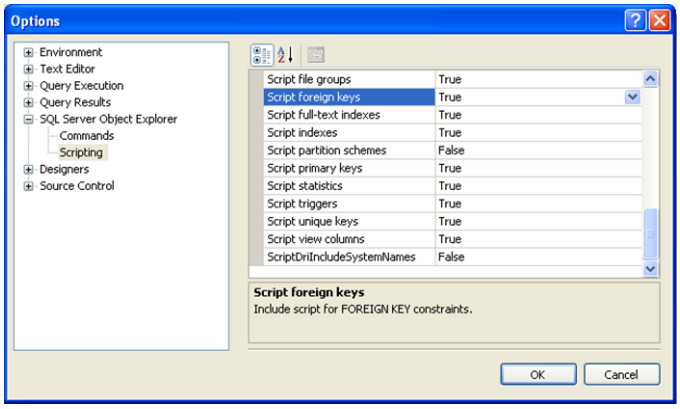


No exemplo que eu queria partilhar aqui, eu vou explorar a possibilidade de criar um script Transact-SQL (T-SQL) com criação de uma tabela – para isto escolha-se um objecto no "Object Explorer", fazemos o click com tecla direita do rato por cima deste objecto e escolhemos a opção "Script Table as" -> "Create To" e "New Query Editor Window".

Parece ser fácil, não é verdade?Na verdade, por defeito após instalação, a SSMS tem definições que não incluem alguns dos objectos associados à nossa tabela no script gerado, o que leva muitos profissionais a dizerem que a SSMS está mal feita e que simplesmente não disponibiliza toda a informação necessária.

Na verdade, a SSMS permite um controlo mais afinado de opções de exportação: podemos escolher nas opções quais objectos (triggers, constraints, statitistics, etc) vão ser incluídos na geração do nosso script de T-SQL.

Para consultar e editar estas opções escolhemos o menu "Tools" e na janela "Options" escolhemos o submenu "SQL Server Object Explorer"->"Scripting" onde poderemos consultar e editar uma lista extensa de objectos que vão ser incluídos ou excluídos na geração do script.



As diversas opções que este menu contém incluem opções como a versão de SQL Server para qual o script deverá ser gerado – se tiver o SQL Server 2008 R2 instalado no PC onde estamos a trabalhar. Mas se o servidor com qual trabalhamos é o SQL Server 2000, podemos simplesmente escolher a opção de SQL Server 2000 no General Scripting Options e o resultado final será gerado tendo em conta o facto que o produto final será utilizado no servidor de SQL Server 2000. As 2 opções "Database Engine Type" e "Script for Server Version" permitem adaptar o script para as necessidades do projecto entre SQL Server 2000 e até SQL Azure inclusive.

Uma outra opção que eu escolho por defeito logo após instalação de SSMS é "Include IF NOT EXISTS clause", que permite uma inserção de um script que vai eliminar o objecto se ele não existir – o que facilita imenso a passagem dos scripts como CREATE TABLE, assim tendo a certeza que caso um objecto com o nome da nossa tabela já exista, ele será logo eliminado pelo script disponibilizado pela SSMS.

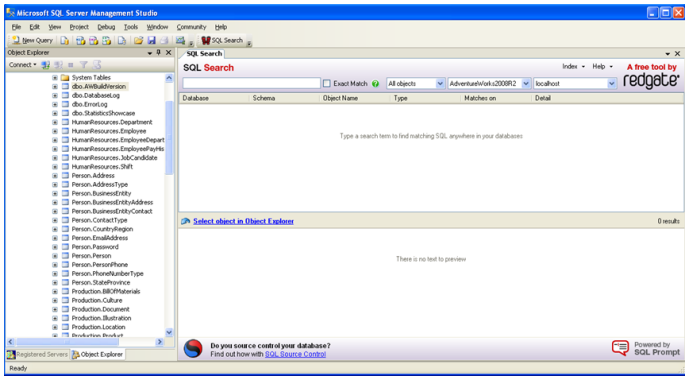
De resto eu deixo-vos explorar todas as diversas opções que existem neste menu, alertando para activar logo o scripting de objectos como Foreign Keys, Indexes, Unique Keys e Triggers por exemplo.

Resta apenas acrescentar uma pequena crítica, pois muitas destas opções deveriam ser muito mais acessíveis para uma edição conforme as necessidades e não serem escondidas dentro de opções de SSMS.

RedGate SQL Search

A famosa empresa inglesa de desenvolvimento de ferramentas para os Developers e DBA's, a RedGate disponibiliza gratuitamente uma das mais úteis ferramentas para quem trabalha com SQL Server - RedGate SQL Search.

A funcionalidade desta ferramenta por si é bastante fácil e durante muito tempo os profissionais utilizavam os quereis para procurar por um objecto na base de dados. O problema era sempre a interface e a complexidade destas queriesPor exemplo, um ajuste que estamos à procura do conteúdo apenas entre os "Stored Procedures" era suficiente para obrigar o utilizador a editar a query e assim na sequência afastar deste método os programadores com menos experiência, pois ninguém quer fazer afirmações erradas.



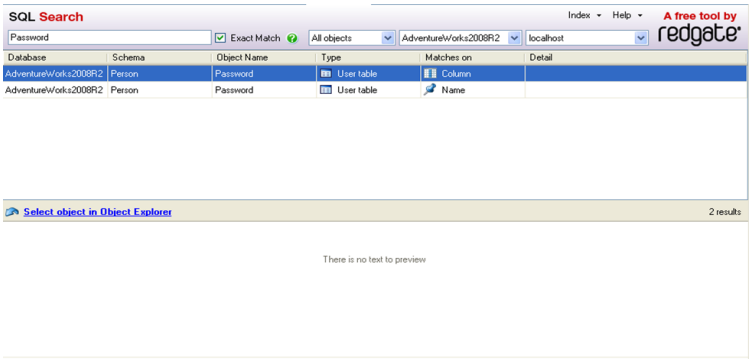
A capacidade de procura de um conteúdo específico é sempre uma mais-valia, pois por exemplo no uso de motores de busca como Google ou Bing para um informático, considero importante saber como limitar uma pesquisa dentro de um site específico (syntax site:www.portugal-a-programar.org) ou conhecimento de aplicação de sintaxe mais básico para excluir uma palavra de pesquisa (incisão do símbolo '-' antes de palavra).

A SQL Search integra-se dentro de SSMS num processo de instalação praticamente instantâneo, e o toolbar da ferramenta consiste apenas num botão, que invoca o painel especial de ferramenta. A interface do SQLSearch é muito simples e intuitiva, tendo apenas as seguintes opções:

- uma textbox para introduzir os termos de pesquisa
- uma checkbox para especificar se a pesquisa deverá ser exacta. Esta opção serve como uma indicação para o SQL Search para activar a utilização de lógica binária tipo AND, enquanto sem activação desta opção, por defeito, a SQL Server utiliza a pesquisa de tipo OR. Um bom exemplo desta lógica é quando pesquisamos pela frase “select *” sem opção de pesquisa exacta a ser activada, a ferramenta vai encontrar todas as referências de palavra select e do símbolo * (nota: o asterisco está utilizado para muitos propósitos como definir o comentário ou indicar a lista completa de colunas do operador SELECT) dentro de universo de pesquisa definida, utilizando a lógica binária de tipo OR.
- uma lista com opções de tipos de objectos procurados, que inclui All Objects, Tables, Views, Procedures, Constraints, Triggers, Functions, e Others. Nota: pela minha experiência, eu gostaria de ter esta lista alargada numa próxima versão da ferramenta, para incluir os filtros pelos Types, Assemblies, Synonims, Users e Roles.
- uma lista com nomes de todas as bases de dados encontradas na instância de SQL Server a qual estamos ligados, e onde para além de escolher uma BD específica podemos também optar por fazer a pesquisa em todas as bases de dados da instância (Essa funcionalidade é mesmo muito útil para os DBA's)
- uma lista com instâncias às quais já temos estabelecido uma ligação.
- uma opção “Index”, meio-escondida no canto superior direito e que tem 2 opções muito importantes que muitos dos profissionais infelizmente ignoram:

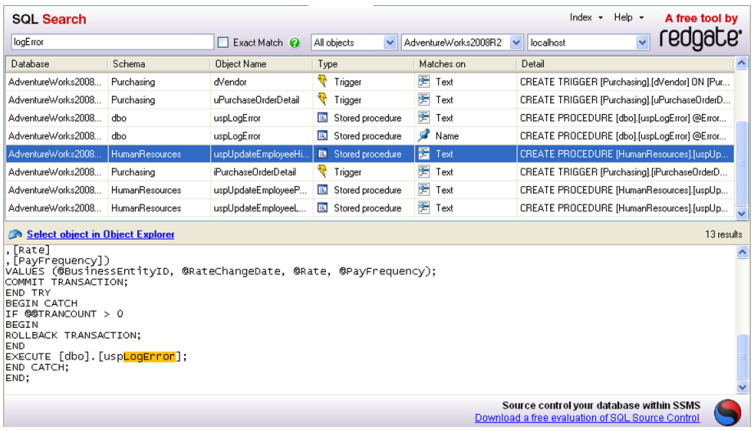
  a. **Refresh** (teclas CTRL+ALT+D) - que permite actualizar a informação interna de ferramenta com os objectos que foram criados/alterados/apagados desde o momento da última invocação do Painel de pesquisa. Essa opção é o caminho mais certo para ter a certeza do resultado de pesquisa, e eu costumo de utilizá-la com alguma frequência durante o dia, em vez de fechar e reabrir o Painel de Pesquisa. :)
  b. **Force Reindex** - é uma opção de Refresh "in extremis", que obriga o SQL Search a fazer uma reindexação total de todos os objectos que a ferramenta permite encontrar, e que deverá ser utilizado em casos raros, quando alguma coisa no funcionamento da ferramenta funcionou erradamente.
- uma opção de ajuda “Help” que disponibiliza 3 opções:

  a. **Support Forum** - é o fórum de RedGate onde podem ser encontradas várias dicas e soluções para utilização de ferramenta
  b. **Provide Feedback** - efectivamente é um link para o formulário de RedGate, onde pode ser transmitido o "feedback" da experiência de ferramenta.
  c. **About SQL Search** - invocação de um ecrã informativo sobre a versão de ferramenta



O processo de pesquisa é muito intuitivo e fácil - basta introduzir uma parte do nome do objecto pretendido e após um click no "Enter", a SQL Search vai efectuar a pesquisa pelo texto pretendido entre os objectos do sistema, os textos de Stored Procedures, Functions, Triggers, etc …

O painel com resultados de pesquisa disponibiliza a funcionalidade extremamente útil – após escolher o objecto encontrado, apenas com um double click poderemos ter este objecto aberto e focado no "Object Explorer", o que na sequência permitirá a utilização deste objecto tal como invocação ou edição de conteúdo no caso de uma stored procedure.



Conclusão

Aconselho a todos que trabalham com SSMS e SQL Server 2005+ a instalar esta ferramenta e começar a experimentar a utilizar – pois o vosso modo de trabalho no dia o dia nunca será o mesmo.

Algumas das funcionalidades que a SQL Search tem, estarão disponíveis na próxima versão de SQL Server, nome de código Denali, cuja FINAL RELEASE está a ser apontado para o final deste ou início do próximo ano. O projecto "Juneau" que deverá integrar a experiência de todos os profissionais que trabalham com SQL Server numa plataforma única, baseada no Visual Studio - é o projecto mais significativo dos últimos anos para os Developers que trabalham com SQL Server.

Nota: todos os screenshots foram tirados de versão SQL Server 2008 R2.

Recursos:

SQL Server 2008 R2
http://technet.microsoft.com/en-us/sqlserver/ff398089

RedGate SQLSearch:
http://www.red-gate.com/products/sql-development/sql-search/

SQL Server "Denali" project Juneau:
http://msdn.microsoft.com/data/tools

AUTOR

Escrito por Niko Neugebauer

Actualmente é um Sénior Professional da Novabase, a maior empresa de informática de Portugal. Possui uma experiência de 13 anos nas áreas de desenvolvimento, desenho e arquitectura de Software. É o fundador e o actual líder da comunidade portuguesa de SQL Server – SQLPort (http://www.sqlport.com). Possui certificação MCITP em DEV de SQL Server 2005 e foi reconhecido pela Microsoft com a atribuição do título MVP (Most valuable Professional). Twitter: @NikoNeugebauer

Elege o melhor artigo desta edição

Revista PROGRAMAR

http://tiny.cc/ProgramarED30_V

NHibernate - Do Import-Package à primeira iteração

ORM - O quê, porquê, quando?

NHibernate é um mapeador objecto-relacional (ORM, do inglês Object-Relational Mapper) que pretende facilitar a tradução de dados armazenados em bases de dados relacionais num grafo de objectos usado por linguagens orientadas a objectos na plataforma .NET. É uma biblioteca madura, já com vários anos de desenvolvimento, open source (usa a GNU Lesser General Public License) e é usada em vários projectos comerciais e open source pelo mundo inteiro.

Neste artigo vamos usar uma biblioteca auxiliar chamada Fluent-NHibernate que trás uma API fluente que tenta ser mais intuitiva e que permita uma descoberta natural das funcionalidades disponíveis conjuntamente com a possibilidade de configurar os mapeamentos de NHibernate de forma automática, tirando partido de convenções definidas em código que indicam qual a regra geral para mapear os objectos com tabelas. A versão de Nhibernate usada aqui é a 3.1, com FluentNHibernate 1.2, ambos instalados através de NuGet (ferramenta para instalação e gestão automática de pacotes criada pela Microsoft sob a égide da Outercurve Foundation).

Domínio

Vamos considerar para este artigo um domínio simples: uma edição com artigos e autores. Uma edição tem vários artigos, um artigo tem vários autores:

```
namespace Portugal_A_Programar.Entidades
{
    public class Edicao
    {
        public DateTime DataEdicao { get; set; }
        public Artigo TemaDeCapa { get; set; }
        public IList<Artigo> Artigos { get; set; }
    }

    public class Artigo
    {
        public string Titulo { get; set; }
        public ISet<Autor> Autores { get; set; }
    }

    public class Autor
    {
        public string Nome { get; set; }
        public string Email { get; set; }
        public ISet<Artigo> Artigos { get; set; }
    }
}
```

Instalação

O processo de instalação é facilitado pelo NuGet:

```
Install-Package fluentnhibernate
```

Este comando, executado na consola de NuGet, instala o NHibernate juntamente com o FluentNHibernate 1.2 e as bibliotecas auxiliares das quais dependem.

Este comando acrescenta as seguintes referências a assemblies:

- Castle.Core
- FluentNHibernate
- Iesi.Collections
- NHibernate
- NHibernate.ByteCode.Castle

Como base de dados vamos usar SQLite, através da biblioteca System.Data.SQLite. Desta forma não é necessário instalar e configurar uma base de dados para correr o código deste artigo. Visto o NHibernate fornecer abstrações sobre a base de dados, todo o código deste artigo funciona da mesma forma para outros motores de bases de dados, sendo apenas necessário mudar o driver de ligação à base de dados e a respectiva connection string.

A instalação deste pacote é igualmente fácil utilizando o NuGet:

```
Install-Package System.Data.Sqlite
```

Configuração

A configuração é feita fluentemente através do interface começado por Fluently.Configure:

```
var sessionFactory = Fluently.Configure()
.Database(SQLiteConfiguration.Standard
.ConnectionString("Data Source=database.sqlite"))
  .Mappings(mapping => mapping.AutoMappings.
  Add(AutoMap.AssemblyOf<Edicao>()
 .Where(t => t.Namespace.EndsWith("Entidades"))))
  .ExposeConfiguration(configuration =>
              new SchemaUpdate(configuration)

.Execute(false, true))
    .BuildSessionFactory();
```

Neste caso, relativamente aos métodos "Database" e "Mappings" estamos a configurar a base de dados e os mapeamentos. Através do método "ExposeConfiguration" estamos a expor a configuração de modo a que o processo de criação e actualização do esquema da base de dados seja feito automaticamente de acordo com o esquema actual de classes e, finalmente, os mapeamentos definidos. Finalmente, estamos também a construir a ISessionFactory que será um dos pontos principais de contacto com o NHibernate.

Configuração da ligação à base de dados

No código anterior estamos a indicar com o método .Database que nos queremos ligar a uma base de dados Standard de SQLite, com a Connection String indicada.

No entanto para ligar a uma base de dados SQLServer 2008 é tão simples quanto substituir a referência ao SQLLite por:

```
MsSqlConfiguration.MsSql2008.ConnectionString
(connectionString)
```

Também é possível aqui configurar outros parâmetros da base de dados através dos métodos disponibilizados pelo configurador, nomeadamente o tamanho do batch de ADO.NET.

Existem configuradores para as mais variadas bases de dados, sendo que é possível sempre construir um através da implementação do interface IPersistenceConfigurer

Configuração dos mapeamentos

Com o método Mappings é possível configurar a forma como NHibernate faz o mapeamento entre as classes existentes no programa e o esquema de tabelas da base de dados.

Neste caso estamos a indicar que queremos mapear automaticamente todas as classes da Assembly da classe "Edição" cujo namespace acabe em "Entidades" com as convenções que estão configuradas por omissão.

Com FluentNHibernate existem 3 tipos de mapeamentos:

- Fluentes
- Automáticos
- Hbm (nativos de NHibernate)

Estes três tipos de mapeamento estão disponíveis nos três membros da instância que é passada na função .Mappings : mapping.FluentMappings, mapping.AutoMappings e mapping.HbmMappings.

Mapeamentos fluentes permitem declarar em código para cada classe a forma como esta se mapeia com a base de dados (incluindo relações, identificadores, colunas e tabelas). Neste caso, para cada classe mapeada existe uma classe de mapeamento.

Mapeamentos automáticos permitem declarar a forma geral de como se mapeiam as entidades através de convenções e overrides. Desta forma é possível evoluir os modelos de classes sem ter que alterar os mapeamentos. Convenções definem as regras gerais que traduzem tipos de dados em tabelas, colunas e chaves, e overrides definem casos particulares que devem ser tratados de forma especial.

Mapeamentos em XML (ficheiros .hbm) são usados nativamente pelo NHibernate em soluções tradicionais com essa biblioteca. De forma análoga ao que acontece com os mapeamentos fluentes, também existe um mapeamento de hbm por classe, podendo estes estar declarados num só ficheiro ou em vários.

Exposição da configuração

Caso seja necessário usar directamente o objecto de configuração de NHibernate é possível obtê-lo através do método .ExposeConfiguration. Como a configuração apenas é concretizada no momento de instanciação da ISessionFactory na chamada ao método de .BuildSessionFactory, para poder aceder à configuração é passada uma Action<Configuration> que recebe a configuração construída por FluentNHibernate e actua sobre a mesma.

Neste caso estamos a construir uma instância da classe SchemaUpdate que acede à base de dados configurada, inspeciona as tabelas e colunas existentes, os mapeamentos definidos e executa os comandos necessários para que a base de dados corresponda ao mapeamento, o que inclui criar tabelas e acrescentar colunas. No entanto, com esta configuração não são realizadas alterações destrutivas - como remover colunas ou tabelas.

Desta forma conseguimos evoluir o desenvolvimento inicial de uma forma mais célere deixando a sincronização da base de dados com o modelo de objectos para esta ferramenta e, numa fase posterior, gerir manualmente o esquema de base de dados.

Primeiro uso

Se tentarmos executar o código de configuração já apresentado, observamos que ele falha com a seguinte excepção:

> *Unhandled Exception: FluentNHibernate.Cfg.FluentConfigurationException: An invalid or incomplete configuration was used while creating a SessionFactory. Check PotentialReasons collection, and InnerException for more detail.*

A InnerException contém o motivo do erro:

> *---> FluentNHibernate.Visitors.ValidationException:*

> *The entity 'Artigo' doesn't have an Id mapped. Use the Id method to map your identity property. For example: Id(x => x.Id).*

Neste caso o erro é relativo à classe Artigo, que não tem uma identificação mapeada para um campo que será a chave primária da tabela. Por omissão os mapeamentos automáticos esperam uma propriedade inteira com o nome Id, o que não acontece em nenhuma das nossas classes.

Identificadores

Existem vários tipos de identificadores:

- Identity/Sequence (gerido pela base de dados)
- HiLo
- Guid/Comb
- Assigned

Identity é a forma tradicional, em que a base de dados gera um identificador para cada linha introduzida numa tabela. Implica que de cada vez que se persiste uma nova entidade seja necessário um round-trip à base de dados para saber qual é o identificador, por oposição a apenas ser feito um round-trip quando se faz commit da transacção.

HiLo é o esquema actualmente recomendado na grande maioria dos casos. Cada cliente (na realidade cada instancia de ISessionFactory), quando precisar de um identificador, reserva na base de dados os próximos N identificadores a serem atribuídos às entidades que vão ser gravadas. Desta forma, é possível gerir os identificadores em batch do lado do cliente, mantendo-se no entanto a unicidade dos mesmos.

É comum observar-se uma subida rápida dos identificadores que usam este esquema durante a fase de desenvolvimento, com largos espaços entre identificadores, porque de cada vez que é instanciada uma ISessionFactory é reservada mais uma sequência de elementos. No entanto, quando o sistema se encontra em produção isto já não se verifica, visto uma ISessionFactory ter um ciclo de vida tão longo quanto a aplicação e, como tal, não é tão comum reservar sequências sem as usar. No entanto, caso seja necessário, é sempre possível alterar o número de identificadores reservados por cada instância, que tem de ser igual entre todas as instâncias concorrentes.

Guid são os tradicionais identificadores únicos que são gerados do lado do cliente. No entanto, devido a aleatoriedade dos mesmos, tipicamente origina uma elevada fragmentação na base de dados. Nesse caso é possível recorrer a Comb, um esquema alternativo que gera identificadores de forma mais sequencial, reduzindo, assim, a fragmentação na base de dados.

Finalmente, um identificador pode ser do tipo **Assigned** quando ele é gerido pelo código cliente da biblioteca, como no caso de chaves naturais.

Convenções e Overrides

No nosso exemplo estamos a usar mapeamentos automáticos. Como tal, há duas formas principais de indicar os mapeamentos: através de regras gerais aplicadas a todas/algumas classes mapeadas (convenções) ou através de indicações específicas para um tipo de mapeamento indicado (overrides)

Convenções

Para este caso vamos definir a convenção de que os identificadores são sempre membros com o nome Id, do tipo inteiros, e gerados no cliente com o mecanismo HiLo.

```
public class IdsConvention : IIdConvention
{
    public void Apply(IIdentityInstance instance)
    {
        instance.Column("Id");
        instance.GeneratedBy.HiLo("HiLo", "Hi",
                                  "100");
    }
}
```

Neste caso estamos a indicar a coluna "Hi" da tabela "HiLo" que será uma tabela na base de dados utilizada para armazenar as sequências, com um valor de 100 para o número de identificadores reservados por ISessionFactory.

Todas as convenções existentes têm como interface base IConvention, sendo possível explorá-las no namespace FluentNHibernate.Conventions.

Caso a convenção apenas se deva aplicar a parte das entidades mapeadas, existem os interfaces que herdam de IConventionAcceptance<TInspector>, tipicamente um para cada convenção, que definem um método que indica se para a parte inspecionada irá ser aplicada a convenção respectiva.

Overrides

No entanto, para o caso da Edicao, vamos fazer um override às convencõesconvenções e indicar que o identificador é o membro DataEdicao, do tipo datetime, e que será atribuido-atribuído por nós, visto ser uma chave natural.

```
public class EdicaoOverride : IAutoMappingOver-
ride<Edicao>
{
 public void Override(AutoMapping<Edicao> mapping)
    {
        mapping.Id(edicao => edicao.DataEdicao)
            .Column("DataEdicao")
            .GeneratedBy.Assigned();
    }}
```

A indicação da classe à qual se aplica o override está no argumento de tipo do interface *IAutoMappingOverride*.

Alterações as classes e nova versão da inicialização

Para tal vamos alterar as classes Artigo e Autor da seguinte forma:

A classe Edicao não necessita de ser alterada, uma vez que a sua chave é definida de forma específica e a propriedade "DataEdicao" já existe.

```
public class Artigo
{
    public int Id { get; set; }
    public string Titulo { get; set; }
    public ISet<Autor> Autores { get; set; }
}

public class Autor
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public string Email { get; set; }
}
```

Para o FluentNHibernate aplicar as convenções e overrides é preciso alterar a chamada ao método .Mappings do seguinte modo:

```
.Mappings(mapping => mapping.AutoMappings
  .Add(AutoMap.AssemblyOf<Edicao>()
    .Where(t => t.Namespace.EndsWith("Entidades"))
.Conventions.AddFromAssemblyOf<IdsConvention>()
.UseOverridesFromAssemblyOf<EdicaoOverride>()))
```

Aqui indicamos que queremos usar todas as convenções existentes na assembly da classe IdsConvention, e os overrides da assembly da classe EdicaoOverride. Deste modo, o FluentNHibernate procura automaticamente os tipos **públicos** dentro das assemblies e usa-os para configuração.

Após estas alterações estamos prontos para tentar correr o código outra vez.

Ao correr esta versão deparamo-nos com mais uma *FluentConfigurationException*:

> *The following types may not be used as proxies:*
>
> *artigo_1.nhibernate.Entidades.Artigo: method get_Id should be 'public/protected virtual' or 'protected internal virtual' artigo_1.nhibernate.Entidades.Artigo: method set_Id should be 'public/protected virtual' or 'protected internal virtual' (...)*

Neste caso, tal como podemos observar na excepção, ele queixa-se de que as propriedades das classes mapeadas devem ser marcadas como virtuais. As mesmas mensagens apareceriam para quaisquer métodos que as classes tivessem.

Lazy loading, proxies e virtual

Quando se carrega um objecto da base de dados, geralmente não se pretende obter o objecto com todas as suas relações e listas preenchidas. Se assim fosse, para além de aumentar a carga sobre a base de dados, seria possível carregar todos os dados se existissem relações do objecto com todas as outras entidades.

Por razões de praticabilidade e de performance, quando um objecto é carregado, todos os objectos relacionados e listagens não são criadas como instâncias das classes mapeadas, mas sim como proxies que representam as mesmas, cujo acesso às propriedades despoletam o carregamento a partir da base (de dados). Dessa forma, caso não se aceda a essas outras entidades, os dados não são lidos da base de dados. A este comportamento dá-se o nome de Lazy Loading

No entanto, para que este mecanismo possa funcionar, de modo a permitir que a Common Language Runtime (CLR) execute os métodos das proxies e não das classes mapeadas, é necessário marcar as propriedades e métodos como virtual.

Outra hipótese é mapear as classes como not-lazy. No entanto esta hipótese deverá ser devidamente considerada uma vez que tem um impacto elevado em termos de comportamento. Para o caso que estamos a mapear neste artigo, supondo que todas as classes fossem mapeadas como não "lazy" isto implicaria que quando se obtivesse uma edição o NHibernate teria de carregar também o artigo de capa, visto não poder colocar uma proxy em vez da instância real. Poderia não ser grave, no entanto para classes mais complexas facilmente se notariam problemas graves de performance uma vez que estaríamos sempre a carregar todos os dados da base de dados, mesmo em situações em que não sejam usados.

Outro ponto a ter em atenção é que lazy-loading de classes não é o mesmo que lazy-loading de colecções. Ou seja, no caso da Edicao, teria de ser carregado o tema de capa, mas a colecção Artigos iria manter-se por carregar até ser iterada, se a classe Artigos não fosse lazy.

Colecções em NHibernate

Podem reparar que as classes estão criadas não com List mas sim com IList e ISet. Depois da discussão relativa ao lazy-loading e proxies torna-se mais facilmente compreensível esta escolha.

Quando um objecto é carregado, por omissão apenas é car-

regado o objecto em si da base de dados. Para suportar lazy -loading das colecções de relações, as instâncias que são colocadas nos membros são também proxies de colecções.

Existem 4 tipos primários de colecções representados em NHibernate:

- Uma **List** (lista) tem o comportamento de uma lista ordenada e é mapeada como IList.
- Um **Map** (mapa) é um conjunto de pares chave, valor e é mapeado como IDictionary.
- Um **Set** (conjunto) é um conjunto não ordenado que permite repetições, mapeado como ISet (não é um tipo nativo sendo que está disponível na biblioteca Iesi.Collections).
- Um **Bag** (saco), como um Set é um conjunto não ordenado de entidades que permite repetições. Não existe um interface específico para este tipo de colecção, sendo que pode ser mapeado para IList ou ICollection.

O comportamento típico das colecções em aplicações mapeia-se em conjuntos, visto a ordem não ser relevante. Caso a ordem seja relevante, deve mapear-se como uma lista e indicar qual a coluna que guarda o índice que define a posição do item na lista.

Alterações às entidades e aos mapeamentos

Agora que já temos uma melhor compreensão do modo de funcionar de mapeamentos e colecções, vamos alterar as nossas entidades para passarem a ter todas as propriedades como virtual:

```
public class Edicao
{
 public virtual DateTime DataEdicao { get; set; }
 public virtual Artigo TemaDeCapa { get; set; }
 public virtual IList<Artigo> Artigos { get;
set; }
}

public class Artigo
{
 public virtual int Id { get; set; }
 public virtual string Titulo { get; set; }
 public virtual ISet<Autor> Autores { get; set; }
}

public class Autor
{
 public virtual int Id { get; set; }
 public virtual string Nome { get; set; }
 public virtual string Email { get; set; }
 public virtual ISet<Artigo> Artigos { get;
set; }
}
```

Uma lista, ao contrário de um saco, é uma sequência ordenada de elementos. Como tal, precisamos de indicar ao NHibernate qual a coluna que guarda o índice de cada elemento na lista. Para isso vamos afinar o nosso mapeamento para a classe **Edicao**:

```
public class EdicaoOverride : IAutoMappingOver-
ride<Edicao>
{

public void Override(AutoMapping<Edicao> mapping)
  {
    mapping.Id(edicao => edicao.DataEdicao)
           .Column("DataEdicao")
           .GeneratedBy.Assigned();
    mapping.HasMany(e => e.Artigos)
           .AsList(index => index.Column
                            ("PosicaoNaRevista"));
   }
}
```

Aqui estamos a indicar que a colecção Artigos da entidade Edicao é uma lista (por omissão é um saco) cujo índice de cada elemento é armazenado na coluna PosicaoNaRevista. De reparar que esta coluna não se encontra na entidade Artigo, sendo um detalhe de implementação da coleção Artigos.

Neste momento, ao correr o programa vemos que o NHibernate cria as tabelas "Artigo", "ArtigosToAutores", "Autor", "Edicao" e "HiLo", com as colunas respectivas:

```
create table "Artigo" (
 Id INTEGER not null,
 Titulo TEXT,
 Edicao_id DATETIME,
 PosicaoNaRevista INTEGER,
 primary key ( Id )
)
create table ArtigosToAutores (
 Artigo_id INTEGER not null,
 Autor_id INTEGER not null,
 primary key ( Autor_id, Artigo_id )
)
create table "Autor" (
 Id INTEGER not null,
 Nome TEXT,
 Email TEXT,
 primary key ( Id )
 )
create table "Edicao" (
 Id DATETIME not null,
 TemaDeCapa_id INTEGER,
 primary key ( Id )
)
create table HiLo (
 Hi INTEGER
)
```

Por omissão as chaves estrangeiras são no formato "<Entidade>_id", mas, mais uma vez este comportamento é reflexo de uma convenção que pode ser alterada da mesma forma como foram alteradas as convenções para os identificadores.

Criar e gravar objectos

Para interagir com a base de dados NHibernate implementa

o padrão Unit-Of-Work através do interface ISession. Este interface fornece métodos para carregar, gravar e apagar entidades, fazer queries à base de dados, iniciar transacções, entre outros.

Para gravarmos uma Edicao, basta abrirmos uma ISession a partir da ISessionFactory criada previamente:

```
using (var session =sessionFactory.OpenSession())

using (var transaction = session.BeginTransaction
())
{
    var edicao = new Edicao {DataEdicao =
                    new DateTime(2011, 07, 01)};

    session.Save(edicao);
    transaction.Commit();
}
```

Como boa prática todas as operações feitas com NHibernate devem correr no contexto de uma transação. Caso não exista nenhuma, o NHibernate cria uma transação para cada operação, sendo este comportamento apenas uma salvaguarda de último recurso e que deve ser considerado (na grande maioria dos casos) um erro.

Se colocarmos um breakpoint após a linha session.Save (edicao) e observarmos a comunicação com o servidor de base de dados, reparamos que, embora já tenha sido indicado que queremos gravar o objecto, ainda não foi executado nenhum comando de SQL. Isto acontece porque as alterações registadas na sessão apenas serão comunicadas quando a transacção for confirmada. Caso seja abortada, as entidades não chegam sequer a ir à base de dados.

Inclusivamente nalgumas bases de dados (MS SQL Server, por exemplo) é possível indicar através do batch-size que queremos efectuar vários comandos na mesma ligação, minimizando os roundtrips à base de dados e aumentando, dessa forma, a taxa de transferência (throughput) da aplicação.

No entanto existem casos em que um .*Save* vai incorrer numa comunicação à base de dados. Por exemplo, se o objecto tiver um identificador gerado por Identity na base de dados, o NHibernate vai gravar imediatamente o objecto para lhe atribuir o identificador. Esta é uma das razões pelas quais não é recomendado usar este tipo de identificadores com um ORM.

Padrões de uso de ISession

Uma instância de ISession é um objecto que representa uma unidade de trabalho atómica, e como tal, deverá ter um tempo de vida curto.

No caso de uma aplicação web, o mais tradicional é obter uma sessão por pedido HTTP. A criação do objecto de sessão tem um custo ínfimo e por si só não efectua nenhuma ligação à base de dados até que sejam feitas operações sobre a sessão. Como tal acaba por ser usual criar sempre uma sessão contextual ao pedido no início deste e fechá-la no final do mesmo.

No caso de aplicações desktop, o padrão aconselhado costuma ser de uma sessão por ecrã, que é criada quando um ecrã é aberto e fechada quando o utilizador fechar o ecrã.

Gravação de várias entidades

O exemplo anterior serviu para mostrar um caso simples. Vamos agora analisar um caso ligeiramente mais complicado:

```
using (var session = sessionFactory.OpenSession())
using (var transaction=session.BeginTransaction())
{
    var autorPrimeiroArtigo = new Autor {Email =
      "ja@qu.im", Nome = "Jaquim"};
    var autorSegundoArtigo = new Autor {Email =
      "ma@nu.el", Nome = "Manuel"};

    var primeiroArtigo = new Artigo
       {
          Autores = new HashedSet<Autor>
            {autorPrimeiroArtigo},
          Titulo = "Introdução ao NHibernate"
        };

    var segundoArtigo = new Artigo
        {
          Autores = new HashedSet<Autor>
            {autorSegundoArtigo},
          Titulo = "Visual (not) Basic"
                          };
    var terceiroArtigo = new Artigo
          {
            Autores = new HashedSet<Autor>
                {autorSegundoArtigo},
            Titulo = "VB e NHibernate"
          };

    var edicao = new Edicao()
          {
          DataEdicao =
              new DateTime(2011, 07, 01),
          Artigos = new List<Artigo>
            {
              primeiroArtigo,
              segundoArtigo,
              terceiroArtigo
            },
           TemaDeCapa = segundoArtigo,
           };

    session.Save(edicao);
    transaction.Commit();
}
```

Neste exemplo estamos a criar dois artigos, com um autor cada um, e a adicioná-los à lista de artigos da edição. No entanto, quando é feita a confirmação da transação ocorre o seguinte erro:

> *object references an unsaved transient instance - save the transient instance before flushing. Type: artigo_1.nhibernate.Entidades.Artigo, Entity: arti*

go_1.nhibernate.Entidades.Artigo

Este erro ocorre porque a entidade Edicao refere duas entidades Artigo que, por não terem sido explicitamente, gravadas são consideradas pelo NHibernate como transientes (ou seja, existem apenas no grafo de objectos do Nhibernate mas ainda não foram persistidas na base de dados).

Cascades

Quando duas entidades estão relacionadas, como o caso de **Edicao** e **Artigo**, é comum o ciclo de vida das mesmas também estar relacionado. Quando se adiciona ou altera um artigo de uma edição e se grava, pretende-se que o artigo seja também gravado, visto estar associado à edição.

Para responder a este requisito existe a funcionalidade de Cascade, semelhante ao Cascade de SQL, mas que funciona de forma independente do mesmo, sendo que em NHibernate existem as seguintes opções:

- **none** - não é efectuado nenhum cascade, ficando a nosso cargo remover e adicionar as entidades
- **save-update** - quando o objecto é adicionado ou actualizado, as associações marcadas com este cascade são verificadas e gravadas ou adicionadas caso seja necessário
- **delete** - quando o objecto é removido, remove todos os objectos que estão associados com este cascade
- **delete-orphan** - mesmo comportamento que delete, com a adição de que objectos que sejam removidos da associação são apagados.
- **all** - comportamento de save-update juntamente com o comportamento de delete
- **all-delete-orphan** - comportamento de save-update juntamente com o comportamento de delete-orphan

É preciso ter cuidado com a configuração de cascades que efectuam remoções, em particular se definidos por convenções, por serem transitivos. Isto quer dizer que é fácil configurar mapeamentos que ao remover determinado objecto, façam com que sejam removidos, não intencionalmente, todo um conjunto de outros objectos relacionados.

Nova convenção para cascades

No nosso exemplo, vamos querer que o comportamento por omissão para cascade seja de save-update. Para tal, basta adicionar a seguinte convenção:

```
public class CascadeSaveOrUpdateConvention :
ICollectionConvention
{
    public void Apply(ICollectionInstance in-
stance)
    {
        instance.Cascade.SaveUpdate();
    }
}
```

Desta forma todas as coleções utilizam o mesmo tipo cascade (a não ser que seja explicitamente definido outro tipo de cascade através de um override).

Carregar uma entidade

Agora que já temos dados na base de dados, vamos querer começar a ir busca-los. O caso mais simples é carregar uma edição pela chave:

```
var edicao = session.Get<Edicao>(new DateTime
(2011, 07, 01));
```

O NHibernate vai buscar o objecto à base de dados, sendo que retorna null no caso em que o mesmo não exista. No entanto, existe outra forma de carregar dados por chave:

```
var edicao = session.Load<Edicao>(new DateTime
(2011, 07, 01));
```

Com o método .Load ele retorna um proxy do objecto que apenas resulta num hit à base de dados, se for necessário (tal como discutimos para o lazy-loading de colecções). Isto quer dizer que no seguinte exemplo não é efectuado nenhum select, visto não ser necessário para atribuir o identificador ao *.TemaDeCapa*:

```
new Edicao
    {
        DataEdicao = new DateTime(2011, 07, 01),
        TemaDeCapa = session.Load<Artigo>(1)
    };
```

No entanto é preciso ter a certeza que o identificador existe, caso contrário é lançada uma excepção quando se tenta aceder ao campo ou se tenta gravar na base de dados (no caso de relações com chaves estrangeiras).

Queries

O NHibernate tem várias formas de efectuar queries à base de dados:

1) Através de Linq:

```
IList<Edicao> edicoesDe2011 =
   session.Query<Edicao>()
      .Where(e => e.DataEdicao >=
              new DateTime(2011, 1, 1)
      && e.DataEdicao < new DateTime(2012, 1, 1))
    .ToList();
```

Aqui temos um interface de Linq para as nossas entidades, onde podemos usar os tradicionais .Where, .Skip, .Take, entre outros. O NHibernate converte o query de Linq em SQL.

2) Usado uma linguagem de queries sobre objectos chamada HQL:

```
var edicoesDe2011 = session.CreateQuery(
  "from Edicao where DataEdicao >= :EsteAno and
DataEdicao < :ProximoAno")
    .SetDateTime("EsteAno",
           new DateTime(2011, 1, 1))
    .SetDateTime("ProximoAno",
           new DateTime(2012, 1, 1))
    .List<Edicao>();
```

Esta é uma forma nativa – e bastante poderosa - que existe desde as primeiras versões do NHibernate, para fazer queries sobre objectos persistidos com uma sintaxe semelhante a SQL.

Este mecanismo, embora poderoso, não facilita a composição de queries, por serem strings. Pela mesma razão não aproveita as funcionalidades de C# ou VB para apoiar os programadores através de auto-complete e verificações em tempo de compilação.

3) A terceira hipótese é a utilização de *ICriteria*:

```
var edicoesDe2011 = session.CreateCriteria<Edicao>
()
    .Add(Restrictions.Ge("DataEdicao",
           new DateTime(2011, 1, 1)))
    .Add(Restrictions.Lt("DataEdicao",
           new DateTime(2012, 1, 1)))
    .List<Edicao>();
```

Esta solução é igualmente expressiva, sendo que a principal vantagem face a HQL é a sua flexibilidade para composição de partes das queries, reduzindo a duplicação de código e possibilitando a criação de algumas abstrações de negócio sobre as queries. No entanto continua a utilizar strings para identificar as propriedades das entidades, o que propicia erros não só quando se escrevem queries, mas também devido a esquecimentos aquando de refactorizações.

4) A hipótese mais recente chama-se de QueryOver e junta a expressividade de ICriteria à tipicação de Linq:

```
var edicoesDe2011 = session.QueryOver<Edicao>()
    .Where(e => e.DataEdicao >=
           new DateTime(2011, 1, 1)
    && e.DataEdicao <
           new DateTime(2012, 1, 1))
    .List();
```

QueryOver é uma camada construída em cima de ICriteria que usa reflexão estática para tirar partido da tipificação de C# e fornecer uma utilização mais intuitiva e menos propensa a erros, mantendo a expressividade de ICriteria. Uma vez que QueryOver expõe o ICriteria que está a ser construído através da propriedade.UnderlyingCriteria, é sempre possível usar o mesmo quando as operações que pretendemos efectuar não se encontram suportadas pela camada de abstração.

Embora o interface para casos simples seja muito parecido ao fornecido pelo interface de Linq, internamente usam aproximações diferentes.

5) Finalmente é sempre possível fazer queries directamente em SQL:

```
var edicoesDe2011 = session.CreateSQLQuery(
    @"
select edicao.DataEdicao
from   Edicao edicao
where  DataEdicao >= :EsteAno and DataEdicao
< :ProximoAno")
   .SetDateTime("EsteAno",
           new DateTime(2011, 1, 1))
   .SetDateTime("ProximoAno",
           new DateTime(2012, 1, 1))
   .SetResultTransformer
     (Transformers.AliasToBean<Edicao>())
   .List<Edicao>();
```

Aqui é preciso ter em atenção à forma como os dados que vêm da base de dados são interpretados pelo NHibernate. Como se está a construir simplesmente um query sq, o NHibernate não sabe à priori qual o mapeamento entre colunas e propriedades ou classes, pelo que precisamos de lhe indicar como deve interpretar é que os objectos vão ser interpretados.

Neste exemplo estamos a ir buscar apenas o Id da edição e usamos um ResultTransformer chamado AliasToBean que olha para as colunas que vêm no resultado da query e mapeia-se como propriedades do tipo de objecto que lhe é passado. Assim, as instâncias resultantes não são entidades completamente hidratadas mas sim simples DTOs ou View-Models, usados em casos de leitura de dados com queries optimizadas.

Queries mais avançadas

Nos exemplos mostrados na secção anterior realizámos queries simples que apenas operam sobre a entidade. No entanto é igualmente possível operar sobre mais do que uma entidade, percorrendo as colecções através de joins. Vejamos o seguinte exemplo, que retorna todas edições cujo tema de capa inclui a palavra "Basic":

```
var edicoesComBasicNaCapa =
      session.QueryOver<Edicao>()
            .JoinQueryOver(e => e.TemaDeCapa)
            .WhereRestrictionOn(a =>
                  a.Titulo).IsLike("%Basic%")
            .List();
```

Aqui estamos a fazer o join entre **Edicao** e **Artigo**, através da relação TemaDeCapa. A funcionalidade de Like da base de dados é disponibilizada com a indicação de qual o campo no qual se quer operar (Titulo) seguido da operação (IsLike).

Finalmente, indicamos que queremos uma lista destes valores, de forma a fazer o pedido à base de dados. Outra possibilidade seria não efectuar já o pedido com List e apenas obter um proxy para a reposta, através do método .Future. Desta forma seria possível preparar vários queries e efectuáefectuá-los todos num só pedido à base de dados.

Se apenas quisssemosquiséssemos obter o numero de edições, bastaria chamar o método .RowCount (ou .RowCountInt64 se tivermos mesmo muitas edições) em vez de .List. Nesse caso o query apenas retorna o count total de linhas em vez das entidades.

Um exemplo mais complexo seria para obter todas as edições que têm artigos do Manuel:

```
Artigo artigo = null;
Autor autor = null;

var edicoesComArtigosDoManuel = sessi-
on.QueryOver<Edicao>()
    .JoinQueryOver(edicao => edicao.Artigos,
            () => artigo)
    .JoinQueryOver(() => artigo.Autores,
            () => autor)
    .Where(() => autor.Nome == "Manuel")
    .List();
```

Aqui para além dos .JoinQueryOver temos também duas variáveis inicializadas a null e usadas apenas em expressões lambda. Esta é a forma usada pelo mecanismo QueryOver para declarar nomes para cada uma das coleções referenciadas pelos joins. As variáveis artigo e autor apenas servem como mnemónicas, e não são usadas na execução do código. Caso precisemos de declarar outros tipos de join, podemos prefixar a chamada ao método com Inner, Left ou Right (Inner.JoinQueryOver(() => artigo.Autores) por exemplo).

O SQL gerado não revela nenhuma surpresa:

```
SELECT this_.DataEdicao as DataEdicao3_2_,
      this_.TemaDeCapa_id as TemaDeCapa2_3_2_,
      artigo1_.Id as Id0_0_,
      artigo1_.Titulo as Titulo0_0_,
      autores5_.Artigo_id as Artigo1_,
      autor2_.Id as Autor2_,
      autor2_.Id as Id2_1_,
      autor2_.Nome as Nome2_1_,
      autor2_.Email as Email2_1_
FROM "Edicao" this_
      inner join "Artigo" artigo1_
            on this_.DataEdicao = arti-
go1_.Edicao_id
      inner join ArtigosToAutores autores5_
            on artigo1_.Id = autores5_.Artigo_id
      inner join "Autor" autor2_
            on autores5_.Autor_id = autor2_.Id
WHERE autor2_.Nome = 'Manuel'
```

No entanto, ao executarmos este query sobre os dados criados previamente, iríamos encontrar resultados duplicados na lista retornada. Se executarmos o query de sql directamente na base de dados o motivo torna-se mais claro, visto o query retornar duas linhas com a mesma entidade, e o NHibernate não saber se a duplicação é intencional.

Uma hipótese é usar o transformador Transformers.DistinctRootEntity, que desduplica as entidades quando interpreta os resultados vindos da base de dados. No entanto esta solução encontra problemas quando é usada em conjunto com paginação. Como as linhas na base de dados se encontram duplicadas e os limites estão a contabilizar as mesmas duplicações, ao ser feito um pedido de 10 itens acabamos por poder ter menos itemsitens na lista resultante, um pedido de 10 itens pode vir com menos.

Por todos estes motivos, a solução mais correcta passa por passar a condição para uma subquery (o que permite aplicar limites sobre as linhas das entidades raiz):

```
Artigo artigo = null;
Autor autor = null;

var edicoesComArtigosDoManuel =
      session.QueryOver<Edicao>()
      .WithSubquery.WhereProperty(edicao =>
            edicao.DataEdicao)
    .In(QueryOver.Of<Edicao>()
      .JoinQueryOver(a => a.Artigos,
            () => artigo)
      .JoinQueryOver(() => artigo.Autores,
            () => autor)
            .Where(() => autor.Nome == "Manuel")
            .Select(a => a.DataEdicao))
    .TransformUsing
(Transformers.DistinctRootEntity)
    .List();
```

Aqui a condição existe na subquery, evitando-se assim resultados duplicados na query principal, que é retornada.

Para definir subqueries usam-se DetachedQueryOver (existindo um correspondente DetachedCriteria por baixo desta implementação), que são criados para uma entidade através do método QueryOver.Of. O builder devolvido por este método tem um comportamento idêntico ao QueryOver da ISession, com a diferença de que lida com queries não ligadas a nenhuma sessão (dai o nome de detached). Estas

queries podem então servir de argumento para subqueries e utilizadas para composição com outras queries.

Resumo e Próximos passos

Neste momento já temos os conceitos-base de NHibernate suficientes para podemos responder aos casos de uso mais simples. Vimos como mapear classes com o FluentNHibernate, como persistir as nossas instâncias e como obtê-las novamente da base de dados.

Existem ainda bastantes tópicos que ficam por abordar, tais como eager loading, projecções, interceptors, entre outros que serão abordados nos próximos artigos.

Referências

http://www.nhforge.org/ - NHForge – site oficial da comunidade de Nhibernate:

http://knol.google.com/k/fabio-maulo/nhibernate/1nr4enxv3dpeq/21# - Knol do FabioFábio Maolo sobre NHibernate

http://fluentnhibernate.org/ - Site da biblioteca FluentNHibernate

https://groups.google.com/forum/#!forum/nhusers - Mailing List oficial de NHibernate

http://nhprof.com/ - Ferramenta de profiling de NHibernate

Livro

NHibernate 3.0 Cookbook, Jason Dentler, Packt, ISBN 13 : 978-1-84951-304-3 – Livro com várias receitas para problemas comuns (e outros nem tanto) encontrados por utilizadores de NHibernate

AUTOR

Escrito por Bruno Lopes

Bruno Lopes fez o curso de Engenharia Informática no IST, e neste momento conta com mais de 5 anos de experiência profissional em IT, em particular nas áreas de desenvolvimento de software web-based. Actualmente é co-fundador da weListen Business Solutions, trabalhando no produto InnovationCast, e participa activamente na comunidade NetPonto, apresentando temas como NHibernate, RavenDB ou IoC.

Tem blog em http://blog.brunomlopes.com, twitter em @brunomlopes e linkedin em http://pt.linkedin.com/in/brunomlopes

Sandboxed Solutions em SharePoint Online 2010

Com o lançamento na Microsoft da sua solução Cloud através do Office 365 tem havido muita procura sobre as diferentes ofertas que este novo serviço disponibiliza.

Office 365 é um conjunto de Serviços da Microsoft na qual se pode aceder em qualquer local a documentos, correio electrónico, contactos, calendários de forma actualizada, tendo como garantia a fiabilidade e segurança a nível empresarial de forma a disponibilizar a pequenas e grandes empresas com um conjunto de serviços essenciais de trabalho a um preço acessível.

Após o lançamento da versão Beta do Office 365 um dos produtos no qual foquei a minha atenção foi o SharePoint Online 2010 e a capacidade de criar Sandbox Solutions.

Sandbox Solutions em SharePoint é um novo conceito para SharePoint 2010 que permite criar código personalizado a aplicar nos sites de SharePoint de forma segura, fornece a capacidade dos Site Collections Administrator de instalar e monitorizar soluções personalizadas sem a necessidade dos Administradores do SharePoint. Mas para ter essa liberdade os Administradores estão limitados as funcionalidades das Sandboxed Solutions, relativo ao código que podem utilizar ou os recursos que podem utilizar/aceder.

Soluções em SharePoint Sandbox

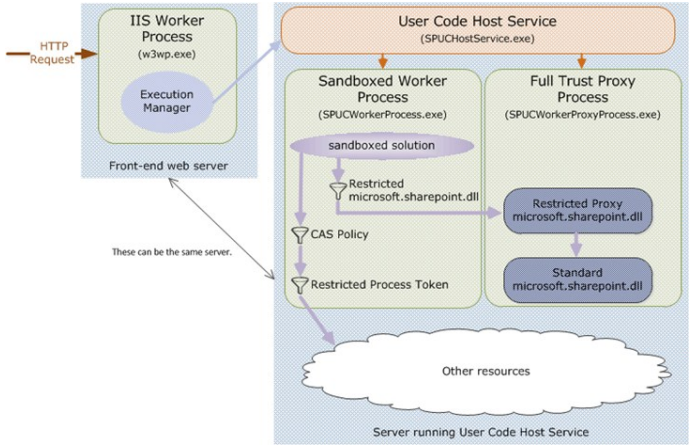
As soluções em SharePoint é um pacote de configurações em XML, assemblies e ficheiros de apoio “.jpg, .png, .resx” entre outros que sao compiladas em um ficheiro tendo como extensão [nome].wsp.

A Soluções em Sandbox utiliza um subconjunto da namespace Microsoft.SharePoint do Object Model do SharePoint sendo considerados seguros e acessíveis de forma a poderem ser utilizados na criação das nossas Soluções.

Todas as Soluções são guardadas em Galerias “Solutions” nos Sites do SharePoint onde os administradores dos sites podem depositar e gerenciar soluções de modo seguro.

Estas soluções são executadas num processo separado do “w3wp.exe” de forma a ter menor impacto possível na execução do código criado nas nossas Sandbox Solutions.

Este novo serviço chamado “SharePoint 2010 User Code Host”, irá ser a zona onde será executado e compilada as soluções em Sandbox de uma forma segura a ter o menor impacto possível em toda a Infra-estrutura do SharePoint.



A diferença das Sandbox Solution para Soluções em Farm é a camada de acesso as diferentes Assembleis e níveis de permissão para sua infraestrutura enquanto as Sandbox Solutions encontram-se muito limitadas a camada dos Sites e Webs, as Farm Solutions fornece um acesso alargado ao conteúdo da infra-estrutura no SharePoint 2010 que vão desde aos serviços das Farm as das Web Application .

Os próximos links listam as capacidades/limitações das Sandbox Solutions.

What Can Be Implemented in Sandboxed Solutions in SharePoint 2010
http://msdn.microsoft.com/en-us/library/gg615464.aspx

Restrictions on Sandboxed Solutions in SharePoint 2010
http://msdn.microsoft.com/en-us/library/gg615454.aspx

Microsoft.SharePoint.dll APIs That Are Available from Sandboxed Solutions
http://msdn.microsoft.com/en-us/library/ee537860.aspx

Visual Studio 2010 Sandbox Solution

Neste exemplo foi criado uma Sandbox Solution em que foi utilizado os seguintes elementos para configurar o nosso Site no SharePoint Online.

Utilizando o Visual Studio 2010 foi criado novo Projecto em SharePoint 2010 para Sandbox que pode ser implementado em Sites do SharePoint Online 2010 site chamado “SandoBoxProjecto” com os seguintes elementos:

- Visual Sandbox Web Part
- Master Page
- Custom Actions
- SharePoint Ribbons
- List Definition

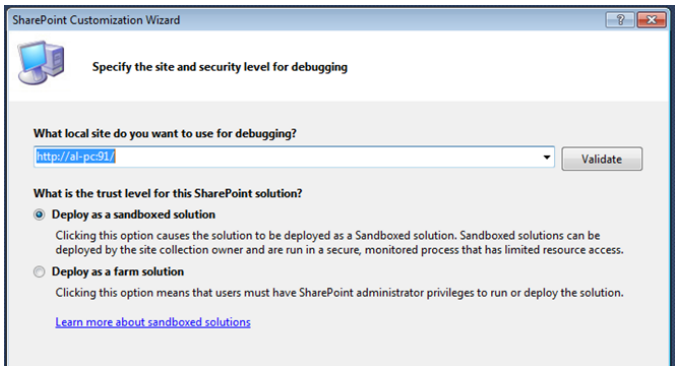
Templates para Visual Studio 2010 de apoio para a criação deste exemplo:

Visual Studio 2010 SharePoint Power Tools
http://visualstudiogallery.msdn.microsoft.com/8e602a8c-6714-4549-9e95-f3700344b0d9/

CKS: Development Tools Edition
http://cksdev.codeplex.com/

SharePoint 2010 Extensibility Projects
http://archive.msdn.microsoft.com/vsixforsp

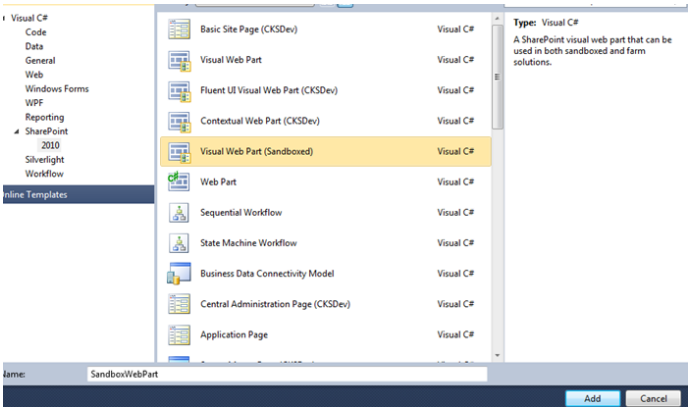
Criar uma nova solução no Visual Studio 2010 e criar novo Projecto para SharePoint 2010, após novo Projecto definir "Empty project" e definir no SharePoint Wizard a opção "Deploy as a Sandboxed Solution", esta opção irá validar a solução a criar para as Sandbox Solutions.



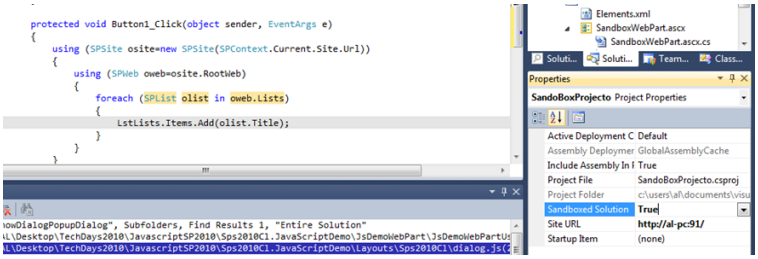
Sandbox Web Part

A funcionalidade tem como acção demostrar como o Template "Visual Studio 2010 SharePoint Power Tools" pode ser utilizado para criar uma Visual Web Part para sandbox Solutions.

Nesta solução criei uma WebPart para listar o nome das listas numa Listbox.

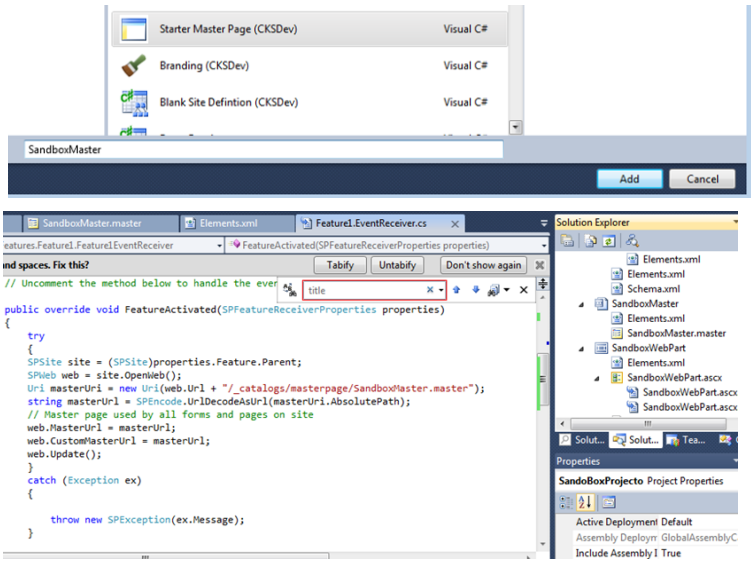


O template fornece a capacidade de visualizar os componentes ASP.Net num User Control onde será desenhada a nossa WebPart.



MasterPages

Neste exemplo foi criado uma nova componente para alterar a MasterPage e aplicar como default assim que o componente for activo.



Para activar alterar a MasterPage foi criado uma "Feature receiver" para copiar e definir a MasterPage "SandboxMaster.master" como a pré-definida.

```
public override void FeatureActivated
(SPFeatureReceiverProperties properties)
{
  try
  {
    SPSite site = (SPSite)
          properties.Feature.Parent;
    SPWeb web = site.OpenWeb();
    Uri masterUri = new Uri(web.Url +
    "/_catalogs/masterpage/SandboxMaster.master");
    string masterUrl = SPEncode.UrlDecodeAsUrl
         (masterUri.AbsolutePath);

    // Master page used by all forms
    // and pages on site
    web.MasterUrl = masterUrl;
    web.CustomMasterUrl = masterUrl;
    web.Update();
  }
  catch (Exception ex)
  {
    throw new SPException(ex.Message);
  }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/
sharepoint/">
<CustomAction
Id="UserInterfaceCustomActions.ECBItemToolbar"
Rights="EditListItems"
Location="EditControlBlock"
RegistrationType="ContentType"
RegistrationId="0x01"
Sequence="550"
Title="Test"
Description="Test Test" >

<UrlAction Url="javascript:SP.UI.ModalDialog.Show
PopupDialog('/_layouts/settings.aspx');"/>
</CustomAction>
</Elements>
```

Custom Actions

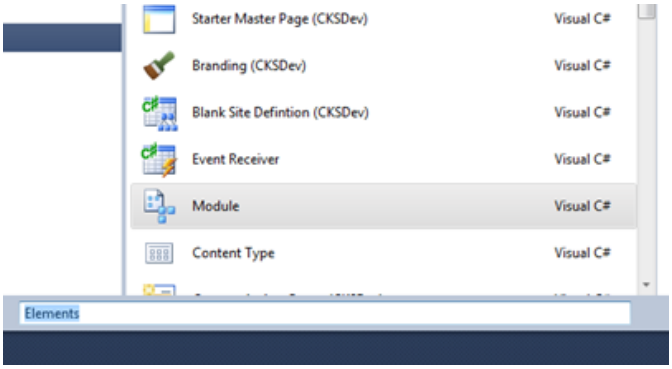
A funcionalidade tem como acção demostrar como o Module “Elements” foi alterado para adicionar novas acções nos menus das List item através da utilização de Content Types.

SharePoint é constituído por Content Types, estes Content Types ajudam a definir a Metadata associada a cada Tipo de Listas, seleccionado a propriedade e a sua interdependência entre os diversos Content Types que estão constituídos na lista.

Este “Custom action” tem como função definir uma acção que seja visível em qualquer tipo de Lista para isso foi definido o Content Type “Item” onde todos os Content Types para listas derivam.

Exemplo:

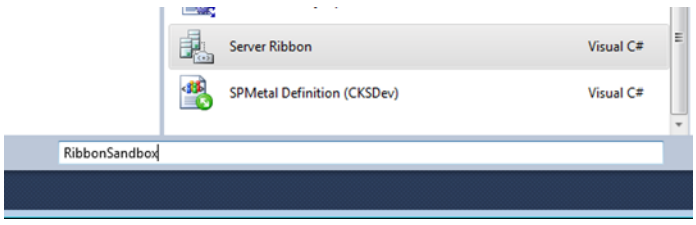
http://grounding.co.za/blogs/brett/archive/2008/09/09/sharepoint-content-type-id-s.aspx

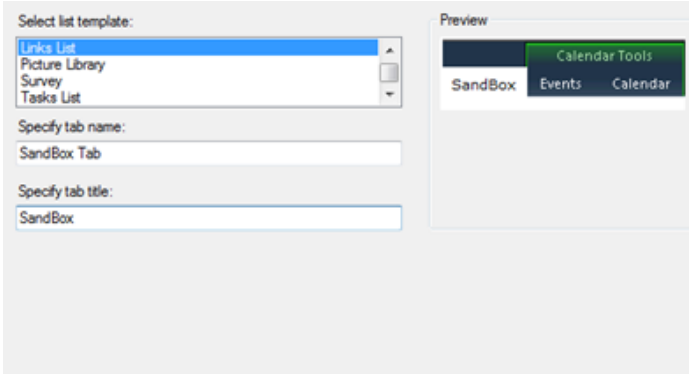


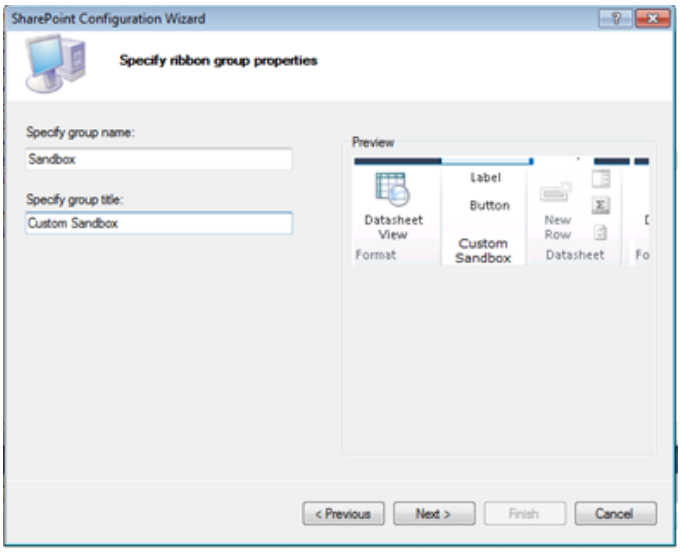
SharePoint Ribbons

A funcionalidade tem como acção demostrar como o Template “Server Ribbon” pode ser utilizado para personalizar o controlo de Ribbon para SharePoint 2010.

Este template disponibiliza um conjunto de customizações pré-definidas para a criação do Ribbons em SharePoint 2010 de uma forma simples.

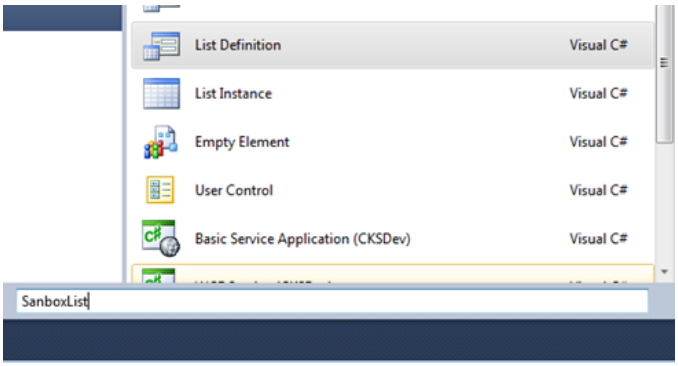


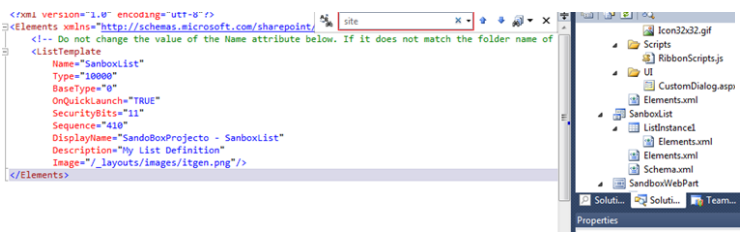




List Definition

Esta funcionalidade ira criar uma nova Lista com campo Title, esta lista poderá ser utilizada ou alterada para registar os erros criadas pelo código das features em sandbox.

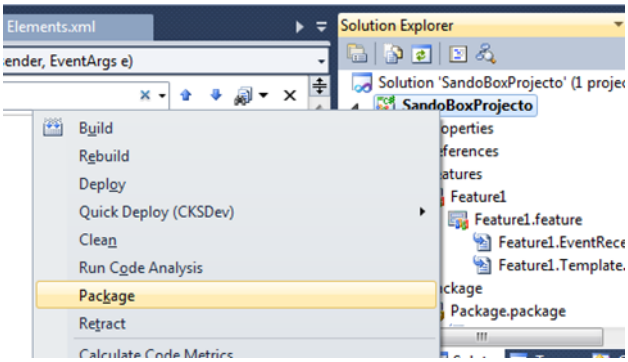


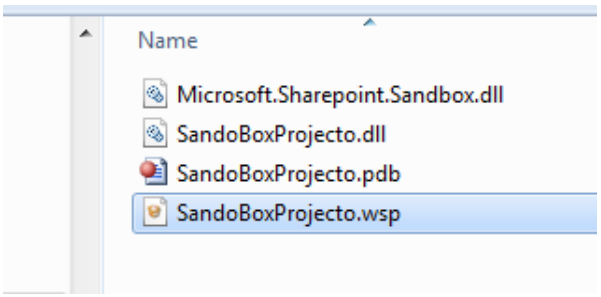


Package em SharePoint

O Package das soluções SharePoint é a pacote de funcionalidades que são entregues ao cliente onde se encontras as novas configurações que o Administrador da Site Collection poderá monitorizar e gerenciar as diversas Sandboxed solutions.

Para criar a Solução em SharePoint no Visual Studio 2010 deverá seleccionar a opção “Package” para criar a solução “SandboxProjecto.wsp”.

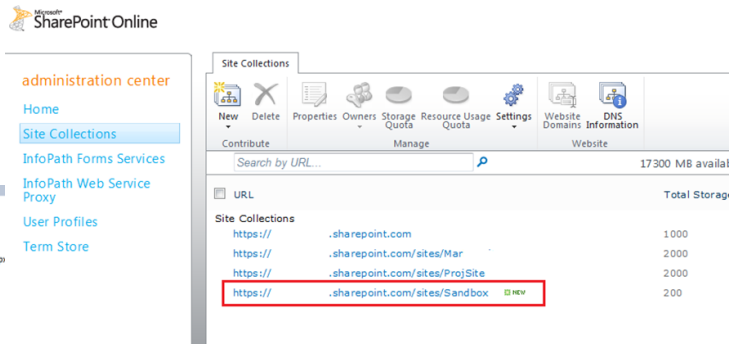




SharePoint Online Administration Center

O Office 365 disponibiliza ao administrador dos Sites SharePoint Online um novo interface onde poderá gerir as diversas Site Collections, o espaço associado e total de recursos alocados a essas máquinas.

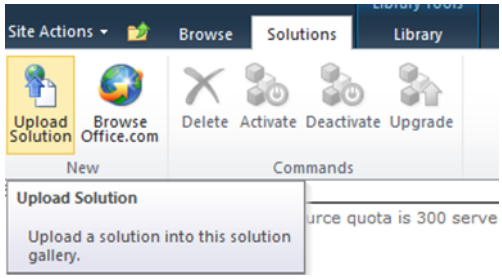
Para a nova solução foi criada uma nova Site Collection chamada “Sandbox”.

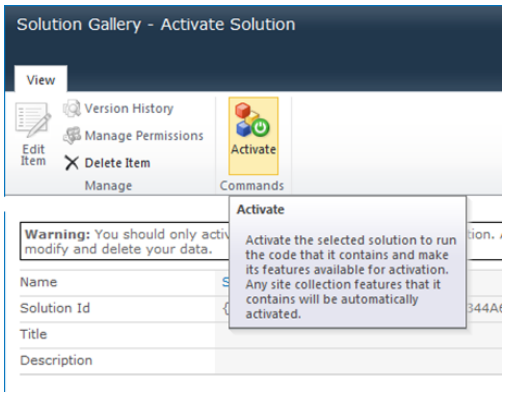


Instalação da Sandbox

Aceder ao site do SharePoint Online criado neste caso “Sandbox” e seleccionar “Site Actions > Site Settings > Galleries > Solutions” seleccionar opção “Solutions” no Ribbon e seleccionar “Upload Solution” e seleccionar o caminho para a solução “SandBoxProjecto.wsp”.

Assim que o projecto esteja no Site em SharePoint



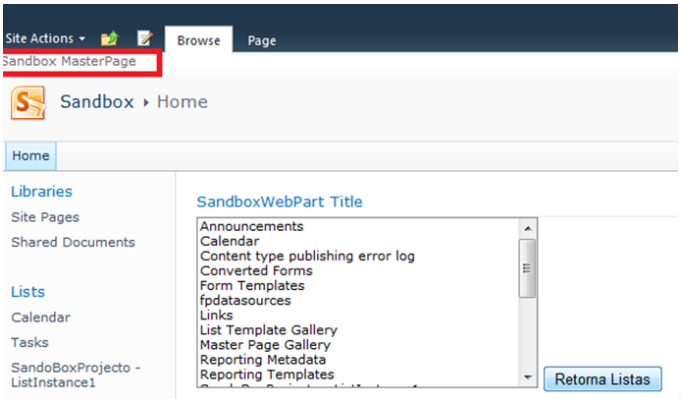


MasterPage e Sandbox Web Part

Após a activação da nossa solução "SandBoxProjecto.wsp" podemos começar a validar as alterações realizadas no Site do SharePoint Online.

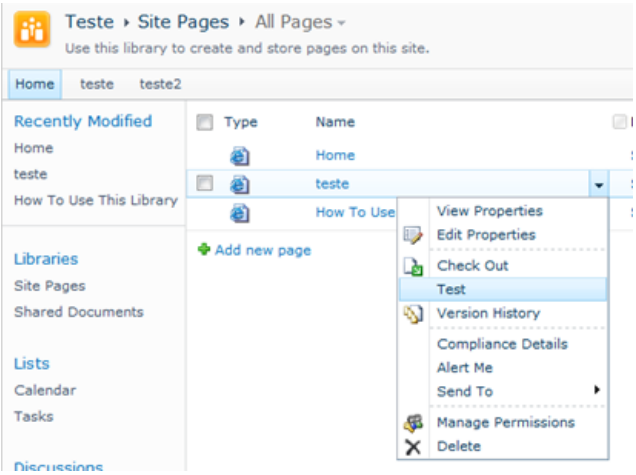
Quando aceder a página principal do Site, no topo esquerdo encontra-se uma mensagem "Sandbox MasterPage" para identificar que a MasterPage foi alterada correctamente pela customizada na solução através da solução "SandboxMaster"

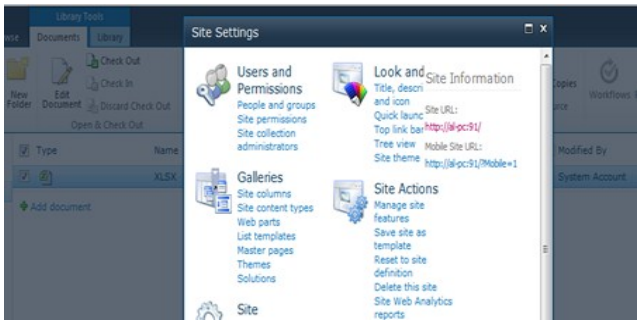
Para aceder a Web Part "Sandbox WebPart" devera ser adicionada a Web Part, para isso devemos aceder a Tab "Page" seleccionar botão "Edit Page" seleccionar opção para adicionar nova Web Part e na janela de diálogo aceder ao Grupo "Custom" e seleccionar Web Part "Sandbox WebPart".



Sandbox Custom Actions

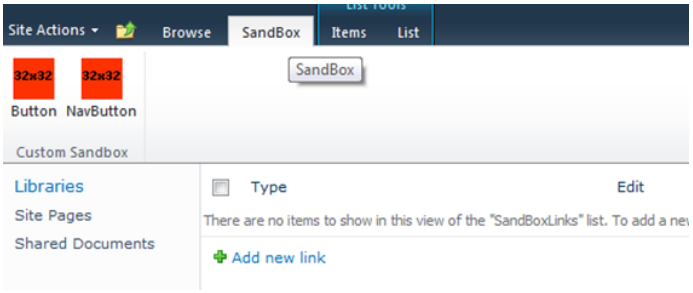
Para visualizar as novas opções do Modulo "Elements" devemos seleccionar uma Lista no SharePoint Online e seleccionar menu dos item dessa Lista, ao seleccionar ira aparecer uma nova opção chamada "Test", esta nova opção irá criar um nova janela onde ira aparecer o Site de Administração do Site em SharePoint Online.





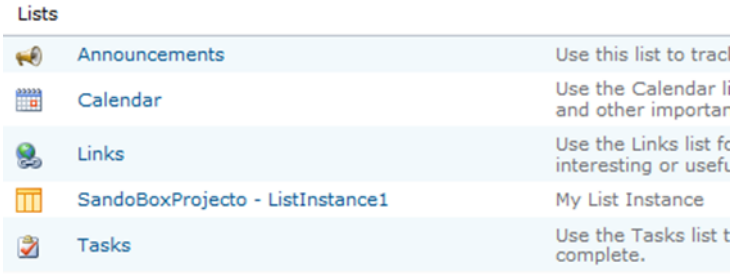
Sandbox Ribbons

Para visualizar as novas opções da solução "RibbonSandbox" devemos seleccionar uma Lista de Links no SharePoint Online, ai ira aparecer uma nova Tab com o nome "Sandbox", ao seleccionar aparece um grupo chamado "Custom Sandbox" com duas novas opções que ira despoletar acções de mensagem.



List Definition

Para visualizar as novas opções da solução "RibbonSandbox



Download da Solução "SandBoxProjecto.zip"

Existem muitos outros componentes que podem ser utilizados na criação das nossas Sandbox Solution aqui ficam uma lista a usar.

What Can Be Implemented in Sandboxed Solutions in SharePoint 2010
http://msdn.microsoft.com/en-us/library/gg615464.aspx

Dicas:

- Criar uma Lista chamada "Logs" onde ira ficar registados todos os erros despoletados pelo Web Part ou outras funcionalidades associadas as Sandbox, as classes WriteTrace e SPDiagnosticsCategory não podem ser utilizadas em Sandbox devido a limitação do Scope ao Site do SharePoint Online.
- Validação do código através de Ferramentas próprias como MICROSOFT SHAREPOINT ONLINE CODE ANALYSIS FRAMEWORK (MSOCAF).
- Leitura documentação da Microsoft associada a Office 365

  http://www.microsoft.com/download/en/details.aspx?id=17069
  http://www.microsoft.com/download/en/details.aspx?id=14889
- Actualmente, não é possível implantar soluções directamente do Visual Studio 2010 para o SharePoint Online. Mas posso recomendar utilizar um Site de SharePoint 2010 "on-premise" com a Sandbox desenvolvida, esta também irá ser aceite no SharePoint Online.

Este é um pequeno exemplo sobre como criar Sandbox Solution utilizando os diversos Templates disponibilizados para o Visual Studio 2010.

Espero que tenham gostado, até a próxima…

Links utilizados como referência para este artigo:

Microsoft Office 365
http://www.microsoft.com/pt-pt/office365

Developing, Deploying, and Monitoring Sandboxed Solutions in SharePoint 2010
http://msdn.microsoft.com/en-us/magazine/ee335711.aspx

Developing for SharePoint Online with Sandbox Solutions
http://msdn.microsoft.com/en-us/hh181574

Sandboxed Solutions Architecture in SharePoint 2010
http://msdn.microsoft.com/en-us/library/ee539417.aspx

What Can Be Implemented in Sandboxed Solutions in SharePoint 2010
http://msdn.microsoft.com/en-us/library/gg615464.aspx

Restrictions on Sandboxed Solutions in SharePoint 2010
http://msdn.microsoft.com/en-us/library/gg615454.aspx

Add Actions to the User Interface
http://msdn.microsoft.com/en-us/library/ms473643(v=office.12).aspx

AUTOR

Escrito por André Lage
Microsoft SharePoint MVP, trabalhando na Suíça em empresa Farmacêutica e Banca. Apoiando as diversas Comunidades em SharePoint pelo mundo com participações em eventos como speaker e efectuando artigos com focos 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