PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº16 - Outubro de 2008 www.portugal-a-programar.org

Microsoft® Visual Studio® 2008 e Microsoft .net Framework 3.5

O que há de novo

ISSN 1647-0710

e ainda...

Concorrência em LINQ para SQL / Fundamentos de segurança em Redes / Grafos - 3ª parte

índice

equipa PROGRAMAR

coordenador
Miguel Pais

coordenador adjunto
Joel Ramos

editor
Pedro Abreu

redacção
João Brandão
Vitor Tomaz
Miguel Oliveira
Augusto Manzano
Ciro Cardoso
Pedro Diogo

colaboradores
David Ferreira
João Alves
João Matos
José Fontainhas
Pedro Teixeira

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

Pequenos Pormenores

32. É este o número de meses que este projecto já conta. São 16 edições, 72 redactores contabilizados, indefinido número de colaboradores anónimos que sabem bem a sua importância, 70 000 visitantes do nosso endereço web, 3771 downloads contabilizados para a última edição... Enfim, estes são alguns dos valores de que nos orgulhamos e não nos importamos de mostrar.

No entanto, este projecto está constantemente em desenvolvimento. Nada do que se obtém nos coloca num lugar em que deixemos de poder ainda mais atingir. O que há a fazer?

É essencialmente em tempo de férias que aproveitamos para observar, de uma perspectiva o mais longe possível e através dos olhos do utilizador, o que pode ser melhorado. Esta melhoria passa muitas vezes pela própria maneira como estamos organizados e nos damos a conhecer de modo a cativar novos participantes.

Nestas alturas apercebemo-nos que, por exemplo, em nenhum sítio do nosso endereço web era referido como podia qualquer utilizador participar neste projecto. Parece estranho, irónico, constrangedor, pensar que 32 meses passaram sem uma explicação presente e de fácil acesso para aqueles que, no fundo, são a essência do nosso projecto. Toda a informação que necessita poderá agora ser encontrada na secção Participar do nosso site.

E para esses mesmos utilizadores que decidem arriscar, que decidem partilhar um pouco do seu conhecimento com os outros através da escrita de um artigo. O que lhes temos para oferecer?

Numa das primeiras discussões que se seguiram a esta nossa tentativa de corrigir o que de mal estava, foi feito um tópico que reflectia sobre a relevância de escrever um artigo para a Revista Programar. "Vale a pena?" era a pergunta feita. Responder a tal não envolve apenas números e estatísticas, envolve a estrutura base do projecto e o que este pode oferecer aqueles que o suportam em termos de conteúdos técnicos. Não o fazendo por dinheiro, existe ou é nosso objecto que exista, um motivo muito mais relevante e nobre que leve alguém a contribuir com um pouco do seu tempo, paciência e boa vontade. Convencemo-nos que a Revista Programar tem a responsabilidade de se assumir como publicação de referência para os países de expressão portuguesa na área da programação, com uma qualidade que redefiniu os padrões do amadorismo, ou do que pode ser atingível desse modo. Como tal, decidimos registar a revista no centro internacional ISSN. Com isto pretendemos que cada vez mais os nossos editores, sendo eles estudantes, profissionais, entusiastas, sintam o orgulho na sua acção ao ponto desta merecer presença em currículo e nele se denotar como representante de importante competência, de um trabalho de qualidade avaliado por uma inteira comunidade.

Ainda assim, e virando-me agora para o outro utilizador, o que nos lê, não conseguimos deixar de nos sentir insatisfeitos sempre que o nosso formato e paradigma, que tentamos que potencie o conhecimento para todos, se torna uma barreira para a leitura e aprendizagem. O formato actual da revista é bastante consistente e portável, mas falha na possibilidade da fácil edição e, principalmente, na transferência dos trechos de código presentes numa folha enquanto imagem ineditável para um ambiente onde cada um possa facilmente correr, alterar e executar. É por isso que decidimos, começando nesta edição, disponibilizar imediatamente após o lançamento do .pdf, todos os artigos ao nosso alcance(1) da presente edição na Wiki do Portugal-a-Programar. Deste modo, referenciar um artigo será mais fácil, os próprios erros passíveis de correcção pelo leitor, havendo toda a liberdade necessária num projecto que tem este valor como referência.

Existem muitos mais pontos que foram analisados, e possivelmente mais novidades serão anunciadas. No entanto não hesitem nem deixem para depois a possibilidade de sugerir, sempre melhorar, um projecto que vive de pessoas que não hesitam em compartilhar um pouco do seu conhecimento. Partilha esta que, infelizmente, muitos se esquecem que outrora ocorreu quando chega a altura de serem eles os agentes fonte, com o dever de retribuir o que obtiveram noutros.

(1) Nem todos os artigos de uma edição são escritos na wiki, esses não estarão disponíveis.

Microsoft alarga investimento em I&D na Europa

A Microsoft continua a impulsionar os seus esforços nas buscas online e depois de anunciar um acordo para o desenvolvimento de um centro de inovação orientado para as buscas empresariais, na Noruega, informa agora que irá estabelecer outros três novos centros de investigação na Europa.

Na sua visita ao velho continente, Steve Ballmer continua a anunciar planos adicionais para aumentar os investimentos em solo europeu e revelou que será aberto um novo Centro Europeu de Tecnologia e Investigação (Search Technology Centre), dedicado à melhoria das tecnologias de busca. A estrutura terá três centros de excelência, em Paris, Munique e Londres, e será um ponto-chave para os serviços de investigação e desenvolvimento de tecnologias em solo europeu.

O objectivo da empresa passa por aproveitar o know-how local e impulsionar a inovação com oportunidades de emprego que ajudarão a reinventar a experiência online e de investigação do consumidor europeu. Ao aumentar os investimentos existentes, a Microsoft pretende impulsionar o crescimento da inovação e da economia do conhecimento europeia, explica a empresa em comunicado.

O STC vai-se unir ao alinhamento de mais de 40 centros de investigação e desenvolvimento da Microsoft, incluindo o laboratório de investigação internacional da empresa em Cambridge, os centros de desenvolvimento de Dublin, Copenhaga e Oslo, assim como aos dois mil investigadores e engenheiros que a empresa já comanda na Europa.

O Centro de Tecnologia e Investigação será dirigido por Jordi Ribas, que ocupará o cargo de director-geral do grupo empresarial Microsoft Connected TV, do qual já foi responsável.

O anúncio da Microsoft segue outros investimentos regionais já efectuados pela companhia em solo europeu, entre os quais, a compra da Multimap, empresa britânica especializada em mapas, da Ciao (alemã) e da Fast, companhia norueguesa de tecnologias de buscas empresariais.

SAPO comemora treze anos de existência

A entrar na adolescência, o SAPO comemora este mês treze anos de existência. Com 900 mil visitas por dia, o portal recebe mais de 3,6 milhões de visitantes por mês a navegar entre notícias, canais temáticos, vídeos e um conjunto de outras áreas de conteúdos que ao longo dos anos têm engordado na diversidade.

Também ao nível da imagem várias alterações têm acompanhado a vida do SAPO. A mais recente foi a renovação da homepage, "uma das mais profundas que fizémos desde sempre", como classifica Celso Martinho, CTO do portal.

"A mudança é sempre difícil num portal tão genérico como o SAPO, é difícil agradar a todos. Por isso mesmo o desenho da homepage actual foi acompanhado de processos rigorosos de recolha de especificações, estudos de usabilidade, focus groups, inquéritos e versões "beta" antes de ser colocada em produção", acrescenta.

As alterações terão agradado aos utilizadores e tido reflexos nas audiências do portal.

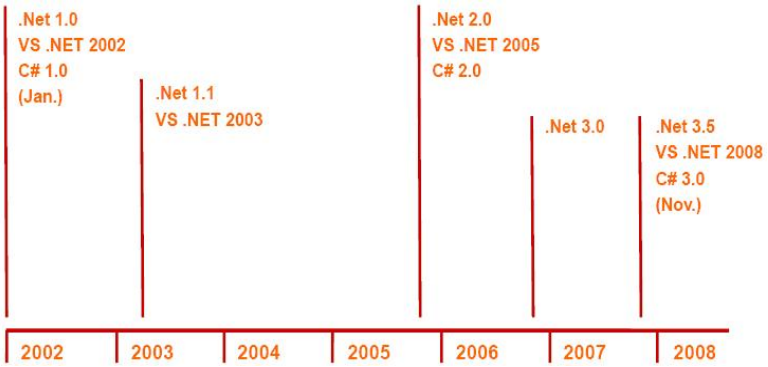
Mozilla quer um Firefox mobile

Lançar uma versão do Firefox para telemóveis é um dos objectivos da Mozilla para os próximos dois anos, avançou a presidente da fundação, Mitchell Baker no seu blog. A ideia não é nova mas ainda não tinha sido posicionada pela empresa no tempo.

Segundo a responsável, a organização quer desenvolver novos produtos e melhorar os já existentes, estendendo a presença do browser a outros mercados, de forma a aumentar a fatia de utilizadores com acesso móvel à Internet.

Mitchell Baker compromete-se ainda a expandir o sucesso do Firefox em acções que aumentem a sua quota de mercado mundial com a concepção de uma nova tecnologia de pesquisa como a do projecto Aurora, que propõe um browser HTML que incorpora comunicações e outras funcionalidades. A navegação e pesquisa são efectuadas num espaço a três dimensões, com miniaturas das páginas organizadas em nuvens relacionais, determinadas por relevância e relação.

Visual Studio 2008 e .NET Framework 3.5 (Parte 1)



Introdução

Esta é a primeira parte de um artigo que pretende fazer com que o leitor compreenda as novidades introduzidas na .NET Framework 3.5 bem como algumas das novidades do Visual Studio 2008.

Contudo, dada a extensão do tema, foi necessário dividi-lo em duas partes. Ambas as partes serão bastante práticas sendo apresentados exemplos de forma a que o leitor os possa usar como uma base de evolução.

Todos os exemplos deste artigo são para a linguagem C#. Contudo as mesmas novidades existem para VB.NET e Visual C++.

Fica aqui um pequeno aperitivo da segunda parte: Windows Presentation Foundation, Windows Workflow Foundation e ASP.NET 3.5.

Evolução da .NET Framework

A .NET Framework surgiu nos inícios de 2001 aquando do lançamento do Visual Studio 2002. Na .NET 1.0 vinha as primeiras versões do C#, VB.NET.

Em meados de 2003 a Microsoft lançou o Visual Studio 2003 e com ele veio uma nova versão da .NET Framework a .NET 1.1. Essa versão da .NET Framework veio corrigir alguns problemas da versão anterior bem como introduzir algumas novidades ao nivel das linguagens de programação.

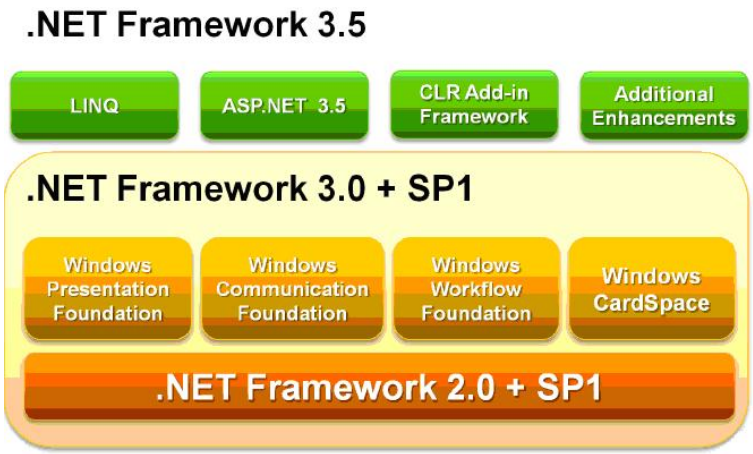
Em 2005 é lançado Visual Studio 2005 e com ele veio a .NET 2.0 e o C# 2.0. Esta versão da framework trouxe muitas novidades e melhorias em relação à anterior e a nível do C# as novidades foram imensas o que contribuíram para que esta linguagem se tornasse ainda mais atractiva.

Em finais de 2006 e 2007 saíram as duas ultimas versões da .NET Framework a 3.0 e a 3.5. Com a .NET 3.5 saiu a versão 3.0 do C# e a versão 9.0 do VB.NET.

O que é a .NET Framework 3.5

A .NET Framework tem sido adoptada pela comunidade de desenvolvimento desde o seu lançamento, em 2002. De forma a fornecer um melhor desempenho, flexibilidade e redução no trabalho de codificação, a Microsoft lança agora o .NET Framework 3.5.

A .NET Framework 3.5 tem dois elementos que são necessários entender: Os Bits vermelhos e os Bits verdes.Quando falamos em Bits vermelhos falamos em todas as funcionalidades existentes nas .NET Framework 2.0 e 3.0 bem como alguns updates a essas duas versões da .NET Framework.



Mas esses updates não são a totalidade das novas funcionalidades mas sim mais um service pack com correcções de bugs e melhorias.

Os Bits verdes são o conjunto das novas funcionalidades e add-ons que foram introduzidos na .NET Framework.

Algumas dessas novas funcionalidades são:

•LINQ – Facilita enormemente a consulta e manipulação de informação a nível de classes, objectos e da base de dados;

•Visual Studio 2008 – Primeiro Visual Studio multi-targeting pois permite construir aplicações direccionadas a varia versões da .NET Framework, permite também o Debug de javascript com intellisense;

•Windows Presentation Foundation – Designer integrado com o Visual Studio, edição visual de XAML com intellisense;

•Varias ferramentas para dispositivos móveis – Unit testing para dispositivos móveis;

•Device emulator 3.0.

Novidades da .NET Framework 3.5

Em 2005 fomos atraídos para o lançamento da segunda grande versão da .NET Framework que introduziu novidades tais como os generics, delegates anónimos, etc. Quando em 2006 foi apresentada a especificação do C# 3.0 todos tivemos a oportunidade de ver qual o caminho que a linguagem estava a seguir. Ao contrário do C# 2.0 que introduziu novas funcionalidades para se manter actualizado com o que de novo estava a surgir nas outras linguagens, o C# 3.0 veio introduzir funcionalidades radicais muitas das quais nunca tinham sido vistas em qualquer linguagem de programação. Pode-se dizer que o futuro da .NET Framewok não é só um futuro de evolução mas sim de revolução.

É com este impulso que vamos ver algumas dessas novidades.

Auto-Implemented Properties

Tradicionalmente a construção das propriedades nas classes requeria a construção das componentes Get e Set. Na maioria dos casos, esses componentes implementavam pouca lógica funcional, acabando por gerar alguma redundância de código nas aplicações.

Implementação das propriedades no C# 2.0:

```
class Pessoa
{
    private string nome;
    private string apelido;

    public string Nome
    {
        get {return nome;}
        set {nome = value;}
    }
    public string Apelido
    {
        get {return apelido;}
        set {apelido = value;}
    }
}
```

No C# 3.0 podemos declarar as componentes Get e Set vazias. Quando isso acontece, o compilador gera automaticamente um campo privado para a classe e constrói uma implementação Get e Set públicas para a propriedade.

Implementação das propriedades no C# 3.0:

```
class Pessoa
{
    public String Nome
    {
        get;
        set;
    }

    public String Apelido
    {
        get;
        set;
    }
}
```

Object Initialization

O C# 3.0 facilita enormemente a inicialização dos valores das propriedades. Anteriormente as propriedades tinham que ser inicializadas no construtor ou explicitamente após instanciar o objecto.

Inicialização das propriedades no C# 2.0:

```
Pessoa antonio = new Pessoa();
antonio.Nome = "António";
antonio.Apelido = "Santos";
```

No C# 3.0 podemos usar a funcionalidades Object Initialization que permite indicar valores para as propriedades sem obrigar à utilização do construtor ou a atribuir valores de forma explícita.

Inicialização das propriedades no C# 3.0

```
Pessoa joao = new Pessoa
{
    Nome = "João",
    Apelido = "Sousa"
};
```

Extension Methods

Extension Methods são métodos adicionais que são definidos para complementar as funcionalidades dos tipos base. Permitem adicionar novos métodos aos tipos existentes na CLR sem ter que recorrer ao sub-classing ou à recompilação do tipo original. Desta forma podemos incrementar o âmbito funcional de tipos base de uma forma extremamente prática.

Criação de um Extension Method:

```
public static class ValidarEmail
{
    public static bool
emailValido(this string str)
    {
            Regex regex = new
Regex(@"[\w-\.]+@([\w-]+\.)+[\w-
]{2,4}$");
            return regex.isMatch(str);
    }
}
```

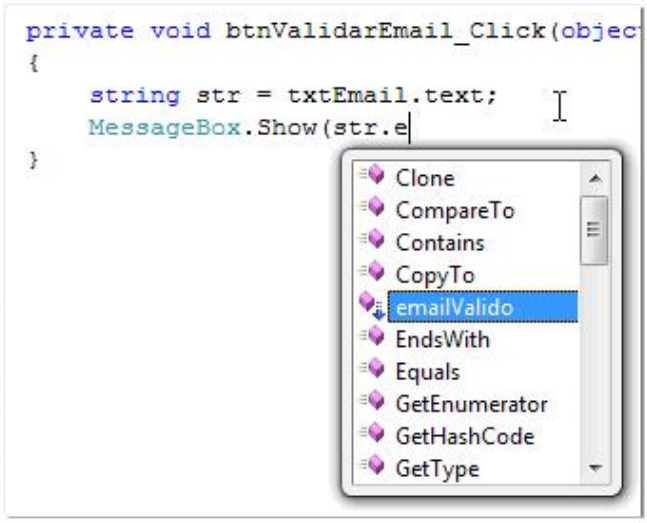
Para definir os Extension Methods é necessário ter em consideração algumas coisas:

•A classe que implementa os Extension Methods tem que ser estática para poder ser usada sem ser necessário instanciar;

•O Extension Method é um método estático e recebe como parâmetro o tipo base que vamos estender;

•O tipo a estender é antecedido pela keyword this para indicar ao compilador que o método deve ser adicionado aos métodos do tipo a estender.

Para usar a classe que contém os Extension Methods basta importá-la com o using. A partir desse momento o método passa a estar disponível para o tipo de dados que foi estendido.



Um exemplo interessante da aplicação dos Extension Methods é a adição de novos métodos aos objectos da .NET Framework. Por exemplo um dos métodos que podemos adicionar é um método In que permite validar se um objecto recebido como argumento é ou não parte do mesmo.

```
public static class ExtensaoIn
{
    public static bool In(this object
obj, IEnumerable<object> col)
    {
        foreach(object o in col)
        {
            if (o.Equals(obj)) return
true;
        }
        return false;
    }
}
```

Type Inference

No C# 3.0 a Microsoft introduziu uma nova keyword o var. Essa keyword permite declarar uma variável em que o seu tipo é implicitamente definido pela expressão usada para inicializar a variável.

Exemplo de uso do Type Inference:

```
var i = 10;
```

No exemplo acima o valor 10 é armazenado na variável i e esta é definida como inteira. A declaração de variáveis usando var permite utilizar as variáveis de forma Strongly

Typed em tempo de compilação. Como é suportada em tempo de compilação, suporta a utilização de intellisense como se tivéssemos indicado o tipo de dados de forma explícita.

Para garantir que a variável é convertida de forma correcta, é necessário que a atribuição seja feita na mesma linha que a declaração e a componente de atribuição tem que ser uma expressão. Deste modo não pode ser um objecto, nem uma collection, nem pode ser nula.

O Type Inference só pode ser usado em variáveis locais. Não pode ser usado nem propriedades ou campos e nem pode ser retornado por métodos.

Anonymous Types

Anonymous Types são classes sem nome que podem ser criadas dinamicamente no C# 3.0. Desta forma consegue-se criar novos tipos de dados em C#. Essas classes são criadas pelo compilador baseando-se na inicialização do objecto que está a ser instanciado.

Em vez de se declarar o tipo de dados de forma tradicional, define-se o tipo inline como parte do código onde está a ser utilizado.

Declaração tradicional:

```
Morada morada = new Morada
{
    Rua = "Alameda das Flores 130",
    Localidade = "Pico"
}
```

Declaração usando Anonymous Types:

```
var morada = new {
    Rua = "Alameda das Flores 130",
    Localidade = "Pico"
}
```

Uma vez declarado o Anonymous Type, podemos usar a variável declarada como um objecto perfeitamente normal.

Note-se que, ao ser usada a inferência de dados com o var, tem-se acesso, dentro do método onde foi declarado, a um ambiente strongly typed, pois o compilador conhece todos os detalhes do Anonymous Type. No entanto não se pode retornar o Anonymous Type para fora do âmbito do método onde foi declarado. Como não existe um nome público para o tipo este não pode ser referenciado do exterior. A única alternativa de retornar um Anonymous Type é retorná-lo como Object.

Lambda Expressions

Expressões lambda são expressões que fornecem uma sintaxe curta para especificar algoritmos como argumentos para métodos.

As expressões lambda já são usadas a bastante tempo em várias linguagens de programação entre as quais podemos destacar o Lisp.

O modelo conceptual das expressões lambda foi criado em 1936 pelo matemático Alonzo Church.

No C# uma expressão lambda é indicada como:

```
[Lista de parâmetros separados por
vírgulas] + [operador =>] + [Expressão
ou bloco de instruções]

(Param1, Param2, ..., ParamN) =>
expressão
```

Exemplo de criação de um filtro usando expressões lambda:

```
Func<string, bool> filter = x =>
x.Length > 4;
```

Novidades no IDE do Visual Studio 2008

Após dez anos de vida o Microsoft Visual Studio chega à sua versão 2008. Entre as inúmeras novidades estão:

•.NET Framework 3.5, com suporte a AJAX;

• Possibilidade de escrever programas não só para versão 3.5 como para as versões 3.0 e 2.0 da .NET Framework;

• LINQ - Language Integration Query, que permite ao programador aproveitar ao máximo o potencial das consultas;

• Debug ao Javascript com recurso intellisense;

• Melhor suporte ao design de HTML e CSS (Split view, pré visualização de master pages encadeadas, gestão de CSS, etc.);

• Melhorias na contrução da interface para aplicações web;

• Possibilidade de integração com o SilverLight;

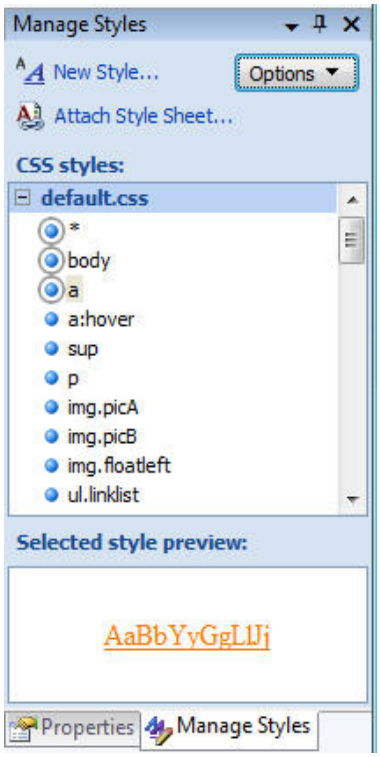
• Designer de WPF integrado (Cider);

• Novos controlos de dados (LinqDataSource, ListView, DataPager);

• Edição de XAML com intellisense.

Tal como foi dito anteriormente, uma das melhorias do Visual Studio 2008 é o suporte ao design de HTML e CSS. O Visual Studio 2008 utiliza o mesmo web designer que a Microsoft fornece com o Expression Web.

Seguidamente vai-se abordar algumas das melhorias que podem ser encontradas no IDE do Visual Studio 2008.

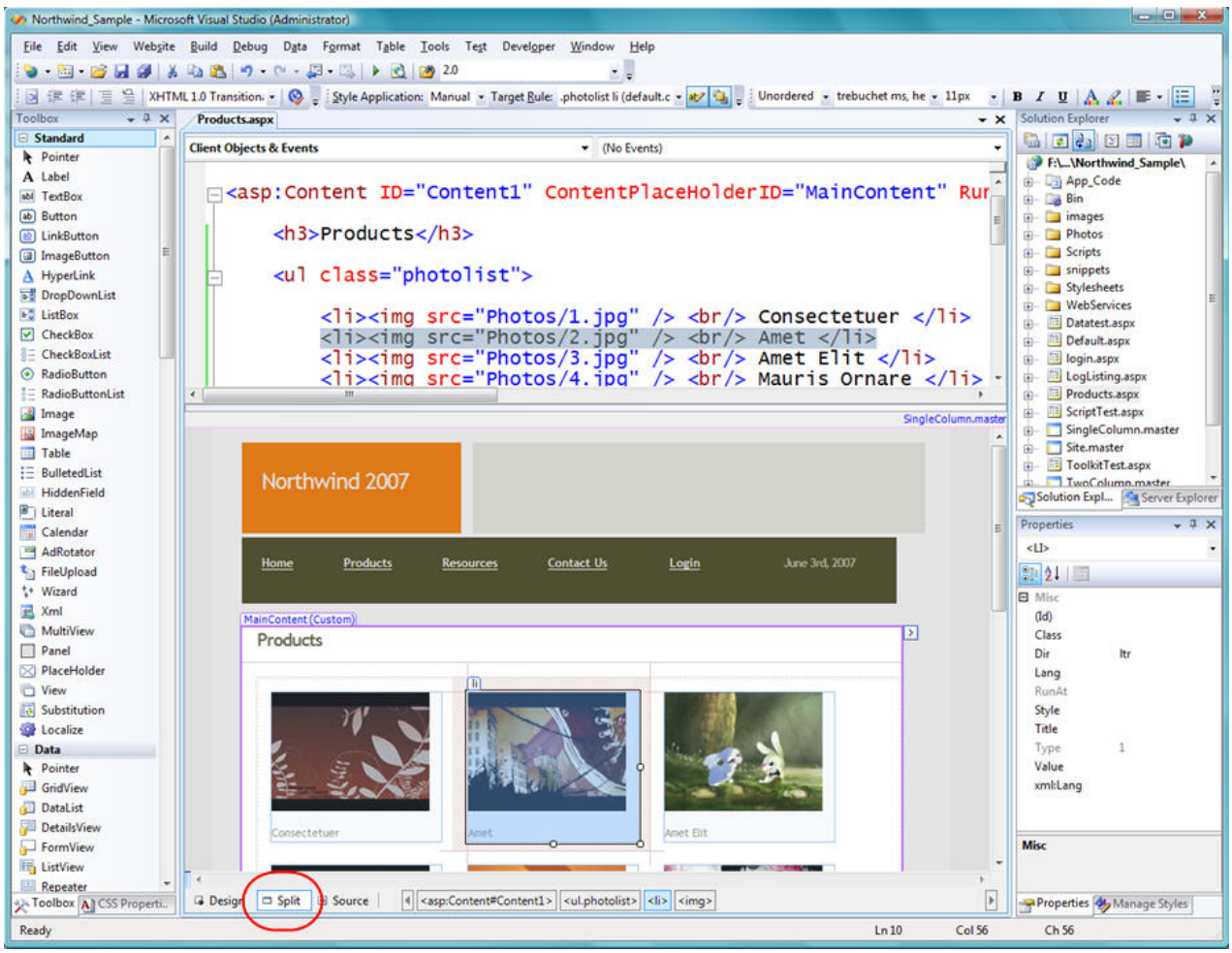


Split View

Para facilitar a visualização em simultâneo do codigo fonte e do design, a Microsoft melhorou o split view:Como se pode ver a nova split view permite ver o código fonte e o design em simultâneo e vermos as alterações realizadas quer sejam feitas no design ou no código fonte.

CSS Style Manager

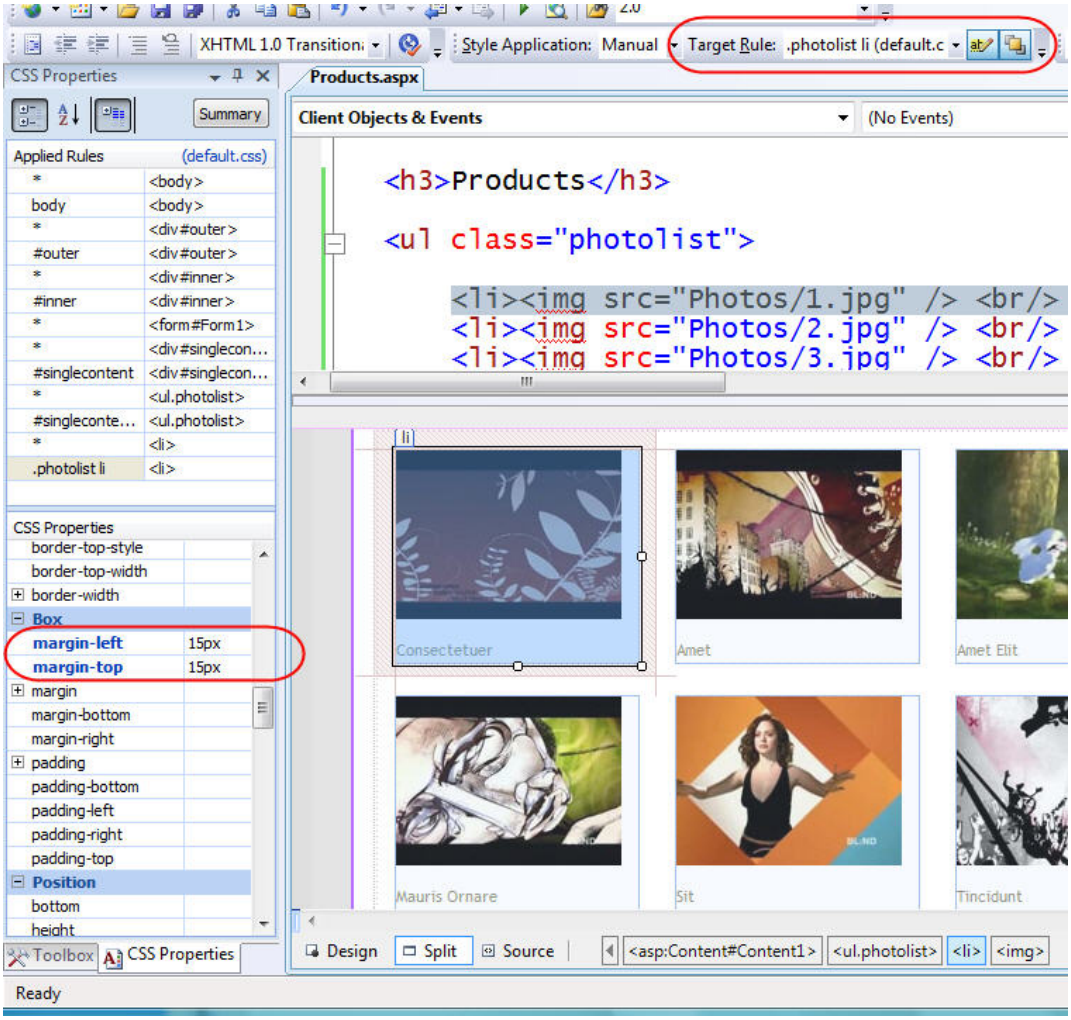
Uma das novidades existentes no Visual Studio 2008 é o "Manage Styles". Esta ferramenta mostra todas as stylesheets de CSS, bem como as regras que as compõem. Pode ser usada tanto na vista de design ou na vista de código fonte.



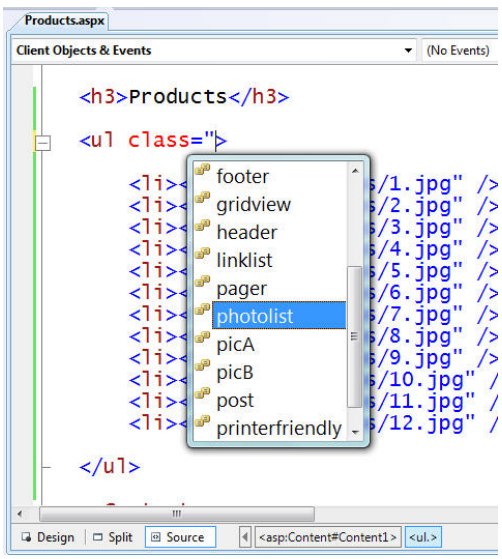
Split View

CSS Properties Window

Outra das novidades que pode ser usada tanto no modo de visão do design e de codigo fonte é a nova janela de propriedades das CSS.



Quando seleccionamos um objecto ASP.NET ou um elemento HTML de uma página, a janela de propriedades das CSS mostra todas as definições aplicadas aquele objecto. Podemos alterar também qualquer definição.



CSS Source View Intellisense

Ao seleccionar um elemento no designer de HTML pode-se usar o intellisense para aplicar regras CSS ao elemento seleccionado.

O intellisense das CSS funciona tanto para páginas HTML como para páginas ASP.NET. Está disponível também para master pages e master pages encadeadas.

Master Pages Encadeadas

O aparecimento das master pages aquando do lançamento da .NET Framework 2.0 foi uma das novidades mais populares dessa versão.

Mas como o leitor se deve recordar o Visual Studio 2005 não permitia a edição de master pages encadeadas. Essa funcionalidade foi corrigida no Visual Studio 2008.

LINQ – Language Integrated Query

O LINQ é um conjunto de recursos introduzidos no .NET Framework 3.5 que permitem a realização de consultas directamente em base de dados, documentos XML, estruturas de dados, colecções de objectos, etc. usando uma sintaxe parecida com a linguagem SQL.

O Visual Studio 2008 integra assemblies que implementam o provider do LINQ permitindo assim a sua utilização com diversas origens de dados.

O LINQ é aplicável a qualquer colecção que implemente o interface IEnumerable<T>, desta forma qualquer Collection e Array pode ser consultada usando LINQ.

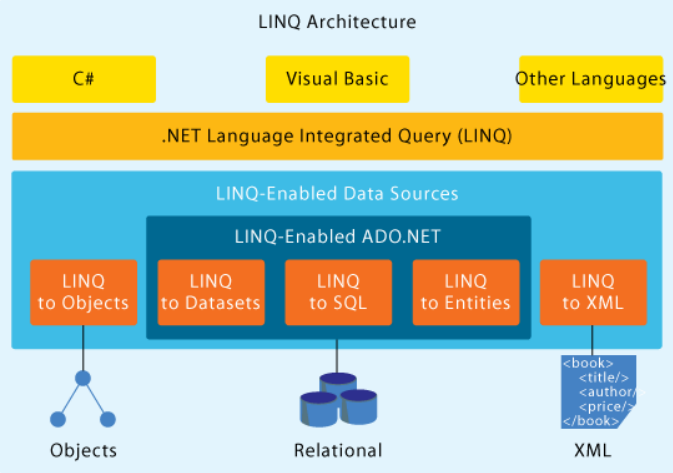
Qualquer classe que implementar o interface IEnumerable<T> pode ser consultada usando LINQ. Na figura pode-se ver a arquitectura do LINQ.

Existem cinco formas nas quais poderemos utilizar o LINQ.

- LINQ to SQL
- LINQ to XML
- LINQ to Objects
- LINQ to DataSet
- LINQ to Entities

Conclusão

Como o leitor pode ver as potencialidades da .NET Framework 3.5 são muitas. Mas tal como alguém disse um dia "Só porque te dei um martelo novo não significa que tudo agora é um prego". Todas estas novidades devem ser utilizadas com moderação.



Na 2ª parte deste artigo falar-se-á com mais algum pormenor do LINQ e também de: Windows Presentation Foundation, Windows Workflow Foundation e ASP.NET 3.5.

Bibliografia

- Visual C# Developer Center
  http://msdn.microsoft.com/pt-br/vcsharp/default(en-us).aspx

- C# Language Specification
  http://download.microsoft.com/download/3/8/8/388e7205-bc10-4226-b2a8-75351c669b09/csharp%20language%20specification.doc

- VB.NET Language Specification
  http://www.microsoft.com/downloads/details.aspx?FamilyID=39DE1DD0-F775-40BF-A191-09F5A95EF500&displaylang=en

- Microsoft .NET Framework 3.5
  http://www.microsoft.com/downloads/details.aspx?familyid=333325FD-AE52-4E35-B531-508D977D32A6&displaylang=en

SOBRE O AUTOR

Residente no Porto, João Brandão tem 9 anos de experiência em desenvolvimento de software e em montagem e elaboração de redes de computadores. Ocupa desde Dezembro de 2005 o cargo de Senior Engineer no Software Development Center da Qimonda AG. Tem como principais actividades e responsabilidades o desenvolvimento software e gestão de projectos numa grande variedade de aplicações na área web e não web.

João Brandão

Concorrência em LINQ para SQL

Introdução

A correcta gestão de acessos concorrentes e transacções são tarefas muito importantes numa aplicação. A arquitectura da DLINQ (LINQ para SQL) já nos fornece uma implementação base preocupada com estes aspectos, mas também permite que o programador personalize essa implementação de modo a adaptá-la às necessidades da sua aplicação.

Não sendo o objectivo deste artigo explicar a base do funcionamento da DLINQ é necessário explicar alguns conceitos mais básicos do seu funcionamento de modo a enquadrar os leitores com menos contacto com esta linguagem/tecnologia. De modo a cumprir esse objectivo iremos falar, ligeiramente mais à frente, sobre a classe DataContext.

Nesta tecnologia, a técnica utilizada para detectar e resolver conflitos de concorrência tem o nome de concorrência optimista. O nome resulta da crença em que a probabilidade das alterações produzidas por uma transacção interferir com outra transacção é baixa. A DLINQ utiliza um esquema de dados desconectados, onde cada utilizador pode fazer as alterações que quiser na sua cópia dos dados (instância de DataContext). No momento em que o programa tentar propagar essas alterações para a base de dados é realizada a detecção de conflitos. A detecção de conflitos consiste basicamente em verificar se os dados presentes na base de dados foram modificados desde que a aplicação cliente fez o seu último acesso. Se for detectado algum conflito, o programa necessita de saber como resolvê-lo, nomeadamente se escreve por cima os novos dados, se descarta os novos dados ou se faz alguma operação entre eles. A detecção de conflitos é uma das funcionalidades da classe DataContext. Quando o programador tenta actualizar a base de dados chamando o método SubmitChanges da classe DataContext é automaticamente realizada uma detecção de conflitos.

Sempre que existe algum conflito é lançada a excepção ChangeConflictException, portanto todas as chamadas ao método SubmitChanges devem estar protegidas por um bloco try/catch.

Além da informação sobre a origem e a mensagem, a classe ChangeConflictException oferece ainda a possibilidade de resolver os conflitos de concorrência através de utilização dos enumerados RefreshMode e ConflictMode como veremos mais à frente.

De modo a tornar o código mais breve e legível muitos dos exemplos de DLINQ disponíveis na Internet ou em livros não definem um modo de detectar e resolver conflitos de concorrência, tenha em atenção que em código para produção deverá sempre defini-los.

A classe DataContext

Na base da DLINQ está a classe DataContext. É esta classe que fornece a maioria dos serviços/funcionalidades e sobre a qual são realizadas a maioria das operações. Como principais funcionalidades disponibilizadas por esta classe temos:

- Gestão da ligação à base de dados.
- Mapeamento entre os objectos e a base de dados.
- Tradução e execução das queries.
- Gestão da identidade dos objectos.
- Gestão de alterações nos objectos.
- Processador de alterações.
- Gestão da integridade transaccional.

Para uma melhor compreensão deste artigo importa saber que as tabelas da base de dados são mapeadas em tipos (classes) do lado da linguagem de programação e que as colunas das tabelas são suportadas através de propriedades. Sempre que forem referidas propriedades será neste contexto, representação de colunas de uma tabela.

Acessos ReadOnly

Vamos começar por falar de uma situação que, apesar de não gerar conflitos de concorrência, pode ocorrer várias vezes na vida de uma aplicação. Muitas vezes pretende-se aceder a uma base de dados apenas para consultar os seus dados sem que se pretenda alterá-los num futuro próximo. Um exemplo disso podem ser os valores a apresentar numa ComboBox, como por exemplo uma lista de países do mundo. A classe DataContext controla as modificações feitas aos valores das propriedades que representam os campos das tabelas da base de dados mantendo em

memória o valor original e o valor actual. A manutenção desses dois valores e o processo de detectar alterações tem um peso computacional. A classe DataContext controla ainda a identidade dos objectos, cada vez que é realizada uma consulta á base de dados, o serviço de gestão de identidade da classe DataContext verifica se um objecto com a mesma identidade já foi devolvido numa consulta anterior. Se assim for, a DataContext irá retornar o valor armazenado na cache interna em vez de o obter da base de dados. Esta gestão de identidade também tem um peso computacional. No caso de pretender apenas dados para leitura pode suprimir estes pesos computacionais e obter daí um aumento de performance na aplicação. Para obter este efeito deverá ser afectada a propriedade ObjectTrackingEnabled da DataContext com falso. Esta afectação indica à framework que não precisa de detectar alterações nos dados nem fazer gestão de identidade.

```
DatabaseDataContext dataContext = new
DatabaseDataContext();
dataContext.ObjectTrackingEnabled =
false;
```

O leitor repare que se fizer uma chamada ao método SubmitChanges será gerada uma excepção visto que não existem alterações a submeter. Será ainda lançada uma excepção caso a propriedade seja afectada com falso após a execução de uma query.

Nota: Ao afectar ObjectTrackingEnabled com falso o valor da propriedade DeferredLoadingEnabled será ignorado e inferido como falso.

Detecção de Conflitos

Existem dois métodos para realizar a detecção de conflitos de concorrência. Se existir alguma propriedade marcada com o atributo IsVersion a verdadeiro, a detecção de conflitos é realizada apenas com base na chave primária da tabela e no valor dessa coluna. Se não existir nenhuma propriedade com o atributo IsVersion como verdadeiro, a DLINQ permite ao programador definir quais as colunas que participam na detecção de conflitos através do atributo UpdateCheck das propriedades.

Nota: O atributo IsVersion especifica se uma propriedade que representa uma coluna da base de dados contém um número de versão ou um timestamp do registo. As definições de UpdateCheck da classe serão ignoradas na presença de uma propriedade marcada com IsVersion a verdadeiro.

De modo a percebermos melhor a forma como os conflitos são detectados vamos tentar perceber como está implementado. Quando é realizada a chamada ao método SubmitChanges é gerado código SQL para realizar a persistência dos dados na base de dados. Quando for necessário realizar uma actualização dos dados na base de dados não é enviado apenas a chave primária da tabela na cláusula where mas também todas as colunas que irão participar na detecção de conflitos. A detecção de conflitos é realizada enviando na cláusula where os valores originais das colunas, detectando-se assim eventuais mudanças. Nada melhor que um exemplo para ficarmos realmente a perceber o comportamento. Vamos imaginar que temos uma tabela Cliente com dois campos: Nome (chave primária) e Cidade.

Table - dbo.Cliente

Nome	Cidade
Vitor	Lisboa

Como operação iremos actualizar o valor de Cidade do cliente Vítor para Torres Vedras. Como podemos observar, a detecção de conflitos é realizada enviando na cláusula where os valores originais:

```
UPDATE [dbo].[Cliente]
SET [Cidade] = @p2
WHERE ([Nome] = @p0) AND ([Cidade] =
@p1)
-- @p0: Input NChar (Size = 10; Prec =
0; Scale = 0) [Vítor]
-- @p1: Input NChar (Size = 10; Prec =
0; Scale = 0) [Lisboa]
-- @p2: Input NChar (Size = 10; Prec =
0; Scale = 0) [Torres Vedras]
```

Para exemplificar a detecção de conflitos usando uma coluna de versão/timestamp foi adicionado um campo com o nome Version à tabela e definido o atributo IsVersion como verdadeiro na propriedade da DataContext que o representa.

Table - dbo.Cliente

Nome	Cidade	Version
Vitor	Lisboa	1

Como podemos observar no exemplo seguinte a detecção é agora realizada utilizando apenas na cláusula where a chave primária e a propriedade marcada como IsVersion.

```
UPDATE [dbo].[Cliente]
SET [Cidade] = @p2
WHERE ([Nome] = @p0) AND ([Version] =
@p1)

SELECT [t1].[Version]
FROM [dbo].[Cliente] AS [t1]
WHERE ((@@ROWCOUNT) > 0) AND
([t1].[Nome] = @p3)
-- @p0: Input NChar (Size = 10; Prec =
0; Scale = 0) [Vítor]
-- @p1: Input Int (Size = 0; Prec = 0;
Scale = 0) [1]
-- @p2: Input NChar (Size = 10; Prec =
0; Scale = 0) [Torres Vedras]
-- @p3: Input NChar (Size = 10; Prec =
0; Scale = 0) [Vítor]
```

O atributo UpdateCheck

Como já vimos, podemos realizar a detecção de conflitos através do atributo UpdateCheck. Exemplo:

```
[Column(DbType =
"nvarchar(50)",UpdateCheck =
UpdateCheck.WhenChanged)]
public string Nome;
```

Apesar de, na maioria das situações, a melhor opção ser informar o utilizador que existe um conflito de concorrência e disponibilizar mecanismos para a resolver, podem existir cenários em que a concorrência não é uma preocupação. Neste caso podemos simplesmente ignorar qualquer alteração concorrente e efectuar sempre a actualização dos registos, ficando gravado na base de dados a ultima actualização submetida. Esta opção pode implementada atribuindo UpdateCheck.Never em todas as propriedades. Existem casos em que estarem dois utilizadores a alterar colunas diferentes da mesma linha não levanta problemas. Para implementar esta situação deverá definir o atributo UpdateCheck como WhenChanged.

Por omissão as propriedades são consideradas com estando marcadas como UpdateCheck.Always, o que significa que irão sempre participar na detecção de conflitos, independentemente de terem sofrido alterações após a última leitura da base de dados. Ter todas as colunas a participar nessa detecção pode ser bastante penalizador para o desempenho da aplicação. O programador deverá rever este atributo em todas as propriedades de modo a apenas deixar marcadas as propriedades vitais para o correcto funcionamento da aplicação. Lembra-se de termos falado há pouco que as detecções eram feitas enviando os valores originais na cláusula where? Ora bem, se a propriedade tiver o atributo UpdateCheck definido como Always, o valor original da mesma irá sempre fazer parte da cláusula where. Se o valor de UpdateCheck estiver definido como WhenChanged essa propriedade irá fazer parte da cláusula where apenas se o valor corrente for diferente do original, ou seja, tiver sido modificado. No caso do atributo UpdateCheck estar definido como Never, essa coluna não estará presente na cláusula where, ou seja, não fará parte da detecção de conflitos .

O parâmetro ConflictMode

O método SubmitChanges aceita como parâmetro uma opção do enumerado ConflictMode, que tem dois valores possíveis: ContinueOnConflict e FailOnFirstConflict. A opção ContinueOnConflict, define que todas as actualizações serão tentadas, os conflitos que ocorrerem serão reunidos numa colecção e retornados no fim do processo. A opção FailOnFirstConflict, que é a opção por omissão, define que as actualizações deverão parar imediatamente após o primeiro conflito.

```
try
{

dataContext.SubmitChanges(ConflictMode.
ContinueOnConflict);
}
catch (ChangeConflictException){   (…)
}
```

O leitor tenha em atenção que ao utilizar o modo FailOnFirstConflict seria de esperar que todas as alterações á base de dados realizadas antes do primeiro conflito acorrer tivessem sucesso. Este comportamento poderia deixar a base de dados inconsistente dado que apenas parte dos dados teriam sido actualizados. Falaremos em transacções mais à frente neste artigo mas é importante perceber nesta fase que, por omissão, o DataContext cria uma transacção sempre que o SubmitChanges é chamado. Se for lançada uma excepção ocorre um rollback automático da transacção.

Resolução de Conflitos

Os objectos em memória guardam o valor original (o valor obtido da base de dados) e o valor actual (novo valor atribuído pelo utilizador). Como já terá percebido, um conflito ocorre quando o valor original do objecto não

corresponde ao valor presente na base de dados, ou seja, na cláusula where vai um parâmetro onde o seu valor não corresponde ao valor presente na base de dados. O processo de resolução de conflitos resume-se basicamente em actualizar os valores dos objectos em memória de modo a deixarem de existir estes conflitos. Os métodos que permitem realizar estas actualizações recebem como parâmetro um RefreshMode, que permite ao programador definir a forma como estes objectos serão actualizados.

Antes de falarmos sobre as formas de resolver os conflitos iremos tentar perceber o que é o enumerado RefreshMode e quais as opções que nos oferece. Iremos falar ainda sobre a propriedade ChangeConflicts, que nos permite obter mais informações sobre os objectos que estão em conflito.

O enumerado RefreshMode

O enumerado RefreshMode permite ao programador definir como os objectos serão actualizados de modo aos conflitos serem resolvidos. Este enumerado disponibiliza 3 opções, KeepChanges, KeepCurrentValues ou OverwriteCurrent Values. Ao utilizarmos RefreshMode.KeepChanges estamos a definir que queremos obter da base de dados para as propriedades da classe todos os dados que foram alterados, excepto se essas propriedades tiverem sido previamente alteradas pelo utilizador. Obteremos da base de dados os dados alterados pelo outro utilizador que não foram alterados por nós. (Serão persistidos por esta ordem de prioridade: os dados alterados pelo utilizador, as alterações obtidas da base de dados ou os valores originalmente obtidos.)

A opção RefreshMode.KeepCurrentValues define que as alterações feitas na base de dados por outros utilizadores serão descartadas e que os dados a serem persistidos são os que estão presentes nos nossos objectos da aplicação. (Serão persistidos por esta ordem de prioridade: os dados alterados pelo utilizador, os valores originalmente obtidos.)

Por fim a utilização de RefreshMode.OverwriteCurrent Values define que os valores a persistir são os valores correntes que estão presentes na base de dados e não os valores que nós temos nos nossos objectos. (Serão persistidos por esta ordem de prioridade: as alterações recebidos da base de dados, os valores originalmente obtidos.)

A Colecção ChangeConflicts

A classe DataContext dispõe de um propriedade que nos permite obter os objectos em conflitos. Essa propriedade tem o nome de ChangeConflicts e retorna uma colecção de objectos do tipo ObjectChangeConflict e representa todas as tabelas onde existem conflitos. Cada instância de ObjectChangeConflict dispões de uma colecção que representa todas as colunas dessa tabela onde existem conflitos. Essa propriedade tem o nome de MemberConflicts e armazena objectos do tipo MemberChangeConflict. Sobre objectos deste tipo podemos consultar informações tais como o valor original, o valor corrente e o valor presente na base de dados.

```
try
{
db.SubmitChanges(ConflictMode.ContinueO
nConflict);
}
catch (ChangeConflictException)
{
  foreach (ObjectChangeConflict tabela
in db.ChangeConflicts)
  {
    foreach (MemberChangeConflict
coluna in tabela.MemberConflicts)
    {
      object ValorOriginal =
coluna.OriginalValue;
      object ValorActual =
coluna.CurrentValue;
      object ValorNaBaseDados =
coluna.DatabaseValue;
    }
  }
}
```

Resolver os Conflitos

Agora que já temos mais conhecimentos sobre o enumerado RefreshMode e sobre a colecção ChangeConflicts iremos falar sobre três formas de resolver os conflitos disponíveis na DLINQ. A primeira e a mais simples de todas é usar o método ResolveAll sobre a colecção ChangeConflicts. Desta forma podemos resolver todos usando um RefreshMode como parâmetro. Todos os objectos em conflito serão actualizados usando o mesmo RefreshMode.

```
try
{
db.SubmitChanges(ConflictMode.ContinueO
nConflict);
}
catch (ChangeConflictException)
{

db.ChangeConflicts.ResolveAll(RefreshMo
de.KeepChanges);
}
```

Se desejarmos um comportamento diferente para cada tabela podemos usar o método Resolve sobre cada ObjectChangeConflict. Como já foi referido, a classe DataContext guarda numa colecção os objectos em conflito. Podemos percorrer essa colecção através da propriedade ChangeConflicts e definir o RefreshMode para cada um desses objectos. Cada ObjectChangeConflict representa uma tabela da base de dados em conflito.

```
try
{
db.SubmitChanges(ConflictMode.ContinueO
nConflict);
}
catch (ChangeConflictException)
{
  foreach (ObjectChangeConflict tabela
in db.ChangeConflicts)
  {
tabela.Resolve(RefreshMode.KeepChanges)
;
  }
}
```

Se desejarmos refinar ainda mais o método de resolução podemos usar o método Resolve sobre cada MemberChangeConflict e definir o RefreshMode para cada coluna da tabela. Cada objecto da classe ObjectChangeConflict dispõe de uma propriedade com o nome de MemberChangeConflicts que nos dá acesso a uma colecção de objectos em conflito. Cada objecto desta colecção representa uma coluna de uma tabela da base de dados em conflito.

```
try
{
db.SubmitChanges(ConflictMode.ContinueO
nConflict);
}
catch (ChangeConflictException)
{
  foreach (ObjectChangeConflict tabela
in db.ChangeConflicts)
  {
    foreach (MemberChangeConflict
coluna in tabela.MemberConflicts)
    {
coluna.Resolve(RefreshMode.KeepChanges)
;
      //
coluna.Resolve(coluna.OriginalValue);
    }
  }
}
```

Como o leitor se apercebeu, a forma de resolver os conflitos é bastante simples, podendo o programador decidir até que nível de profundidade pretende ir.

Transacções

O controlo de transacções serve para garantir a integridade dos dados, isso significa que através dele podemos ter a certeza de que todos os dados serão gravados na base de dados. Em caso de eventual falha nalgum momento da gravação todo o processo volta ao estado inicial, sem inserções parciais. Ao ocorrer um erro durante o processo é realizado um rollback e nada é gravado na base de dados. Ao completar o processo com sucesso, a transacção gera um commit e grava os dados na base de dados. Como podemos observar, nos exemplos até agora apresentados, nunca ouve a preocupação de implementar um modelo transaccional de modo a que, na ocorrência de conflitos, os registos gravados antes de o conflito ocorrer voltem ao estado inicial. Esta situação poderia deixar a base de dados inconsistente já que alguns registos seriam gravados e outros não. De modo a resolver este problema, por omissão, a classe DataContext cria uma transacção quando o método SubmitChanges é chamado. Caso ocorra algum conflito é feito rollback automaticamente sobre a transacção. No modo de transacção implícita é verificado se a chamada ao método SubmitChanges é realizada no contexto de uma transacção ou se a propriedade Transaction da classe DataContext está afectada com uma transacção local iniciada pelo utilizador. Irá utilizar a primeira que encontrar. Caso não encontre nenhuma é iniciada uma nova transacção para executar os comandos SQL gerados. Se todos os comandos correrem com sucesso a DLINQ faz commit à transacção e retorna. Caso exista algum conflito faz rollback.

Como o leitor já percebeu a classe DataContext tem uma propriedade com o nome Transaction. Esta propriedade permite ao programador definir qual a transacção a utilizar nas comunicações com a base de dados. O programador é responsável pela criação, commit, rollback e libertação dos recursos dessa transacção e só pode utiliza-la no contexto da própria instância de DataContext.

```
try {
  dataContext.Connection.Open();
  dataContext.Transaction = data
Context.Connection.BeginTransaction();
dataContext.SubmitChanges(ConflictMode.
ContinueOnConflict);
  dataContext.Transaction.Commit();
}
catch (ChangeConflictException) {
  dataContext.Transaction.Rollback();
}
```

Outra opção e a mais recomendada é utilizar um objecto da classe TransactionScope. Este tipo de transacções adapta-se automaticamente aos objectos que manipula. Se fizer apenas acesso a uma base de dados utilizará uma transacção simples. Caso faça acesso a mais do que uma base de dados utilizará uma transacção distribuída. Outra vantagem é que não necessário iniciar ou fazer rollback a transacções deste tipo, apenas é necessário fazer commit. A operação de commit é realizada chamado o método Complete da classe TransactionScope e será feito o rollback automático da transacção caso o método Complete não seja chamado.

```
using (TransactionScope scope = new
TransactionScope())
{

dataContext.SubmitChanges(ConflictMode.
ContinueOnConflict);
  scope.Complete();
}
```

O leitor repare que se existir algum conflito de concorrência o método SubmitChanges irá lançar uma excepção fazendo com que o método Complete não seja chamado e a transacção sofrerá um rollback automático.

Implementar Concorrência Pessimista

Por vezes poderá ser necessário utilizar um esquema de concorrência pessimista. Um esquema de concorrência pessimista bloqueia os recursos durante a edição. Neste esquema de concorrência, os recursos permanecem bloqueados até ao fim da edição dos mesmos. A principal vantagem deste esquema é a garantia de que nenhum outro utilizador tem acesso de escrita aos recursos bloqueados. Como desvantagens temos principalmente duas. Para manter os recursos bloqueados é necessária uma ligação permanente à base de dados. Esta ligação permanente pode ser um recurso crítico como, por exemplo, aplicações Web com muitos utilizadores. O facto de os recursos ficarem bloqueados pode ser uma grande vantagem mas também poderá ser uma grande desvantagem. Imaginemos que um utilizador faz uma pausa para café ou para almoço. Os recursos ficam bloqueados durante todo esse tempo, mesmo que não tenham sido alterados ainda. A DLINQ fornece-nos uma forma muito simples de implementar este esquema de concorrência. Basta juntar a leitura e a escrita dos dados na mesma transacção. Para realizar esta tarefa vamos usar a classe TransactionScope e um bloco using, onde no fim do mesmo será feita a chamada ao método Complete da TransactionScope.

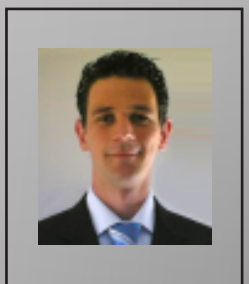
```
DatabaseDataContext dc = new
DatabaseDataContext();
using (TransactionScope t = new
TransactionScope())
{
    Cliente cliente = (from c in
dc.Clientes
                       where c.Nome ==
"Vitor"
                       select
c).Single();
    //ou
    //Cliente cliente =
dc.Clientes.Single(c => c.Nome ==
"Vitor");

    cliente.Cidade = "Torres Vedras";
    dc.SubmitChanges();
    t.Complete();
}
```

Usando o esquema de concorrência pessimista não existem conflitos para resolver visto que os recursos são bloqueado durante a transacção e portanto nenhum outro utilizador tem permissão os modificar.

Bibliografia

- Optimistic Concurrency Overview (LINQ to SQL) - http://msdn.microsoft.com/en-us/library/bb399373.aspx
- Introducing Microsoft LINQ - Microsoft Press
- LINQ in Action - Manning
- Pro LINQ Language Integrated Query in C# 2008 - Apress
- Professional LINQ - Wrox

SOBRE O AUTOR



Vitor Tomaz é estudante de Engenharia Informática e Computadores no ISEL bem como Trabalhador Independente na área de Tecnologias de Informação. Tem especial interesse por Programação Concorrente, Segurança Informática e Aplicações Web.

Vitor Tomaz

Grafos – 3ª Parte
Outras aplicações da pesquisa em profundidade

No seguimento da introdução ao mundo dos grafos (ver Edição 10) e termos apresentado alguns dos algoritmos mais usados em problemas baseados em grafos (ver Edição 12), pretende-se agora mostrar outras aplicações da pesquisa em profundidade. Nomeadamente, no âmbito da conectividade de grafos.

Tal como nos artigos anteriores, para facilitar a procura de informação adicional, os nomes dos algoritmos e das palavras-chave são apresentados em inglês. Por outro lado, vou usar a notação (v,u) para referir uma aresta de um vértice v para u.

Conectividade

Um grafo não dirigido diz-se conexo se a partir de qualquer vértice é possível atingir todos os outros vértices. Analogamente, um grafo dirigido com esta propriedade diz-se fortemente conexo.
Assim, um grafo não dirigido diz-se biconexo se não existem vértices cuja remoção torna o grafo desconexo. Ou seja, se considerarmos o grafo da figura 1, podemos observar que o grafo não é biconexo.

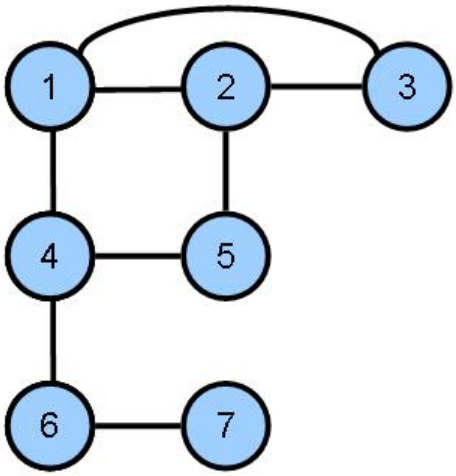


Figura 1: Exemplo de um grafo não dirigido

Se removermos o vértice 4, os vértices 6 e 7 ficam isolados dos restantes vértices. Assim, diz-se que o vértice 4 é um ponto de articulação. Por outro lado, se removermos a aresta (4,6) o grafo torna-se desconexo (novamente, os vértices 6 e 7 ficam isolados). Sendo assim, diz-se que a aresta (4,6) é uma ponte de articulação.

Exercícios

1.1) Escreva um programa que determina se um grafo não dirigido é conexo.

1.2) Escreva um programa que determina se um grafo dirigido é fortemente conexo.

Depth-first search

Na primeira parte (ver Edição 10), foi apresentado o essencial sobre a pesquisa em profundidade. Contudo, de modo a facilitar a compreensão dos algoritmos discutidos mais à frente neste artigo, é conveniente introduzir alguns conceitos adicionais.

Ao longo de uma pesquisa em profundidade, pode-se guardar o antecessor de cada vértice na pesquisa construindo assim uma árvore ou conjunto de árvores (floresta) de antecessores – depth-first tree/forest. Tome-se como exemplo um grafo dirigido com 7 vértices e a respectiva depth-first forest obtida através duma DFS iniciada no vértice 1 e assumindo que os vértices adjacentes são visitados por ordem crescente de número.

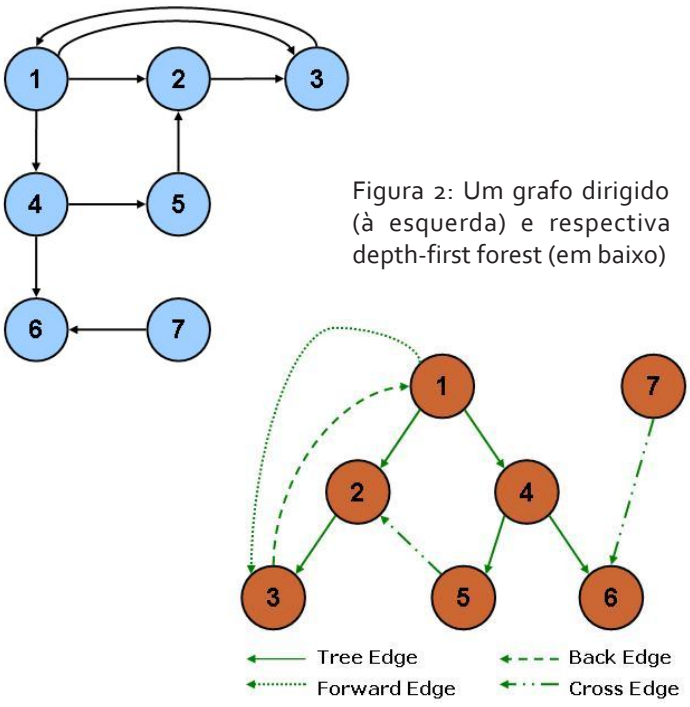


Figura 2: Um grafo dirigido (à esquerda) e respectiva depth-first forest (em baixo)

Assim, ao atingir um vértice u a partir de um vértice v através de uma aresta (v,u), o antecessor (pai) de u é v e (v,u) pertence à depth-first forest. Assim, considerando uma pesquisa num grafo dirigido, todas as arestas (v,u) do grafo podem ser classificadas em 4 tipos [1]:

1.Tree Edge – são as arestas que pertencem à depth-first forest.

2.Back Edge – são as arestas que ligam v a um antecessor u na depth-first tree de v.

3.Forward Edge – são as arestas que ligam v a um sucessor u numa depth-first tree, mas não pertencem à depth-first forest (ver exemplo).

4.Cross Edge – são todas as outras arestas que ligam vértices da mesma depth-first tree, desde que nenhum dos vértices seja antecessor do outro, ou ligam vértices de depth-first trees diferentes. Ver arestas (5,2) e (7,6), respectivamente.

Por outro lado, é conveniente manter uma variável de tempo e guardar para cada vértice o seu tempo de descoberta (quando é iniciada a pesquisa nesse vértice) e o tempo de fim de processamento.

Sendo assim, chegamos ao seguinte pseudo-código:

```
SearchGraph()
    Para cada vértice v
        marcar v como não visitado
        pai[v] = nil
    tempo = 0
    Para cada vértice v
        Se v não foi visitado
            DFS(v)

DFS(v)
    marcar v como visitado
    d[v] = tempo = tempo + 1
    Para cada u adjacente a v
        Se u não foi visitado
            pai[u] = v
            DFS(u)
    f[v] = tempo = tempo + 1
```

Exercícios

2.1) Em que condições se pode obter uma floresta de árvores de antecessores, em vez de uma única árvore?

2.2) Que tipos de arestas existem num grafo não dirigido? Porquê?

2.3) Reescreva a função DFS() para que classifique devidamente o tipo das arestas encontradas. Dica: use convenientemente os tempos de descoberta e de fim de processamento dos vértices.

Pontos e Pontes de Articulação

Agora o problema é dado um grafo não dirigido, determinar os pontos e pontes de articulação. Inicialmente, a abordagem natural é utilizar força bruta: retirar cada vértice/aresta do grafo e verificar se o grafo se mantém conexo. Esta abordagem teria um tempo de execução de O(V*(V+E)) e O(E*(V+E)), respectivamente. Será que podemos fazer melhor? Felizmente, sim! É possível determinar os pontos e pontes de articulação em simultâneo e em tempo linear – O(V+E) - com recurso à pesquisa em profundidade.

O algoritmo consiste numa pesquisa em profundidade, marcando os tempos de descoberta dos vértices tal como apresentado anteriormente. Ao longo da DFS, para cada vértice v, é determinado qual é o vértice w possível atingir a partir de v (ao encontrar uma aresta para um vértice já visitado, é usado o tempo de descoberta desse vértice, mas este não é revisitado), com menor tempo de descoberta (low[v]=d[w]).

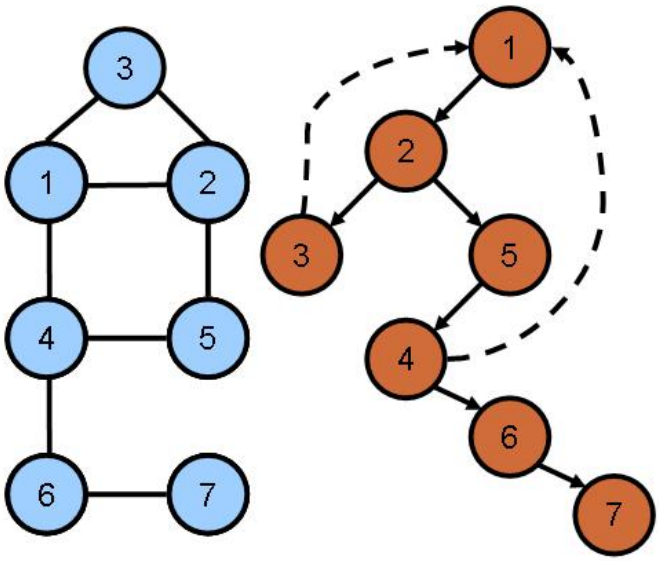


Figura 3: Um grafo não dirigido (à esquerda) e respectiva depth-first tree (à direita)

Sendo assim, um vértice v é um ponto de articulação se tem algum sucessor w tal que low[w] >= d[v]. Ou seja, não existe nenhuma ligação entre os sucessores de v com os seus antecessores (observar na figura 3 que os vértices 4 e 6 são pontos de articulação). Qualquer ligação entre estas 2 partes tem de passar por v. Esta condição provoca que a raiz da pesquisa é sempre considerada um ponto de articulação (low[raiz]=d[raiz]=1). Assim, é necessário um teste adicional:

a raiz da pesquisa só é um ponto de articulação se tiver mais do que um filho na árvore de pesquisa, pelo mesmo motivo referido atrás: se isto acontecer, a raiz divide o grafo em subgrafos, onde é obrigatória a passagem pela raiz para os ligar.

Por outro lado, uma aresta (v,w) é uma ponte de articulação se low[w] = d[w]. Isto é, não existe nenhuma ligação entre w ou os seus sucessores na depth-first tree e os antecessores de w. Assim, esta aresta é a ponte entre o subgrafo atingido a partir de w e o resto do grafo (ver aresta (4,6) da figura 3).

Por fim, apresenta-se o pseudo-código do algoritmo (para uma demonstração da correcção do algoritmo, ver [1] ou [2]):

```
DFS(v , pai_v)
     low[v] = d[v] = tempo = tempo+1
     num_filhos = 0
     Para cada vizinho w de v
          Se d[w] = infinito
               num_filhos = num_filhos + 1
               DFS(w , v)
               low[v] = min(low[v],low[w])
               Se low[w] >= d[v]
                    v é ponto de articulação
                    Se low[w] = d[w]
                         (v,w) é ponte
          Senão Se w ≠ pai_v
                    low[v] = min(low[v],d[w])
     retorna num_filhos

EncontraPontosPontes()
     tempo = 0
     Para cada vértice v
          d[v] = infinito
     Para cada vértice v
          Se v não foi visitado e DFS(v) < 2
               v não é ponto de articulação
```

Bibliografia

[1] Cormen et al, Introduction to Algorithms, 2ª Edição, MIT Press 2002

[2] Weiss, Mark Allen, Data Structures & Algorithm Analysis in C++, 2ª Edição, Addison Wesley, 1999

SOBRE O AUTOR

O interesse de Miguel Oliveira pela algoritmia despertou no secundário, sendo cultivado na FEUP onde frequenta o 4º ano do MIEIC. Deste interesse surgem várias participações em concursos, destacando-se o 11º lugar no South Western European Regional Contest 2007 (melhor equipa portuguesa). Recentemente, foi monitor no estágio de preparação das Olimpíadas de Informática.

Miguel Oliveira

Algoritmos para o Cálculo de Dígito Verificador

Introdução

O cálculo de dígito verificador (em inglês check digit) é um procedimento algorítmico matemático que visa garantir a integridade (evita entradas inválidas de dados) e autenticidade (diminui acções fraudulentas) no uso de um código numérico, alfabético ou alfanumérico único (chave primária) para a identificação dos dados de um registo dentro de um sistema de informação. Entende-se como registo o conjunto de dados relacionados a um indivíduo, empresa, produto ou serviço que serão administrados em um sistema de armazenamento de dados.

O processo de uso de dígito verificador ocorre no mínimo com a definição de um valor numérico (entre os valores 0 e 9) acrescentado ao corpo do código de identificação tornando-se parte deste código de identificação. Normalmente o dígito verificador é inserido ao final do código de identificação. Dependendo da necessidade poderá ser acrescentado a um código de identificação mais de um dígito verificador.

O uso de dígito verificador ocorre em uma ampla gama de aplicações. É comum fazer uso deste procedimento na definição de números de matriculas (escolas, clubes, órgãos governamentais, etc.), em números de contas correntes bancárias, em números de cartões de crédito, na identificação de publicações de livros (ISBN – International Standard Book Number), na identificação de revistas e periódicos (ISSN – International Serial Standard Number), códigos de barra, entre outras possibilidades.

Este artigo objectiva apresentar mecanismos algorítmicos utilizados para a obtenção de dígitos verificadores com códigos de identificação numéricos e alfanuméricos sem focar a especificidade de alguma necessidade particular. Ao final deste será demonstrado o procedimento de cálculo e obtenção de mais de um dígito verificador (dois dígitos) para um código de identificação.

Métodos Algorítmicos

O método algoritmo a ser usado para a definição e obtenção de dígito verificador pode ser de cunho particular e desenvolvido de acordo com necessidades específicas ou pode ser utilizado um algoritmo já consagrado. Dos métodos utilizados para tal finalidade há dois que podem ser facilmente integrados, sendo os algoritmos de módulo 10 e de módulo 11. Os algoritmos de módulo 10 (HP, 2000. p. 580) e módulo 11 (HP, 2000. p. 581) são utilizados para a geração de dígitos verificadores para códigos de identificação numéricos. Para os códigos de identificação alfanuméricos ou alfabéticos (HP, 2000. p. 582) faz-se normalmente a substituição das letras usadas no código por um valor de referência definido para cada letra do alfabeto em uma tabela de valores para em seguida aplicar um dos métodos algorítmicos de módulo 10 ou módulo 11.

Dígito Verificador para Código Numérico

O dígito verificador ou simplesmente dv de um código numérico é normalmente gerado por meio do uso de um dos algoritmos módulo 10 e módulo 11. Os módulos 10 e 11 são a definição do resultado do resto da divisão por 10 ou por 11 de um valor numérico obtido por meio do somatório do resultado da multiplicação de um valor de peso com as partes componentes unitárias (dígito) de um código numérico.

- Algoritmo: Dígito Verificador Módulo 10

O cálculo do dígito verificador obtido por meio do módulo 10 é efectuado pegando-se o código numérico e relacionando-se abaixo deste código da direita para a esquerda os valores de peso sucessivamente de forma que cada dígito componente do código seja multiplicado pelos valores de peso. Quando o valor do resultado da multiplicação do valor de peso com o dígito componente do código numérico for maior ou igual a 10, ou seja, um valor com dois dígitos este valor deverá gerar outro valor de apenas um dígito, que será obtido a partir da soma da unidade com a dezena. Assim sendo, se o valor for 10 deverá ser usado 1, se for 11 deverá ser usado 2, se for 12 deverá ser usado 3 e assim sucessivamente.

Em seguida, devem ser somados os dígitos obtidos com a multiplicação do valor de peso com o valor de cada dígito componente do código numérico. O resultado obtido desta soma deve ser dividido por 10 a fim de gerar um resultado de quociente inteiro para então saber o valor do resto da divisão por 10 (módulo 10). Em seguida faz-se a subtracção de 10 o valor do resto da divisão e o valor obtido será o dígito verificador.

No sentido de demonstrar a aplicação do algoritmo descrito anteriormente considere o código numérico de identificação 987654 que deverá ter calculado um dígito verificador a ser incorporado no final do código numérico de identificação, a partir do uso do valor de peso 12. Assim sendo, considere:

Algoritmo: Módulo 10						
Código numérico	9	8	7	6	5	4
Peso multiplicador	x 1	x 2	x 1	x 2	x 1	x 2
Resultado parcial	9	16	7	12	5	8
Ajuste quando >= 10	-	7	-	3	-	-
Resultado final	9	7	7	3	5	8
Σ do resultado final	39					
Divisão do Σ por 10	quociente = 3			resto = 9		
DV = 10 – resto	1					

Desta forma, o código de identificação fica definido como 987654-1 (ou 9876541), sendo o primeiro trecho o número do código de identificação e o segundo trecho o valor do dígito verificador.
O formato descrito com o uso do valor de peso 12 é o mecanismo padrão para a obtenção de dígito verificador em módulo 10. No entanto, pode-se a partir de uma necessidade particular alterar a forma de distribuição dos valores de peso, podendo se utilizar outras combinações, tais como: 13, 14, 15, 16, 17, 123, 1234, ou qualquer outra forma de limitando-se a sequência máxima 123456789. Pode-se também fazer uso de um valor de peso fixo em toda a extensão do código de identificação.
A título de demonstração acompanhe o cálculo do dígito verificador em módulo 10 do código de identificação 987654 utilizando-se peso 1234.

Algoritmo: Módulo 10						
Código numérico	9	8	7	6	5	4
Peso multiplicador	x 3	x 4	x 1	x 2	x 3	x 4
Resultado parcial	27	32	7	12	15	16
Ajuste quando >= 10	9	5	-	3	6	7
Resultado final	9	5	7	3	6	7
Σ do resultado final	37					
Divisão do Σ por 10	quociente = 3			resto = 7		
DV = 10 – resto	3					

Desta forma, o código numérico fica definido como 987654-3 (ou 9876543).

- Algoritmo: Dígito Verificador Módulo 11

O cálculo do dígito verificador obtido por meio do módulo 11 é de certa forma muito parecido com o cálculo obtido pelo algoritmo de módulo 10. Após relacionar abaixo do código de identificação da direita para a esquerda os valores de peso sucessivamente até completar a sequência total de dígitos que compõem o código de identificação cada dígito componente do código numérico deverá ser multiplicado pelos valores de peso. Em seguida, deve-se somar os dígitos obtidos com a multiplicação do valor de peso com o valor de cada dígito componente do código numérico. O resultado obtido desta soma deve ser multiplicado por 10 e divido por 11 a fim de gerar um resultado de quociente inteiro para então saber o valor do resto da divisão por 11 (módulo 11). O valor do resto da divisão é o valor do dígito verificador.
O cálculo de dígito verificador em módulo 11 pode gerar um dv com valor 10. Neste caso, fica a critério do usuário deste algoritmo usar o valor 10 ou substituir este valor por outro valor de 0 até 9, ou mesmo usar uma letra em seu lugar. Uma prática comum é substituir o valor 10 pela letra X que é na prática o valor 10 em algarismo romano. Há casos em que o valor 10 é considerado 1 usando-se a mesma regra usada para o cálculo do multiplicador em módulo 10.
No sentido de demonstrar a aplicação do algoritmo descrito anteriormente considere o código numérico de identificação 987654 que deverá ter calculado um dígito verificador a ser incorporado no final do código numérico de identificação. Assim sendo, considere:

Algoritmo: Módulo 11						
Código numérico	9	8	7	6	5	4
Peso multiplicador	x 7	x 6	x 5	x 4	x 3	x 2
Resultado parcial	63	48	35	24	15	8
Σ do resultado final	193					
Σ x 10	193 x 10 = 1930					
Divisão do Σ x 10 por 11	quociente = 175			resto = 5		
DV = resto	5					

Desta forma, o código de identificação fica definido como 987654-5 (ou 9876545), sendo o primeiro trecho o número do código de identificação e o segundo trecho o valor do dígito verificador.
O formato descrito com o uso do valor de peso 234567 (que pode chegar até a sequência 23456789) é o mecanismo padrão para a obtenção de dígito verificador em módulo 11. Assim como no módulo 10, pode-se a partir de uma necessidade particular alterar a forma de distribuição dos valores de peso.
A título de demonstração acompanhe o cálculo do dígito verificador em módulo 10 do código de identificação 987654 utilizando-se o valor de peso fixo 8.

Algoritmo: Módulo 11						
Código numérico	9	8	7	6	5	4
Peso multiplicador	x 8	x 8	x 8	x 8	x 8	x 8
Resultado parcial	72	64	56	48	40	32
Σ do resultado final	312					
Σ x 10	312 x 10 = 3120					
Divisão do Σ x 10 por 11	quociente = 283			resto = 7		
DV = resto	7					

Desta forma, o código numérico fica definido como 987654-7 (ou 9876547).

Dígito Verificador para Código Alfanumérico

Os cálculos de dígito verificador obtido para um código de identificação alfanumérico ou mesmo alfabético é conseguido a partir da distribuição do abecedário de A até Z e sua numeração com valores de sequências de 1 até 9 a partir da letra A reservando o valor o para representar o uso do espaço em branco quando este existir. Desta forma, ter-se-á dois segmentos completos de 1 até 9 para as sequências de letras de A até I e de J até R e um segmento incompleto de 2 até 9 para a sequência de letras de S até Z. Assim sendo, observe o seguinte:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 2 3 4 5 6 7 8 9
```

A partir da tabela de letras e valores faz-se a substituição da letra usada por seu valor correspondente na sequência do código de identificação e aplica-se um dos métodos de obtenção de dígito verificador módulo 10 ou módulo 11.

Após obter o valor do dígito verificador anexa-se este valor ao final da sequência do código de identificação alfabética ou alfanumérica. Não é aconselhável fazer uso do valor do dígito verificador na sua forma alfabética.

No sentido de demonstrar a aplicação do algoritmo descrito anteriormente considere o código alfanumérico de identificação ABC987 que deverá ter calculado um dígito verificador a ser incorporado no final do código numérico de identificação. Assim sendo, considere:

Algoritmo: DV a partir de código alfanumérico c/ Módulo 11						
Código numérico	**A**	**B**	**C**	**9**	**8**	**7**
Equivalência numérica	**1**	**2**	**3**	-	-	-
Sequência numérica	**1**	**2**	**3**	**9**	**8**	**7**
Peso multiplicador	**x 7**	**x 6**	**x 5**	**x 4**	**x 3**	**x 2**
Resultado parcial	7	12	15	36	24	14
Σ do resultado final	**108**					
Σ x 10	**108 x 10 = 1080**					
Divisão do Σ x 10 por 11	**quociente = 98**			**resto = 2**		
DV = resto	**2**					

Assim sendo, o código alfanumérico fica definido como ABC987-2 (ou ABC9872).

Utilização de mais de um Dígito Verificador

Imagine como situação hipotética a necessidade de um estabelecimento de ensino definir um código de identificação para o seu número de matricula a fim de ter uma chave primária para controle de seus alunos em seu sistema de informação, de forma que o número de matricula seja formado pelo ano de ingresso na instituição e do número serial de matricula no ano.

A informação do ano de ingresso na instituição será representada de forma alfanumérica e a informação do número serial de matrículas, a partir de 1 será representada na forma numérica. Por exemplo, um aluno matriculado no ano de 2008 sob número de matrícula 150 terá seu cadastro identificado pelo código B00H0150 mais o dv de duas posições que será inserido no início do código de matrícula como parte integrante deste código.

Observe que o código B00H0150 é formado com as letras B e H, sendo B equivalente ao valor 2 e a letra H equivalente ao valor 8, as quatro últimas posições do código 0150 representa o número de matrícula efectuada no respectivo ano. Os valores zero entre as letras B e H não possuem representação literal na tabela alfanumérica apresentada no tópico anterior e por esta razão permanecem com o valor como zero.

O dígito verificador do número de matrícula será calculado em duas etapas. A primeira etapa calcula o primeiro dígito verificador com base no algoritmo de módulo 10 com peso 12 e a segunda etapa calcula o segundo dígito verificador com base no algoritmo de módulo 11 com peso 23.

Observe a seguir a demonstração de cálculo do primeiro dígito verificador do código de matrícula:

Cálculo do primeiro dígito verificador – Módulo 10								
Código numérico	**B**	**0**	**0**	**H**	**0**	**1**	**5**	**0**
Substituição em valores	**2**	-	-	**8**	-	-	-	-
Peso multiplicador	**x 1**	**x 2**	**x 1**	**x 2**	**x 1**	**x 2**	**x 1**	**x 2**
Resultado Parcial	**2**	**0**	**0**	**16**	**0**	**2**	**5**	**0**
Ajuste quando >= 10	-	-	-	7	-	-	-	-
Resultado final	**2**	**0**	**0**	**7**	**0**	**2**	**5**	**0**
Σ do resultado final	**16**							
Divisão do Σ por 10	**quociente = 1**				**resto = 6**			
DV = 10 – resto	**4**							

A seguir anexa-se o primeiro dígito verificador, neste caso 4 ao número do código de matrícula de forma que este fique configurado como 4B00H0150.

A próxima etapa para o cálculo do segundo dígito verificador

consiste em pegar o novo valor e distribuí-lo da seguinte forma:

Cálculo do segundo dígito verificador – Módulo 11									
Código numérico	4	B	0	0	H	0	1	5	0
Substituição em valores	-	2	-	-	8	-	-	-	-
Peso multiplicador	x 3	x 2	x 3	x 2	x 3	x 2	x 3	x 2	x 3
Resultado Parcial	12	4	0	0	24	0	3	10	0
Σ do resultado final	53								
Σ x 10	53 x 10 = 530								
Divisão do Σ x 10 por 11	quociente = 48				resto = 2				
DV = resto	2								

A seguir anexa-se o primeiro dígito verificador, neste caso 2 ao número do código de matrícula de forma que este fique configurado como 24B00H0150.

Note que foram utilizados os dois paradigmas para a obtenção dos dígitos verificadores do código de matrícula.

Referência Bibliográfica

HEWLETT PACKARD. HP Data Entry and Forms Management System (VPLUS): Reference Manual – HP 3000 MPE/iX Computer System. 6. ed. USA, 2000. 697 p.

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem 23 anos de experiência em ensino e desenvolvimento de programação de software. É professor da rede federal de ensino no Brasil, no Centro Federal de Educação Tecnológica de São Paulo. É também autor, possuindo na sua carreira mais de cinquenta obras publicadas.

Augusto Manzano

Fundamentos de Segurança em Redes (Parte I)

A segurança em redes é uma área cada vez mais complexa á medida que surgem por sua vez mais ameaças e a um ritmo acelerado. Daí que o objectivo da segurança em redes é ter um conjunto de medidas e soluções com o objectivo de proteger uma rede, proteger a sua integridade, proteger a informação contida nela e proteger os utilizadores contra uma vasta variedade de ameaças.

A Internet deixou de ser algo exclusivo para certas áreas (cientificas e militares) e passou a ser uma ferramenta essencial para o quotidiano da maior parte das pessoas. Algumas das aplicações mais usadas, como os Web Browser, FTP, E-mail, VoIP são parte essencial das redes que existem que usamos diariamente e que facilitam em muito a maneira como comunicamos e interagirmos com outros. Mas junto com estas grandes vantagens surgiu uma grande responsabilidade, a Segurança das Redes.

Como podemos tornar as nossas redes mais seguras?

Um bom esquema de segurança envolve um planeamento exaustivo para podermos determinar onde é necessário maior segurança e que soluções são as mais indicadas para a nossa rede. Esse planeamento envolve levar em conta três aspectos básicos: Confidencialidade (privacidade), Integridade (proteger os dados de modificações não autorizadas) e Acessibilidade. Assim, eis algumas coisas que devemos levar em consideração quando pensamos em proteger o melhorar a protecção da nossa rede.

Defesa em camadas

Certamente, não é preciso referir os diversos tipos de ameaças a que uma rede está sujeita sejam essas ameaças exteriores ou interiores. Ora, esta variedade de ameaças leva a uma situação em que apenas uma camada de protecção não é o suficiente para proteger uma rede. Não basta dizer, "Mas eu tenho uma firewall". A segurança de algo não se resume apenas á instalação de determinados software e ficarmos descansados. Antes devemos encarar a segurança de uma rede, como sendo uma solução de diversas camadas, cada uma com o seu objectivo e levar em conta que é também necessário recorrer a soluções tanto baseadas em hardware como em software.

Perímetro de Defesa

Apesar de existir uma arquitectura de redes aparentemente unificada e global, certas áreas exigem que haja segmentos de redes. Segmentos esses que são estabelecidos por dispositivos capazes de regular e controlar o que entra e o que sai. Ao criarmos um bom perímetro de defesa devemos basear as nossas medidas em algumas áreas básicas. Devemo-nos preocupar com o Controlo de Acesso usando para isso um firewall que cria uma fronteira\perímetro que controla os acessos. A Autenticação também é necessária e é usada em conjunto com o Firewall, dado que o papel dela é verificar quem é que vai passar pela Firewall e verificar se essa pessoa é quem diz ser. A utilização de um Intrusion Detection, também é importante para olhar para a Firewall e ver se algo de anormal se está a passar e se tal acontecer alertar para esse problema.

Outra área é a segurança dos conteúdos a que os utilizadores podem aceder. E para isso é necessário que aliada á Firewall esteja uma outra camada que tenha como objectivo verificar onde os utilizadores vão na Internet. Terá que ser uma aplicação que tenha a capacidade de fazer análise aos ficheiros e que consiga também bloquear endereços (URL's). Por fim, no caso de existir a necessidade de acessos remotos é também necessário que exista uma encriptação do tráfego gerado entre os dois pontos.
Mas antes de focarmos a nossa atenção, nestes componentes importantes para a protecção da nossa rede, vamos ver como podemos melhorar a segurança de uma rede através do seu "desenho".

Subnetting

Uma forma de criarmos um bom perímetro de defesa é a utilização de técnicas usadas em arquitecturas de redes e que além de ajudarem a poupar endereços IP também tem algo a dizer no que toca a segurança. Uma dessas técnicas é o IP Subnetting o processo de dividir uma rede IP em pequenas sub-redes chamadas de subnets.

Como é que podemos criar uma subnet?

Para calcularmos uma subnet e também para entender com funciona este cálculo temos de em primeiro lugar levar identificar a classe em que se insere o IP. Ora existem 5 classes, mas apenas 3 são usadas, vamos então ver quais é que são:

```
Network Address Range Class A
```

Os desenhadores do esquema de endereços IP definiram que o primeiro bit do primeiro byte da classe A tem que estar sempre off ou seja 0. Significa então que a Classe A tem valores compreendidos entre 0 e 127, inclusive. Ou seja de:

```
00000000 = 0 a 01111111 = 127
```

```
Network Address Range Class B
```

Numa classe B foi definido que o primeiro bit do primeiro byte deve estar sempre on ou 1, mas que o segundo bit deve estar sempre off. Portanto teremos valores de 128 a 191.

```
10000000 = 128 a 10111111 = 191
```

```
Network Address Range Class C
```

Numa classe C, foi definido que os primeiros 2 bits do primeiro byte devem estar sempre on, mas o terceiro bit deve estar sempre off. Teremos assim valores de 192 a 223.

```
11000000 = 192 a 11011111 = 223
```

Existe ainda mais duas classes, a classe D que vai do 224 a 239 que é usada como multicast address e portanto são endereços reservados. E a classe E que vai dos valores 240 a 255 que são usados para propósitos científicos.

Mas ainda existem alguns endereços reservados:

```
0.0.0.0
```

```
1.1.1.1
```

```
127.0.0.1
```

```
255.255.255.255
```

Outro caso em que temos endereços reservados é por exemplo, quando temos o endereço 127.2.0.0 e 127.2.255.255 estes dois endereços estão reservados, os únicos endereços que poderiam ser usados são do 127.2.0.1 a 127.2.255.254. Ou seja quando os "nodes" são todos 0 ou quando os "nodes" são todos 1 não se pode usar esses endereços. Na figura em baixo podem ver quais são os "nodes" de cada classe.

	8 bits	8 bits	8 bits	8 bits
Classe A	Network	Node	Node	Node
Classe B	Network	Network	Node	Node
Classe C	Network	Network	Network	Node

- Subnet Masks

Para que o esquema de subnet address funcione é necessário que cada máquina na rede saiba qual a parte do host\node address irá ser usada como subnet address. E faz isso por designar uma subnet mask a cada máquina. Uma subnet mask tem um valor de 32 bit e que permite ao host quando recebe um pacote IP distinguir qual é a parte de network ID e qual é a parte de host ID. Mas nem todas as redes necessitam de subnets o que significa que usam as subnet mask por defeito e são elas:

Class	Format	Subnet mask (por defeito)
A	network.node.node.node	255.0.0.0
B	network.network.node.node	255.255.0.0
C	network.network.network.node	255.255.255.0

Vamos pegar num pequeno exemplo de como se calcula as subnets para um determinado endereço IP da classe C.

Estas são as masks disponíveis para a Classe C.

255.255.255.128	/25	11111111.11111111.11111111.100000000
255.255.255.192	/26	11111111.11111111.11111111.110000000
255.255.255.224	/27	11111111.11111111.11111111.111000000
255.255.255.240	/28	11111111.11111111.11111111.111100000
255.255.255.248	/29	11111111.11111111.11111111.111110000
255.255.255.252	/30	11111111.11111111.11111111.111111000

Começamos então pela Classe C, porque é mais fácil de se entender ao fazermos um cálculo. Usando a maneira mais rápida existem 5 perguntas que devem ser respondidas e quando todas elas forem respondidas temos então o subnetting feito.

Por exemplo no 192.168.10.0 /26

1. Quantas subnets é possível ter com a subnet masks escolhida? Ora a subnet mask é a /26:

11111111.11111111.11111111.110000000 (255.255.255.192)

Lembrem-se que o endereço 192.168.10.0 pertence à classe C. Na classe C o node address é que interessa, por isso do

11111111.11111111.11111111.110000000 apenas nos interessa o último byte, 11000000.

Pegando neste valor 11000000, quantos bits é que temos on? Temos 2.

Quanto é que é 2^ 2? É 4, por isso temos 4 subnets.

2. Quantos hosts temos por subnet?

Bem pegando novamente no 11000000, quantos bits temos off? Temos 6, portanto:

2^ 6 – 2 = 62

Temos então 62 hosts por cada subnet.

3. Quais são as subnets válidas?

Bem para fazermos o cálculo existe uma constante que é o 256. O 11000000 dá o valor de 192 e este valor que vamos precisar para responder á pergunta. 256-192 = 64.

Começamos do 0 a contar por isso 0, 64, 128 e 192 ou seja 0+64=64, 64+64=128, 128+64=192 e pára no 192 em virtude de ser esse o valor da nossa subtnet.

4. Qual é o broadcast address de cada subnet e quais são os hosts válidos?

Para respondermos a esta pergunta o quadro que se segue vai ajudar-nos. Primeiro começamos por preencher o campo das subnets. E gira tudo à volta disso, reparem:

Subnet	**192.168.10.0**	**192.168.10.64**	**192.168.10.128**	**192.168.10.192**
Primeiro host	192.168.10.1	192.168.10.65	192.168.10.129	192.168.10.193
Ultimo host	192.168.10.62	192.168.10.126	192.168.10.190	192.168.10.254
Broadcast address	192.168.10.63	192.168.10.127	192.168.10.191	192.168.10.255

- Mas qual é o objectivo de uma subnet em concreto?

Uma subnet tem como objectivo reduzir o tráfego numa rede, optimizando o desempenho de uma rede e simplificando a sua gestão e de certa forma aumentando a sua segurança.

Ao usarmos subnets estamos a organizar os hosts em grupos lógicos, estamos a dividir uma rede em diversas partes e usamos router's para conectá-las aumentando assim a segurança. Usando um router podemos também recorrer às access-list.

- O que é uma access-list?

É algo do género:

RouterCisco(config)#access-list 10 deny 172.16.40.0 0.0.0.255

RouterCisco(config)#access-list 10 permit any

E qual é o objectivo?

Bem, ao usarmos access-list ganhamos o controlo sobre o tráfego que passa pela nossa rede, podendo implementar politicas de segurança assim como obter estatísticas básicas do fluxo dos pacotes podendo até mesmo permitir ou proibir que certos pacotes passem pelo router. Podemos assim proteger dispositivos ou redes sensíveis a acessos não autorizados. O exemplo acima é apenas um pequeno exemplo do que poderá ser uma access-list. Como podemos ver é uma lista usada para filtrar o tráfego na rede por examinar o endereço de origem no pacote.

Uma access-list poderá ser usada para filtrar o tráfego de saída, permitindo apenas a saída de pacotes que sejam endereçados a IP's que nós escolhemos como sendo válidos. Mas estas talvez sejam já opções mais avançadas para quando a nossa rede atinge um tamanho considerável e é necessária uma maior segurança.

Mas uma access-list quando bem usada e implementada é uma ferramenta bastante poderosa.

Switching e VLANs

Switching usando a camada 2 do modelo OSI, consiste no processo de segmentar uma rede usando para isso diversas características dos switchs e dessa camada do modelo OSI. Baixa latência, baixo custo, maior velocidade e bridging baseado em ASIC são alguns dos benefícios que podemos usufruir ao usarmos switches na nossa rede. Outra vantagem da utilização de switch's é que não existe modificação dos pacotes como acontece em router, apenas é lida a frame o que acelera o processo de switching e evita erros, mas esse é um outro assunto à parte.

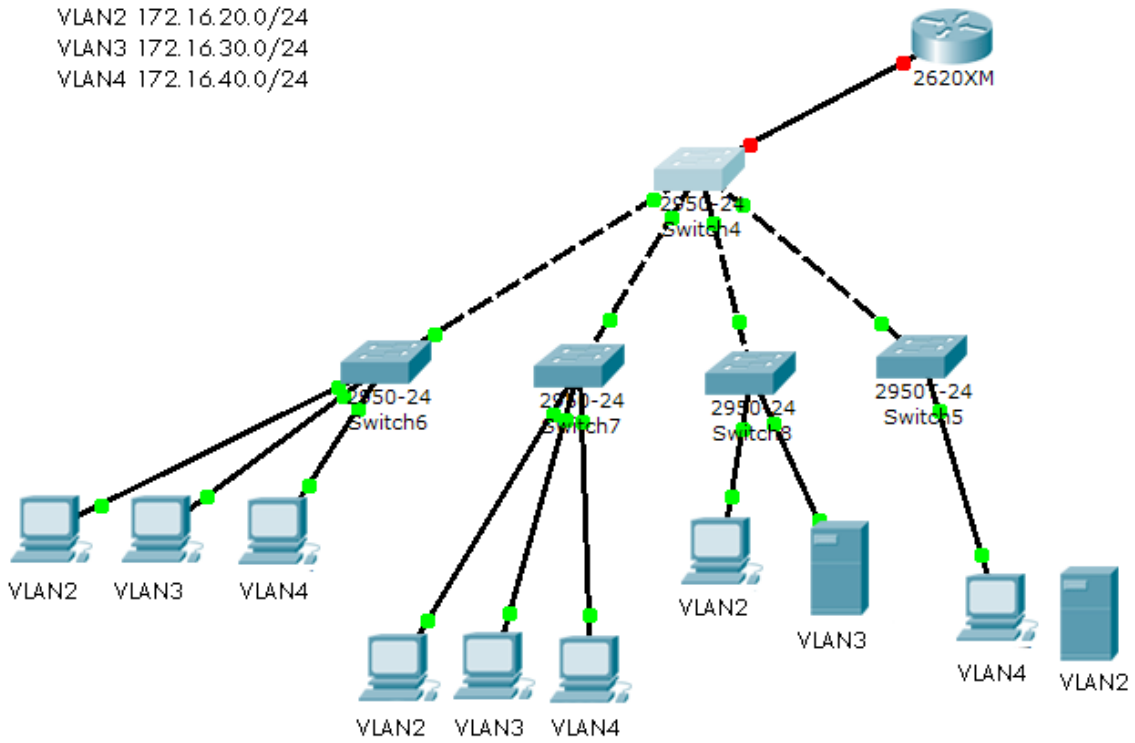
Com switchs também é possível a utilização de um conceito de arquitectura chamada de VLAN – Virtual Local Area Network.

Em termos de segurança o que é que podemos dizer? Bem, numa rede dita normal, qualquer pessoa que se conecte fisicamente à rede consegue ter acesso à rede e aos seus recursos, consegue ligar um network analyzer e consegue verificar o tráfego da rede.

Mas ao usarmos VLAN, criamos diversos broadcast groups, controlando cada porta junto com os recursos disponíveis assim como a cada utilizador.

Por defeito membros numa VLAN não consegue comunicar com uma outra VLAN, no entanto é possível fazer isso. Recorrendo a um router podemos ter comunicação entre diversas VLANS e podemos implementar restrições no que toca a protocolos, endereços MAC e aplicações.

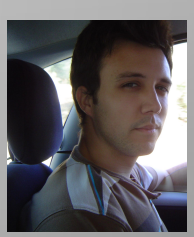
Através desta imagem, podemos verificar que os membros de uma VLAN não precisam de estar necessariamente ligados ao mesmo switch. Assim por exemplo os membros da VLAN3 podem sem problemas aceder ao servidor que se encontra ligado a um outro switch e neste caso o membro da VLAN2 apesar de estar ligado ao mesmo switch, não consegue ter acesso ao servidor. A menos que exista alguma permissão da parte do router. Como podemos ver a utilização de VLANs aumenta em muito a segurança de uma rede.



Referências / Links de apoio

Sybex CCNA Cisco Certified Network Associate Study Guide

SOBRE O AUTOR



Residente em Lisboa, Ciro Cardoso é um grande amante do vasto mundo que é a informática em especial a segurança. Está actualmente a trabalhar na próxima etapa do percurso Cisco com a certificação CCNP. Gosta bastante da área de redes, configuração de protocolos assim como a implementação.

Ciro Cardoso

Barcamp Portugal 2008 Coimbra

A realizar-se no nosso país, em Coimbra, desde 2006, o Barcamp teve este ano outra edição.

Definindo em poucas palavras, o Barcamp (http://barcamp.org) trata-se de um evento, com modelo de conferência, onde os participantes trazem as apresentações, workshops, e outros, aos restantes. Acima de tudo, a confraternização entre os participantes é sem dúvida o aspecto marcante da conferência.

Como tem vindo a ser habito, o pessoal da WeBreakStuff (http://webreakstuff.com) organizou uma vez mais um evento excelente. O espaço onde o Barcamp tem vindo a decorrer em Coimbra - o Pólo II da Universidade de Coimbra - não podia ser melhor, oferecendo um excelente espaço para conferências e também um espaço exterior agradável, importante para as "conversas de café"!

As apresentações deste ano tocaram em diversos assuntos, não estando todas elas relacionadas com a Internet, como por exemplo Gestão de Projectos com Scrum, onde foram dados os básicos sobre Scrum, ou um novo evento apresentado no Barcamp - o Social Media Cafe (http://lisbon.socialmediacafe.net) - que pretende reunir no mesmo espaço bloggers e outros para discutir, numa conversa de café, os Social Media em Portugal. Porém a grande maioria das apresentações está relacionada com Internet, e pudemos assistir a excelentes apresentações a falar sobre XMPP, RDFa, Ruby on Rails, Design e usabilidade, entre outros.

As apresentações foram gravadas em vídeo, e espera-se que as mesmas sejam disponibilizadas mais tarde, para aqueles que quiserem assistir no site do Barcamp Portugal.

Para aqueles que quiserem consultar o site do evento: http://barcamppt.org/

SOBRE O AUTOR

Natural de Peniche, Pedro Diogo foi desde sempre um apaixonado por tecnologia. Neste momento encontra-se a estudar Engenharia Electrotécnica e de Computadores no IST. Tem especial interesse por Programação Web, mais propriamente Ruby on Rails.

Pedro Diogo

GNOME Documentation Library

http://library.gnome.org/devel/

A biblioteca de documentação do GNOME disponibiliza diversos recursos a programadores e designers, como guias, referências de APIs e outras informações que ajudam ao desenvolvimento das suas aplicações.

Web Cheat Sheet

http://www.webcheatsheet.com/

No WebCheatSheet são disponbilizados vários tutoriais, artigos técnicos sobre as várias linguages de programação utilizadas na construcção de um website assim como ferramentas específicas para bases de dados.

Jeroen Wijering

http://www.jeroenwijering.com/

Website pessoal do developer Jeroen Wijering onde, entre outros projectos, é possível encontrar o JW Player, o reprodutor de vídeo mais utilizado na Internet hoje em dia.

jQuery

http://jquery.com/

O jQuery é uma biblioteca JavaScript rápida de propósito geral desenhada para mudar a forma como se programa em JavaScript. Permite o controlo de eventos, facilita o uso de AJAX, entre muitas mais funções através dos diversos plugins que existem para a mesma.

Queres participar na Revista PROGRAMAR? Queres integrar este projecto, escrever artigos e ajudar a tornar esta revista num marco da programação nacional?

Vai a

www.revista-programar.info

para mais informações em como participar,
ou então contacta-nos por

revistaprogramar
@portugal-a-programar.org

Precisamos do apoio de todos para tornar este projecto ainda maior...

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org