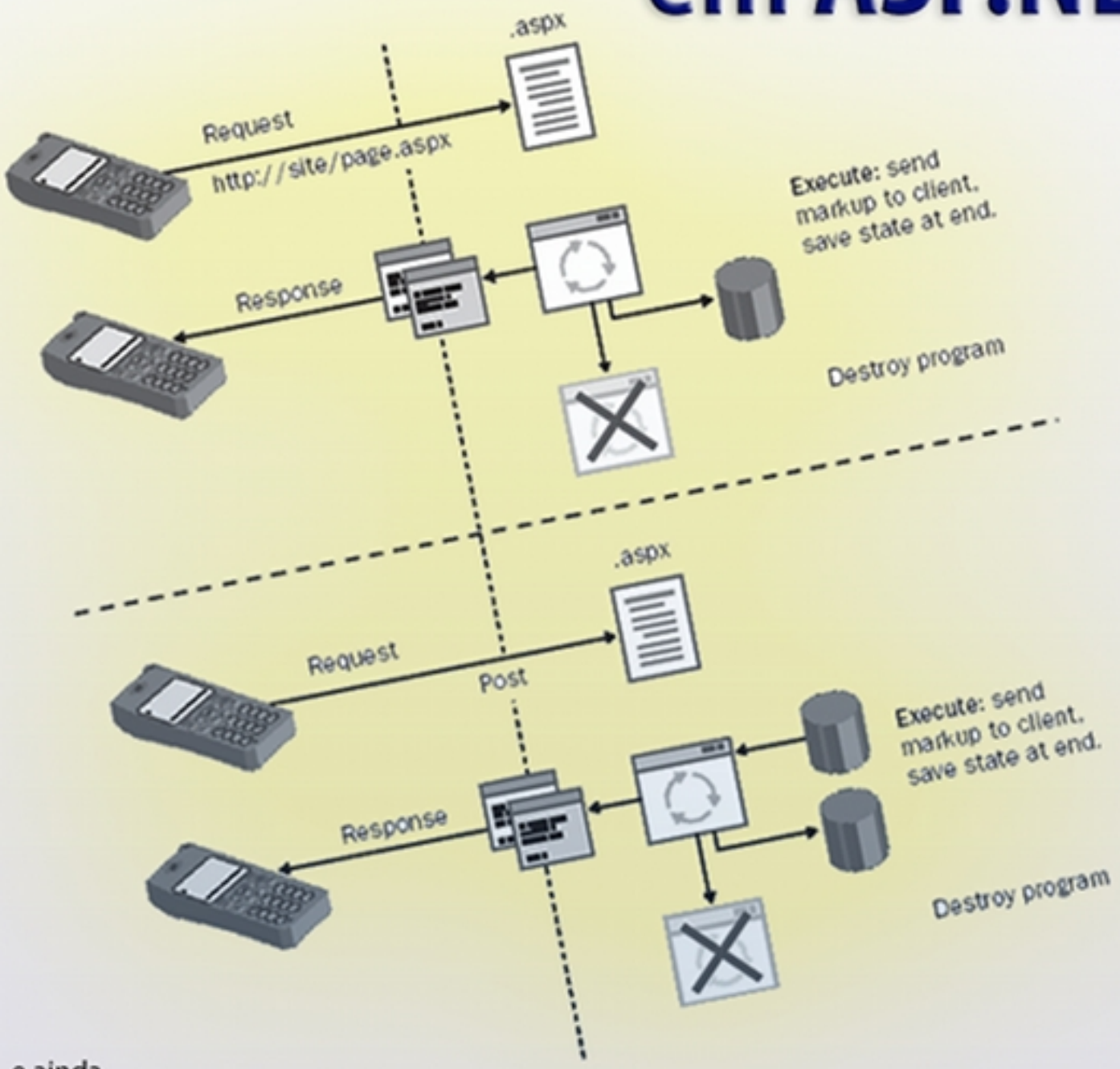
PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº15 - Agosto de 2008

www.portugal-a-programar.org

Estado de Visualização em ASP.NET



e ainda...

Programas Circulares em Haskell / Lucene / Internet Protocol v6

índice

equipa PROGRAMAR

coordenador
Miguel Pais

coordenador adjunto
Joel Ramos

editor
Pedro Abreu

redacção
Augusto Manzano
Ciro Cardoso
João Rodrigues
Miguel Bento
Miguel Alho
Vitor Tomaz

colaboradores
José Fontainhas
David Ferreira
Pedro Teixeira

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

O futuro (da pesquisa...)

Quando se fala em pesquisa decerto todos reconhecem o nome Google quase ao ponto de sinónimo, não apenas de pesquisa, mas mais importante que isso, de pesquisa com sucesso. Ao longo dos anos o Google tem sabido manter-se na ribalta do acesso ao conhecimento em todo o mundo, enveredando por outros projectos mas dando sempre a devida atenção à inovação (que os marcou desde o inicio) no seu produto principal. Menos conhecido, menos acedido e utilizado em terras lusas, mas igualmente popular a nível mundial encontra-se o Digg, representando o conceito de website social relativo a notícias. O conceito é trivial mas a ideia inovadora para a altura: notícias são submetidas por utilizadores sendo promovidas por votos positivos destes.

O que têm em comum estes dois gigantes do seu ramo? Bem, nos últimos dias (segunda semana de Julho) foi reportada peloTechCruch uma nova "feature" do Google que parece agregar o conceito do Digg às pesquisas de utilizadores, ou seja, cada um tem a possibilidade de enviar um resultado do Google para os confins da numeração de páginas, ou trazer outro para o topo. Além disso há a possibilidade de comentar os websites e os seus respectivos comentários, com um sistema de pontuação bastante semelhante ao do Digg. Com isto qualquer um poderá pôr o resultado que realmente lhe interessa como primeiro a ser retornado pelo Google para aquelas keywords, o que é algo que faz bastante sentido. A maneira como esta componente social influenciará num todo no futuro os resultados do Google será algo interessante de analisar.

Posto isto algo desde logo salta à vista: tudo o que antes recebíamos como passivo está a evoluir para algo activo. O tempo em que o utilizador não tinha nada a dizer na informação que recebia acabou. Já ninguém quer receber passivamente informação sem ter voto na matéria, os utilizadores querem a escolha e a decisão. E tudo isto faz sentido. Toda a Internet já percebeu isso, mas seria interessante que não só este meio virtual aderisse a este estilo.

A televisão começa a ser menosprezada pela Internet, onde cada um vê o que quer às horas que quer e a televisão não ganhará de novo a batalha se não se converter ao mesmo género. As mais recentes tentativas na área da televisão em Portugal são boas, mas ainda não representam o estado real em que os utilizadores se encontram. A verdadeira televisão do futuro pode mostrar ao utilizador sempre que ele queira um programa da semana passada, instantaneamente e sem a construção de enfadonhos planos de gravação agendada. A própria rádio usual já não faz sentido. Decerto utilizadores da Internet habituados a ouvirem apenas o que gostam em rádios de serviços como o Last.fm deverão já achar ridículo alguém se submeter a ouvir algo que pode muito bem não corresponder em nada aos seus gostos. Seria importante que muita gente olhasse para a web e começasse a aplicar o que lá vê às mais pequenas coisas da nossa sociedade, a nossa maneira de receber e lidar com a informação seria bem melhor.

Veremos o que o futuro nos reserva...

---

1) Referência: http://www.techcrunch.com/2008/07/16/is-this-the-future-of-search/
2) Uma nota para a mudança dos meses de publicação da revista, estes são agora os meses pares. Esperamos que a decisão não tenha prejudicado ninguém e que continuem a confiar neste projecto.

Wii é a nova líder de mercado nos EUA

A Wii já é a consola mais vendida nos Estados Unidos. Só em Junho foram comercializados 666 mil equipamentos, o que fixou o número total de dispositivos vendidos, desde Novembro de 2006, altura em que saiu para o mercado, nos 10,9 milhões de consolas.

Os resultados foram recolhidos pelo NPD Group que concluiu na análise ao mercado referindo que o equipamento da Nintendo já destronou a Xbox 360 da Microsoft, lançada um ano antes.

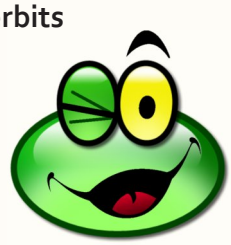
Como em todos os segmentos de mercado, também neste as opiniões divergem, com cada fabricante a defender o seu posicionamento no mercado. Todas afirmaram na Electronic Entertainment Expo (E3) que são as mais populares embora os números dêem a liderança das vendas à Nintendo.

A Wii foi concebida a pensar num tipo de divertimento diferente das consolas que já existiam no mercado. Introduziu uma nova forma de jogar graças aos comandos sem fios e jogos que apelam ao movimento do utilizador. A estratégia da Nintendo foi bem recebida pelo público de todas as idades, em todo o mundo, acabando por se revelar um caso de sucesso.

Os números não enganam e mostram o destaque que a consola alcançou no mercado norte-americano, onde as concorrentes PS3 e Xbox 360 venderam respectivamente 405,5 e 219,8 mil equipamentos.

As vendas totais no mesmo mercado subiram 53 por cento face a Junho do ano passado, colocando o valor da indústria nos 1,69 mil milhões de dólares. Já as vendas de jogos subiram 61 por cento, fixando-se agora nos 872,6 milhões de dólares.

SAPO Summerbits



Já arrancou o SAPO Summerbits, um programa promovido pela empresa do Grupo PT em parceria com a Associação Ensino Livre que tem como objectivo promover bolsas para estudantes que desenvolvam código para projectos de software livre.

Os incentivos destinam-se a alunos que tenham mais de 18 anos e vínculos a qualquer grau de ensino ou tipo de escola, desde que nacional. No final, as 10 ideias que mais se destaquem, quer sigam linhas de tecnologias existentes ou sejam completamente novas, serão financiadas em 2.500 euros.

A meta do SAPO Summerbits é atingir um patamar de referência entre as iniciativas académicas e junto das comunidades de software livre do país. Os interessados em participar no programa deverão inscrever-se até 19 de Julho deste ano na página oficial do evento.

Apple torna-se o terceiro maior fornecedor de computadores

A Apple tornou-se o terceiro maior fornecedor de computadores pessoais nos EUA, ultrapassando a Acer, de acordo com uma pesquisa da Gartner, que adiante um crescimento de 38,1% em vendas, no segundo trimestre de 2008.

O mercado de PC´s nos EUA apresentou um crescimento médio de 4,2%, no último trimestre, com a venda de 16,5 milhões de máquinas. Só a empresa de Steve Jobs comercializou 1,4 milhões de desktops naquele período, enquanto a Acer registou 1,33 milhões de vendas.

A Dell, que lidera o segmento, vendeu 2,5 milhões de unidades, seguida pela HP. Apesar do terceiro lugar, a Apple ainda tem uma participação tímida no mercado mundial de computadores, o que pode representar uma oportunidade de crescimento.

No Japão, por exemplo, a marca vendeu 130 mil máquinas no primeiro trimestre, de acordo com a IDC.

Estado de Visualização em ASP.NET

Antes de entrar em pormenores sobre estado de visualização é necessário compreender o que é um controlo e como este funciona em ASP.NET. De certa forma o ASP.NET é uma arquitectura baseada em controlos, já que uma página é um controlo e qualquer controlo pode conter controlos filhos.

A arquitectura é semelhante à arquitectura de janelas do Windows, onde o próprio ambiente de trabalho é uma janela, que pode conter janelas filhas. Cada janela é apresentada, apresentando primeiro o seu conteúdo e depois apresentando o conteúdo das janelas filhas. O mesmo se passa em ASP.NET em que cada controlo é apresentado, apresentando primeiro o seu conteúdo e depois apresentando o conteúdo dos seus filhos. A apresentação de uma janela em Windows envolve o desenho de pixeis no ecrã, enquanto que a apresentação de um controlo ASP.NET envolve a geração de HTML para preencher uma parte da resposta a um pedido HTTP.

Uma página serve como controlo raiz e tem três controlos filhos imediatos: um controlo literal para gerar o texto de início da página, um controlo do lado do servidor HtmlForm para representar o formulário e todos os seus controlos filho, e por fim outro controlo literal para gerar o texto de fim da página. Todos os controlos adicionados a uma página estarão dentro do formulário e portanto serão filhos do HtmlForm, em que a ordem pela qual estão definidos dentro desse controlo será a ordem pela qual serão apresentados. Cada um destes controlos tem o seu próprio estado de visualização. Todos os controlos que correspondem a elementos de formulário têm a sua manutenção de estado suportada através do envio automático do valor dos elementos quando o formulário é submetido (post back). Todos os outros terão que definir o seu mecanismo de persistência de estado e será sobre este assunto que iremos conversar.

Em ASP.NET, o estado de visualização entre post backs é mantido através de uma colecção de pares nome/valor acessíveis por qualquer controlo a partir da propriedade ViewState. Quase todo o estado do controlo, se não todo, fica guardado nesta colecção. Esta propriedade retorna uma instância do tipo System.Web.UI.StateBag, que é muito semelhante a uma tabela de hash mas tem a capacidade de registar alterações (fazer tracking), ou seja, permite que sempre que um valor seja alterado na colecção seja também marcado como "dirty".

A função de registo de alterações pode estar ligada ou desligada, mas uma vez ligada não pode ser desligada. De modo a activar a função use o método TrackViewState(). Se a função de tracking estiver ligada qualquer alteração a um objecto fará com que esse objecto fique marcado como "dirty". Podemos consultar se o objecto está marcado ou não através do método IsItemDirty(string chave) ou forçar a marcação através do método SetItemDirty(string chave). O leitor tenha em atenção um pormenor, após a activação de TrackViewState() qualquer alteração será marcada, mesmo que o objecto seja alterado para o mesmo estado, como por exemplo:

```
stateBag["nome"] = "valor";
stateBag.IsItemDirty("nome"); // falso
stateBag.TrackViewState();
stateBag["nome"] = "valor";
stateBag.IsItemDirty("nome"); //
verdadeiro
```

A colecção ViewState, como já foi referido, guarda pares nome/valor, os pares podem ser indexados por string e ter qualquer object como valor. Exemplo :

```
ViewState["ViewStateVariableName"] =
1;
```

Para ter acesso à variável guardada basta fazer a indexação com a chave e a respectiva conversão.

```
int number = (int)
ViewState["ViewStateVariable"];
```

Esta colecção também permite guardar os nossos próprios tipos quase tão facilmente como os tipos básicos. Para tal basta apenas o tipo ser serializável, ou seja, consegue-se converter uma instância desse tipo para uma sequência de bytes e posteriormente fazer a sua recuperação. Vai compreender a necessidade de o tipo ser serializável mais á frente neste artigo quando falarmos de um campo de input oculto chamado __VIEWSTATE.

```
[Serializable]
public class Pessoa
{
    public string _nome;
    public int _idade;

    public Pessoa(string nome, int
idade)
    {
        _nome = nome;
        _idade = idade;
    }
}
```

Como a classe Pessoa está marcada como serializável pode ser guardada em ViewState:

```
Pessoa p = new Pessoa("Vitor", 25);
ViewState["Cliente"] = p;
```

Lembre-se que terá de efectuar a respectiva conversão quando necessitar de obter o valor guardado.

```
Pessoa p = (Pessoa)
ViewState["Cliente"];
```

O protocolo HTTP é stateless, ou seja, cada pedido é executado independentemente e sem conhecimento de pedidos anteriores. Portanto cada pedido feito à arquitectura ASP.NET será servido por uma instância diferente do controlo e por isso não é possível guardar estado de visualização entre post backs em campos de instância. Devido a isso as propriedades em ASP.NET terão um aspecto diferente já que deverão usar a colecção ViewState para guardar qualquer valor:

```
public int ValorInteiro
{
    get { return
(int)ViewState["ValorInteiro"]; }
    set { ViewState["ValorInteiro"] =
value; }
}
```

Nos casos em que é necessário ter um valor por defeito teremos também que ter em conta que estamos a utilizar a colecção ViewState. Tal como uma tabela de hash, uma StateBag irá retornar null se a colecção não contém uma entrada com essa chave. Portanto, se o valor retornado for nulo é porque ainda não foi atribuído, então deve retornar o valor padrão, caso contrário, retornar o valor obtido.

```
public string Texto
{
    get { return ViewState["Texto"] ==
null ?
          "Valor por Defeito" :
(string)ViewState["Text"];
    }
    set { ViewState["Texto"] = value;
}
}
```

Repare que ao afectar com null uma propriedade definida com este padrão essa propriedade passará a retornar o valor por defeito e não null como acontece com as propriedades ditas normais. Uma alternativa poderá ser afectar a propriedade com String.Empty (um campo que representa um string vazia) em vez de null. De salientar ainda que cada controlo pode aceder á sua colecção ViewState em qualquer momento e por qualquer razão, não apenas através de propriedades.

Deve-se utilizar esta colecção quando as propriedades reflectem directamente tipos primitivos. No caso de controlos que têm pretendem manter um estado mais complexo, usando tipos próprios poderá ser mais complicado utilizar a colecção ViewState. Como alternativa podemos sobrepor dois métodos virtuais definidos na classe base Control sendo eles o SaveViewState() e o LoadViewState(). Estes métodos permitem escrever e ler manualmente o estado do controlo a partir da stream ViewState. Para um objecto poder ser guardado nessa stream tem que ser serializável. O método SaveViewState(), como o nome indica, permite guardar o estado de visualização. Note que este método também é responsável por chamar o método da base e guardar, no objecto a retornar, o resultado dessa chamada.

```
protected override object
SaveViewState()
{
    ArrayList OsMeusDados = new
ArrayList();
    object[] vState = new object[2];
    vState[0] = base.SaveViewState();
    vState[1] = OsMeusDados;
    return vState;
}
```

O método LoadViewState() permite recuperar o estado de visualização. Note que este método também é responsável por chamar o método da base, passando ao mesmo os dados que lhe pertencem.

```
protected override void
LoadViewState(object savedState)
{
     if (savedState != null)
     {
          ArrayList OsMeusDados;
          // Obter o array de objectos
guardados em SaveViewState
          object[] vState =
(object[])savedState;

          if (vState[0] != null)

base.LoadViewState(vState[0]);
          if (vState[1] != null)
               OsMeusDados =
(ArrayList)vState[1];
     }
}
```

O leitor certamente reconhece um padrão de recursividade na chamada a estes métodos e é desta forma que a arquitectura ASP.NET constrói uma estrutura de dados com todo o conteúdo de estado de visualização. Como já foi referenciado, o protocolo HTTP "não tem memória". Isso significa que o estado da página terá que ser guardado e posteriormente reposto no pedido seguinte. Os pares nome/valor colocados na colecção ViewState antes da apresentação da página são armazenados num campo de input oculto, __VIEWSTATE, e quando a página é de novo acedida através de um pedido de POST, o conteúdo do campo __VIEWSTATE é analisado e usado para reconstituir a colecção ViewState.

Falta-nos agora perceber como é mantido o estado dos controlos de formulário. Como já foi dito anteriormente, todos os controlos que correspondem a elementos de formulário têm a sua manutenção de estado suportada através do envio automático do valor dos elementos quando o formulário é submetido (post back). Esta manutenção de estado é realizada através da implementação da interface IPostBackDataHandler.

```
public interface IPostBackDataHandler
{
 bool LoadPostBackData(string
postDataKey, NameValueCollection
postCollection);
 void RaisePostDataChangedEvent();
}
```

Neste caso a arquitectura faz uma pesquisa no conteúdo do POST de modo a determinar se existem entradas que

correspondem a controlos que implementam IPostBackDataHandler e de seguida é invocado o método LoadPostData para todos esses controlos. A string postDataKey, passado como argumento, contém o identificador único associado ao controlo, que pode ser usado para indexar sobre postCollection para localizar o valor corrente do controlo dentro da colecção como se pode verificar no seguinte exemplo:

```
public virtual bool
LoadPostData(string postDataKey,
NameValueCollection postCollection) {
    string valorAntigo = Text;
    string valorNoPOST =
postCollection[postDataKey];

    if
(!valorAntigo.Equals(valorNoPOST)){
          Text = valorNoPOST;
          return true;
    }
    return false;
}
```

O resultado deste método deve ser true se mudou o valor do estado do controlo, caso contrário, o método deve devolver false. Para todos os controlos que retornam true neste método é chamado o RaisePostDataChangedEvent de cada um desses controlos de modo a poderem ser desencadeados eventos de alteração de estado. O leitor repare que o carregamento de dados de POST ocorre em duas fases, o método LoadPostData ocorre antes do método de Load da página e o método de RaisePostDataChangedEvent ocorre depois do método de Load. Isto permite que na altura em que as notificações são geradas todos os controlos tenham o seu estado reposto.

Note ainda que, para além dos eventos de alteração de estado (eventos changed), como por exemplo alterações do texto de uma caixa de texto ou a alteração do índice seleccionado de uma DropDownList, também os eventos de reacção utilizam este mecanismo. Por exemplo, o evento de click de um botão é detectado e lançado utilizando esta interface. Como se deve ter apercebido, a manutenção de estado de controlos de formulário não usa o campo oculto __VIEWSTATE em nenhuma das suas fases e é

completamente independente deste. É por essa razão que, mesmo que o ViewState não esteja activo, todos os controlos de formulário têm o seu estado automaticamente reposto.

No entanto, na fase LoadPostData, é necessário saber o valor antigo de modo a poder compará-lo com o que vem no POST. Uma das formas é usar o ViewState para previamente guardar esse valor, que depois será usado na comparação com o valor recebido no POST. Note que, apesar da sua ligação próxima, são funcionalidades independentes.

Voltando ao ViewState, a estrutura de dados com todo o conteúdo de estado de visualização não é directamente convertida para string e armazenada no campo __VIEWSTATE. Lembra-se de falarmos que uma StateBag permitia fazer tracking? Ora bem, apenas são salvos no campo oculto as entradas da StateBag que estão marcadas como "dirty", ou seja, apenas as entradas em que os seus estados são diferentes do estado estático ou estado por defeito. Não faz sentido guardar estado que será reposto automaticamente assim que exista um novo pedido e seja criada uma nova instância do controlo para atender esse pedido. Neste momento o leitor poderá estar a perguntar-se "então e os controlos definidos declarativamente?" Podemos definir declarativamente um controlo, como por exemplo:

```
<asp:Label ID="Label1" runat="server"
Text="Label">
```

No momento em que a arquitectura ASP.NET faz parse do formulário, ao encontrar o atributo runat="server" cria uma instância desse controlo e tenta corresponder todos os atributos definidos a propriedades publicas do controlo. No exemplo anterior olhando para o atributo Text percebemos que a arquitectura vai afectar a propriedade pública Text com o valor "Label". Sabendo nós que quase todo o estado do controlo, se não todo, fica guardado em ViewState poderíamos ser levados a pensar que estes dados seriam considerados "dirty" visto que estamos a mudar o seu estado. Tal não acontece porque o TrackViewState() só é chamado no evento Init do ciclo de vida de uma página e todas estas afectações são feitas antes desse momento, ou seja, não serão consideradas "dirty".

Como referido, quando o ocorre a fase de Init é chamado o TrackViewState(). Após o TrackViewState() ser chamado para todas as StateBags é chamado o LoadViewState(), que faz com que todo o estado dinâmico de visualização guardado seja reposto e, mais importante ainda, visto que neste momento o tracking já está activo todo o estado dinâmico carregado será considerado "dirty". Esta sequência fará com que o estado dinâmico de visualização seja reposto e fique novamente persistente para futuros post backs, ou seja, faz com que seja novamente seriado para __VIEWSTATE. Através deste artigo espero que o leitor fique com uma ideia mais clara de como manter estado nos seus controlos tendo em atenção que quase todo, se não todo, o estado de visualização de um controlo é guardado em ViewState, mas apenas é persistido entre post backs o estado dinâmico. O estado por omissão que é criado quando é criada uma instância desse controlo (estado estático) não é persistido.

Bibliografia

Essential ASP.NET with Examples in C# - Fritz Onion - Addison Wesley

Understanding ASP.NET View State - http://msdn.microsoft.com/en-us/library/ms972976.aspx

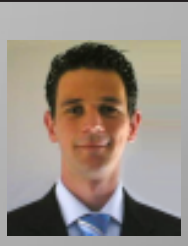
ViewState in ASP.NET 2.0 - http://www.beansoftware.com/ASP.NET-Tutorials/ViewState-In-ASP.NET.aspx

TRULY Understanding ViewState - http://weblogs.asp.net/infinitiesloop/archive/2006/08/03/Truly-Understanding-Viewstate.aspx

IPostBackDataHandler Interface - http://msdn.microsoft.com/en-us/library/system.web.ui.ipostbackdatahandler.aspx

StateBag Class - http://msdn.microsoft.com/en-us/library/system.web.ui.statebag.aspx

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

Lucene: programar um motor de busca

O que é o Lucene

O Lucene talvez dever-se-ia tratar por "a" Lucene, uma vez que se trata de uma biblioteca de recuperação de informação textual (do inglês "information retrieval") de código aberto e criada por Doug Cutting. Originalmente, foi escrita em Java, mas foi rapidamente adaptada a outras linguagens de programação, nomeadamente Python (Pylucene), Perl (Plucene), C# (Lucene.net), C++ (CLucene), e Ruby (Ferret). Contudo, estas adaptações estão normalmente ligeiramente atrasadas no que toca à versão original em Java, actualmente mantida e alojada pela Apache Sofware Foundation.

Simples de aprender a usar, mas poderosa nas mãos de um programador experiente, esta biblioteca suporta desde índices estáticos com um campo, até múltiplos índices em paralelo, com centenas de campos e milhares de acessos simultâneos. É ideal para todo o tipo de projectos, desde o simples website com a "search box" até um grande motor de busca sobre, por exemplo, a colecção de PDFs que se tem no disco rígido.

Onde é usado o Lucene

Provavelmente desconhecido para o leitor, o certo é que já passou os dedos por esta útil ferramenta. Ao fim de contas, quem nunca pesquisou no site Sourceforge, ou na CNET, ou mesmo até na Wikipedia? Para os utilizadores de Ubuntu, quem nunca tirou partido do software de indexação recentemente incorporado na distribuição: o Beagle. Este, não é mais que suportado pela Lucene.Net, a implementação em C# desta biblioteca. Assim, a utilização desta biblioteca por estas entidades (re)afirma por si só a qualidade e a utilidade da mesma.

Como funciona o Lucene

O modo de funcionamento do Lucene é trivial de se perceber, mas mais complexo de se manusear na perfeição. Para começar, como já se indicou, há uma palavra chave em todo o processo: índice. O índice não passa de uma estrutura altamente ordenada de texto, ou de pedaços de texto (tokens), que torna a pesquisa muito mais eficiente. Porém, falha na capacidade de compressão de dados. Um índice de 6 milhões de abstracts científicos atinge facilmente os 1.7 GB, no caso de armazenar apenas tokens, ou os 6.6 GB caso guarde o texto. Esta diferença de tamanho vai, obviamente, afectar a performance. Daí, uma vantagem para a pesquisa, é ter o índice fragmentado em vários "sub-índices", de modo a cada um ter menos volume de informação.

O modo como o texto é tratado pelo Lucene, também é alvo de atenção. Ao lermos texto para o lucene, ele será primeiro submetido a um processo chamado analisador (analyzer), que pode ter várias formas: Standard, Simple, Stop, regional, etc. Sem entrar em grandes explicações, cada analisador trata o texto de maneira diferente. Uns ignoram a pontuação, outros são específicos para certas línguas (como a nossa com o til e os acentos), outros ignoram palavras comuns, baseando-se numa lista de "stop-words", etc.

De seguida, tem que se criar um índice, que pode ser, na maior parte das vezes, ou baseado no disco rígido (acesso mais lento), ou na memória RAM (temporário). Uma estratégia para aumentar a velocidade de construção do índice passa por criar o índice na RAM e no fim, "despejá-lo" para o disco rígido. Na pesquisa, o mesmo índice pode voltar a ser lido para a RAM, potenciando (muito significativamente) a performance. Tendo o índice criado, há que criar um escritor de índices, de modo a populá-lo com os nossos textos. De entre as várias opções a passar ao escritor, estão o analisador em causa e o índice previamente criado.

A inserção de documentos no índice faz-se por último, adicionando documento a documento, criando os campos que forem precisos (por exemplo, a data de criação de um ficheiro, e o seu conteúdo, ou o título de um livro, a sua editora, autor, ano de lançamento, e preço). Quantos mais campos, mais lento ficará o índice, mas mais específica poderá ser a nossa pesquisa. Tomando o exemplo da livraria, podemos pesquisar apenas por editora, ou por ano de lançamento, ou mesmo por preço. A maneira de guardar a informação também varia. Podemos ter a necessidade de guardar alguns campos para mais tarde devolver ao utilizador que pesquise, mas outros campos podem ser fúteis e são apenas guardados "tokens" dos mesmos. Estes tokens são fragmentos únicos do texto e não estão disponíveis para futuras recuperações. Exemplificando, um resumo de um artigo científico está dividido em título e resumo. Podemos pesquisar sobre os resumos por uma determinada palavra, sendo devolvidos os títulos que correspondem à nossa pesquisa.

No fim do índice criado, pode-se, opcionalmente, mas sempre recomendado, passar por um processo de optimização que vai aumentar a performance da pesquisa

no índice recém criado. Este processo vai reduzir os "segmentos" do índice a apenas um, facilitando assim a tarefa de abrir o índice, em vez de múltiplos pequenos ficheiros. Explicações sobre a formação destes segmentos estão para além do âmbito deste artigo, bastando por agora dizer apenas que são resultado da construção do índice, podendo ser imaginados como sub-índices que degradam significativamente a performance devido ao seu diminuto tamanho e conteúdo.

Após se ter um índice, resta programar o pesquisador. O pesquisador vai usar um pesquisador de índices (IndexSearcher), que vai precisar da directoria do índice (tal como o IndexWriter), e de um parser da pesquisa (QueryParser), que vai depender do nome do campo onde pesquisar e de um analisador (igual ao usado na construção do índice). Recuperam-se depois os resultados, que estão num iterador, e depois vai-se retirar de cada objecto-resultado o campo pretendido (voltando ao exemplo do artigo científico, o título do artigo). Os resultados estão ordenados por ordem de relevância. O próprio Lucene tem um mecanismo de atribuição de relevância, baseado no TF-IDF, que dá mais peso a documentos com maior frequência do termo, e a termos pouco comuns no geral.

Desta forma, é simples construir um motor de busca, desde que se tenha maneira de ler os ficheiros (parser de PDFs para PDFs por exemplo). A documentação oficial do projecto é bastante completa e simples de seguir, fornecendo um caminho perfeito para tirar as dúvidas que possam surgir. Para além disso, há bastantes mailing-lists que ajudam os novatos. A cereja no topo do bolo, é a inclusão de vários scripts exemplo na distribuição do Lucene, e do PyLucene por exemplo. Estes exemplos são um óptimo ponto de partida para o Lucene.

PyLucene: usando o Lucene em Python

A biblioteca PyLucene está disponível desde, pelo menos, 2004. Criada e mantida por uma única pessoa, Andi Vadja, veio não só trazer aos utilizadores de Python o poder do Lucene, como também o fez de uma forma extremamente simplista e fiel à biblioteca original. De início, havia duas implementações: GCJ e JCC. Porém, desde a versão 2.3.0 do Lucene, a implementação em GCJ foi deixada de parte. As principais diferenças entre as implementações traduziam-se na velocidade de pesquisa, onde a GCJ liderava por larga margem, velocidade de indexação, onde era a JCC a levar a melhor, e noutras características como o tamanho dos índices suportados (maior na JCC) e maior estabilidade (JCC). Contudo, o processo de instalação tornou-se mais complicado, daí que apesar das fontes serem cedidas para compilação pelo utilizador, há quem disponibilize versões binárias para os mais novatos.

Exemplos de código

Tal como qualquer biblioteca em Python, chama-se o Lucene da seguinte forma:

```
import lucene
```

Porém, isto não chega. Com a implementação JCC, trabalha-se dentro de uma máquina virtual de Java, sendo portanto necessário criá-la, podendo definir-se os seus parâmetros (maxheap - limite máximo de memoria RAM a ser usado, initialheap - memória RAM a ser usada inicialmente, etc):

```
lucene.initVM(lucene.CLASSPATH)
lucene.initVM(lucene.CLASSPATH,
maxheap=800m)
lucene.initVM(lucene.CLASSPATH,
maxheap=2g)
lucene.initVM(lucene.CLASSPATH,
initialheap=800m)
lucene.initVM(lucene.CLASSPATH,
initialheap=200m, maxheap=800m)
```

O resultado que se deve obter, numa consola de Python, é semelhante a:

```
>>> import lucene
>>> lucene.initVM(lucene.CLASSPATH)
<jcc.JCCEnv object at 0x8192240>
>>>
```

Visto como iniciar a máquina virtual, pode-se agora tirar partido de todo o potencial da biblioteca.

Vamos começar por criar um indexador, ou seja, um script que crie um indíce a partir de uma pasta contendo ficheiros de texto, no formato txt, por uma questão de simplicidade. É de ter em atenção que, ao contrário dos ficheiros txt, os ficheiros como por exemplo os PDF precisam de um parser para os ler, uma vez que ao Lucene é alimentado o conteúdo do ficheiro, e não o ficheiro em si. A lógica por detrás do funcionamento do Lucene já foi explicada anteriormente.

```
import lucene, os

lucene.initVM(lucene.CLASSPATH) #
Iniciar a máquina virtual
```

Indexador.py

```
indexPath = "Texto/Index" # indicar o
caminho da pasta com os ficheiros

analyzer = lucene.StandardAnalyzer() #
criar o analisador

indexDir =
lucene.FSDirectory.getDirectory(indexPa
th) # criar o índice

indexWriter =
lucene.IndexWriter(indexDir, analyzer)
# criar o escritor do índice

for root, dirs, files in
os.walk('Texto'): # percorremos os
ficheiros da pasta
    for eachfile in files:
        fileinfo = eachfile.split('.')
        filename = '
'.join(fileinfo[:-1])
        if fileinfo[-1] == 'txt': #
Para assegurar que só lemos ficheiros
de texto simples

            document =
lucene.Document() # Criamos um
documento pronto a ser inserido no
índice


document.add(lucene.Field(("filename"),
 filename, lucene.Field.Store.YES,
lucene.Field.Index.UN_TOKENIZED)) #
Adicionamos o campo do nome do
ficheiro, que vai ser posteriormente
devolvido em pesquisas ao utilizador

            contents =
open(os.path.join(root, eachfile),
'r').read()


document.add(lucene.Field(("contents"),
 contents, lucene.Field.Store.NO,
lucene.Field.Index.TOKENIZED)) # Campo
com o conteudo do ficheiro, tokenizado
e não armazenado.


indexWriter.addDocument(document) #
Adiciona o documento ao indice

indexWriter.close() # Fechamos o
escritor
```

De seguida, podemos escrever em meia dúzia de linhas um pesquisador para comprovarmos que o nosso índice foi bem construido. No meu exemplo, indexei apenas 2 ficheiros, cada um com a seguinte frase:

hello.txt – "Hello World?"

goodbye.txt – "Python is a wonderful language!"

Pesquisador.py

```
import lucene

lucene.initVM(lucene.CLASSPATH)

directory =
lucene.FSDirectory.getDirectory('Texto/
Index') # apontar para a directoria do
indice

searcher =
lucene.IndexSearcher(directory) #
criar o pesquisador
analyzer = lucene.StandardAnalyzer() #
criar o analisador

term = raw_input('Escreva o termo a
pesquisar: ')

query = lucene.QueryParser("contents",
analyzer).parse(term) # Passamos o
termo a procurar para o QueryParser,
de modo a ser analisado, apontando
tambem que campo do indice deve ser
procurado.

hits = searcher.search(query) #
Recuperamos os resultados

if len(hits)>0:
    results = [hits[x] for x in
range(len(hits))] # Criamos uma lista
com os documentos dos resultados
    filenames = [doc.get("filename")
for doc in results] # recuperamos o
campo filename de cada resultado
    scores = [hits.score(x) for x in
range(len(hits))] # recuperamos a
pontuaÃ§Ã£o que cada documento teve na
pesquisa

    for i in range(len(filenames)):
        print "%s, %s" %(filenames[i],
scores[i]) # Devolvemos os resultados
else:
    print "Sem resultados!"
```

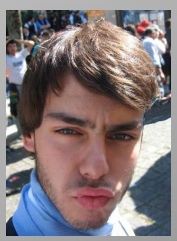
O resultado deste script será:

```
joao@jUbuntu:~/Desktop$ python
pesquisador.py
Escreva o termo a pesquisar: python
goodbye, 0.5

joao@jUbuntu:~/Desktop$ python
pesquisador.py
Escreva o termo a pesquisar: hello
hello, 0.625

joao@jUbuntu:~/Desktop$ python
pesquisador.py
Escreva o termo a pesquisar: portugal
Sem resultados!
```

Conclusão

A biblioteca Lucene permite ao programador criar rapidamente um motor de busca que pode ser usado como aplicação por si só, ou para ser incorporado noutra, ou num website. Apesar de drasticamente simples, pode tomar contornos bastante complexos à medida que se vagueia pelas suas funcionalidades, permitindo uma personalização e uma costumização fenomenal, ideal para perfeccionistas que querem lidar com todos os pormenores, ou para aqueles que precisam de um sistema com um determinado conjunto de características. Para além disso, é grátis, de código livre, de maneira a que para ser usada não precisamos de dispender nenhum tostão e podemos sempre dar uma olhadela ao código por detrás do seu funcionamento para ganhar um maior conhecimento.

SOBRE O AUTOR



Estranhamente, João Rodrigues não vem das hostes dos informáticos tendo tido o primeiro contacto com a programação no 3º ano do curso. Um descontraído finalista da Licenciatura em Bioquímica em Coimbra, com particular interesse pelas áreas da Bioinformática e da Proteómica, entrou para a Revista PROGRAMAR e com o objectivo de mostrar que há mais na informática para além de bits e bytes.

João Rodrigues

Programas Circulares

Introdução

Introduzido originalmente como demonstração do poder da "lazy evaluation" nas linguagens funcionais nos anos 80, os programas circulares são considerados como uma técnica poderosa, concisa, eficiente e elegante para resolver um particular tipo de problema. Se um algoritmo tiver de percorrer uma estrutura de dados várias vezes, numa linguagem de avaliação atrasada, o mesmo poderá ser expresso com uma única travessia.

Como o nome indica, programas circulares caracterizam-se por a sua definição ter uma aparência circular, isto é, um dos argumentos de uma função depende de um dos argumentos que a mesma função retorna "x = f(x)". Numa linguagem com "strict evaluation" nunca irá terminar, enquanto que numa linguagem "lazy" às vezes conseguirá terminar. A avaliação atrasada irá sempre obter a ordem correcta de processamento, se essa ordem de facto existir. Assim, a função f pela sua definição será virtualmente circular, já que no momento da sua avaliação irá ser decomposta em não circular.

No entanto, também é sabido que os programas circulares são difíceis de entender e escrever. Facilmente o programador define um programa realmente circular, ou seja, que nunca termine. Mas deixemos de conceitos teóricos e passemos para algo mais prático. Por uma questão de hábito e facilidade de demonstração, a linguagem usada irá ser Haskell.

RepMin

Comecemos pelo exemplo clássico repmin, o problema é o seguinte.

Imaginemos que temos uma árvore binária de folhas, queremos obter uma nova árvore morfologicamente idêntica à original, mas agora todas as suas folhas terão que ter o valor mínimo da árvore original. (o valor mínimo que a árvore criada terá, tem como referência aos valores das folhas iniciais).

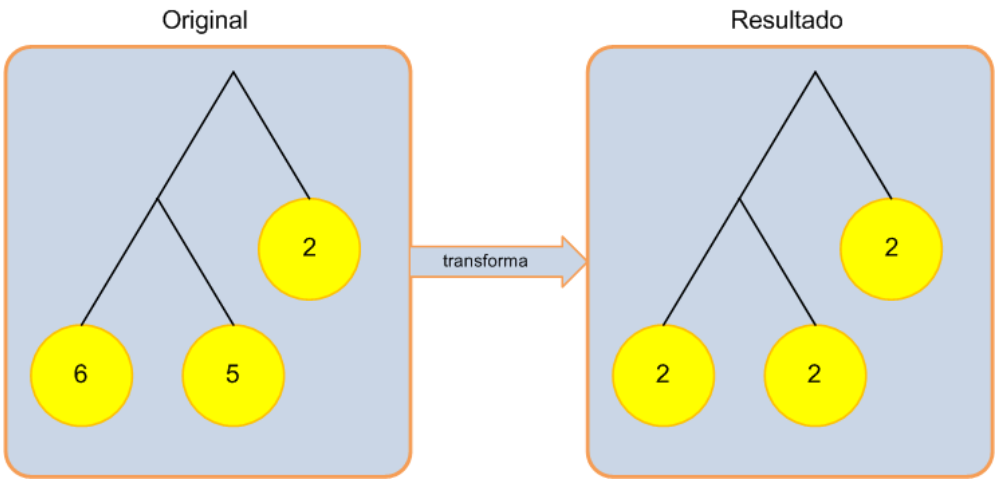
```
data Arvore = Folha Int          -- Uma folha terá um valor que será um inteiro
            | Nodo Arvore Arvore -- Um nodo terá duas sub-árvores
```

Agora para resolver o problema iremos criar a função transforma que irá pegar numa árvore e devolver uma nova árvore.

```
transforma :: Arvore -> Arvore
transforma arv = replicaArv arv (minArv arv)
```

Como podemos constatar a função transforma irá dar uso a duas funções.

A função minArv que irá calcular o valor mínimo de uma árvore:



```
minArv :: Arvore -> Int
minArv (Folha x) = x
minArv (Nodo e d) = min (minArv e)
(minArv d)
```

A função replicaArv que irá replicar uma dada árvore com um valor mínimo.

```
replicaArv :: Arvore -> Int -> Arvore
replicaArv (Folha _) m = Folha m
replicaArv (Nodo e d) m = Nodo
(replicaArv e m) (replicaArv d m)
```

Como podemos ver, a árvore irá ser percorrida duas vezes. Uma primeira vez para obter o valor mínimo da árvore e uma segunda vez para replicar a árvore com o valor mínimo.

Resolução Circular (Uma travessia)

Vejamos agora como podemos transformar o código de cima ficando um programa circular. A forma mais simples de efectuar essa transformação é recorrendo à fusão das duas funções através do uso de tuplos. Portanto começamos a definir a função repmin como o par das funções replicaArv e minArv. A nossa função repmin irá receber uma árvore e um hipotético valor mínimo.

```
repmin :: (Arvore, Int) -> (Arvore,
Int)
repmin (a, m) = (replicaArv a m,
minArv a)
```

Vamos agora tentar fundir as duas funções, simplificando o código. Ora uma árvore pode ser folhas ou nodos, portanto temos que ter os dois casos. Comecemos pelas folhas por ser mais simples. Então caso seja uma folha, a linha de cima fica da seguinte forma:

```
repmin ((Folha x), m) = (replicaArv
(Folha x) m, minArv (Folha x))
```

Usando a definição de replicaArv e minArv obtemos um código equivalente, simplificando-o ainda mais.

```
repmin ((Folha x), m) = (Folha m, x)
```

Simples.
Vejamos agora no caso de ser um nodo.

```
repmin ((Nodo e d), m) = (replicaArv
(Nodo e d) m, minArv (Nodo e d))
```

Mais uma vez, usamos as definições das funções replicaArv e minArv.

```
repmin ((Nodo e d), m) = (Nodo
(replicaArv e m) (replicaArv d m), min
(minArv e) (minArv d))
```

Agora o passo seguinte poderá não parecer tão intuitivo. Mas visto que tanto a função replicaArv como a função minArv seguem o mesmo padrão de recursividade, podemos usar a própria definição de repmin para simplificar ainda mais o código.

```
repmin ((Nodo e d), m) = (Nodo e' d',
min m1 m2)
  where (e', m1) = repmin (e, m)
        (d', m2) = repmin (d, m)
```

Bem, se for um Nodo também não há muito mais a fazer. Recapitulemos como temos definido a nossa função repmin:

```
repmin :: (Arvore, Int) -> (Arvore,
Int)
repmin ((Folha x), m) = (Folha m, x)
repmin ((Nodo e d), m) = (Nodo e' d',
min m1 m2)
  where (e', m1) = repmin (e, m)
        (d', m2) = repmin (d, m)
```

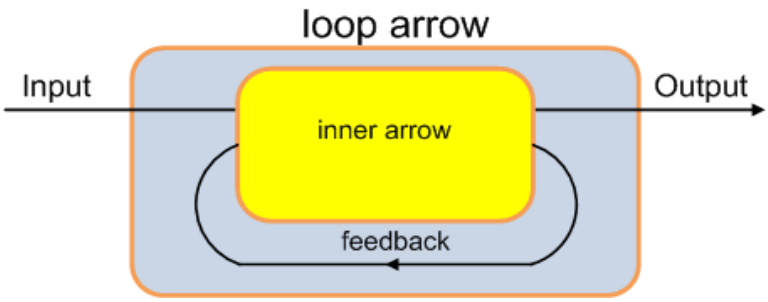
Bom, dado uma árvore e um hipotético valor mínimo, replica essa árvore com esse valor hipotético e ao mesmo tempo calcula o valor mínimo da árvore, tudo em apenas uma travessia. Ainda melhor era conseguirmos dizer que aquele valor hipotético era o valor mínimo calculado por nós. E é aqui que entra a função circular, redefinamos então a função transforma para nos permitir tal proeza.

```
transforma :: Arvore -> Arvore
transforma arv = arv'
  where (arv', m) = repmin (arv, m)
```

E está feito. De salientar o uso do símbolo m nos dois lado da função repmin, ou seja, é ao mesmo tempo o argumento e o resultado da função. No momento de execução da função transforma, o Haskell devido a ser lazy como que por magia irá transformar a árvore com um valor que só terá acesso no futuro, é como se fosse possível viajar no tempo.

Arrows

O exemplo foi feito de forma a demonstrar totalmente o processo de converter o código para a versão circular. Na verdade, o último passo é desnecessário. Haskell possui Arrows, um conceito similar às populares Monads mas mais generalista.
Ora com arrows uma pessoa consegue com bastante facilidade descrever e conjugar computações. E como já devem estar à espera, existe uma arrow já pré-definida na biblioteca para o nosso caso. Essa arrow tem de nome loop e tem o seguinte esquema.



A função loop é em si uma arrow que possui internamente uma outra arrow que dado um input irá devolver um ouput com um auxílio de um feedback. Como podem reparar este feedback irá ser a nossa referencia circular.

Do ponto de vista do Haskell, a função loop tem a seguinte assinatura.[1)]

```
loop :: (Arrow a) => a (b, d) (c, d) -
> a b c
```

Analisando a assinatura, podemos ver que a arrow interna tem de ser do tipo 'a (b, d) (c, d)'. Vamos ter o input do tipo 'b' , o output do tipo 'c' e o feedback será do tipo 'd'. No fim vamos ter como resultado a arrow 'a b c', ou seja, uma computação que dados aas devolve bbs. Vejamos como podemos usar isto no nosso caso.

1) Não é totalmente verdade, de facto o loop é um ArrowLoop que é uma extensão da classe Arrow, mas simplifica o processo de explicação.

Já que estamos a trabalhar apenas com funções, para começar podemos dizer que a nossa Arrow a é uma função. Então loop fica:

```
loop :: ( (b, d) -> (c, d) ) -> (b ->
c)
```

Visto que o nosso input e output irão ser árvores, portanto o nosso 'b' e 'c' irão ser do tipo Arvore. O feedback irá ser o nosso hipotético valor minimo, ou seja, o 'd' será um Int. Fazendo estas substituições na assinatura do loop ficamos com:

```
loop :: ( (Arvore, Int) -> (Arvore,
Int) ) -> (Arvore -> Arvore)
```

Já está mais agradável. A função loop recebe como argumento uma função e devolve outra função. Se vermos com atenção, vimos que a assinatura desta função (a de argumento) é exactamente a mesma que a da nossa função repmin. De facto, para pormos a função repmin como circular basta passar para a função loop como argumento. Visto isto, podemos redefinir a nossa função transforma usando a biblioteca Control.Arrow.

```
transforma :: Arvore -> Arvore
transforma = loop repmin
```

Média

Agora que já demonstrámos um método simples de transformar um programa com múltiplas travessias em apenas uma, vamos ver outro exemplo clássico, pois este método simples está longe de ser perfeito, e nem sempre produz resultados ideais. Mas que se estivermos atentos facilmente contornamos o problema. Temos então o seguinte problema.

Dada uma lista de valores, queremos calcular a média da lista e incrementar cada valor da lista pela média obtida.

Assim de imediato vemos que temos três travessias na lista. Uma travessia para calcular o número de elementos na lista, uma segunda travessia para somar os elementos da lista e uma terceira travessia para incrementar cada valor da lista pela média. De salientar que a primeira travessia e a segunda travessia não dependem uma da outra e portanto é muito simples fazer a sua fusão e é apenas necessário uma travessia. Mas não é esse o ponto em questão, e é para reforçar a ideia, que para cada posição da lista temos que efectuar três operações.

Bem, não vou estar agora a fazer passo a passo, mas à primeira tentativa o mais provável era obtermos algo do género.

```
transforma :: [Double] -> [Double]
transforma = loop incrMedia

incrMedia :: ([Double], (Double,
Double)) -> ([Double], (Double,
Double))
incrMedia ([], _) = ([], (0,0))
incrMedia ((x:xs), (soma, numElems)) =
( (x + media) : xs', (soma' + x,
numElems' + 1))
  where (xs', (soma', numElems')) =
incrMedia (xs, (soma, numElems))
        media = soma / numElems
```

O problema deste código é que se formos executar com uma lista nunca irá terminar. O problema está no cálculo da média em todo os passos. Podemos solucionar este problema separando o cálculo da média da travessia da lista, isto é, já termos o nosso feedback (a média) já calculada. A versão correcta seria então:

```
transforma :: [Double] -> [Double]
transforma = loop incrMedia

incrMedia :: ([Double], Double) ->
([Double], Double)
incrMedia (lista, media) = (lista',
soma / total)
  where (lista', (soma, total)) =
incrMedia' (lista, media)

incrMedia' :: ([Double], Double) ->
([Double], (Double, Double))
incrMedia' ([], _) = ([], (0, 0))
incrMedia' ((x:xs), media) = ((x +
media) : xs', (soma + x, total + 1))
  where (xs', (soma, total)) =
incrMedia' (xs, media)
```

Como podemos verificar as duas soluções são muito parecidas, mas na última versão inserimos uma função extra no meio para separar o cálculo da média da travessia da lista. É necessário, visto que se tivermos um valor hipotético de média, a assinatura da função não encaixa, já que a cada passo o feedback irá ser o somatório e a contagem dos elementos da lista. Daí termos uma função extra para servir de adaptador, para o nosso feedback ter o mesmo tipo, tanto à entrada como à saída.

Já que se está a usar Arrows pode-se sempre pôr o código ainda mais conciso, apesar de se tornar um pouco menos legível. Fica ao gosto de cada um a forma que prefere.

```
transforma :: [Double] -> [Double]
transforma = loop (incrMedia >>> id
*** uncurry (/))

incrMedia :: ([Double], Double) ->
([Double], (Double, Double))
incrMedia ([], _) = ([], (0, 0))
incrMedia ((x:xs), media) = incrMedia
                          >>> ((x +
media):) *** ((+x) *** (+1))
                          $ (xs,
media)
```

Conclusão

Este artigo tinha como objectivo dar a conhecer o conceito de programas circulares e espero que tenha tido sucesso. Foi feita apenas uma pequena introdução do conceito, com o auxílio de um método muito simples de transformação de programas. O método aqui indicado tem várias falhas.

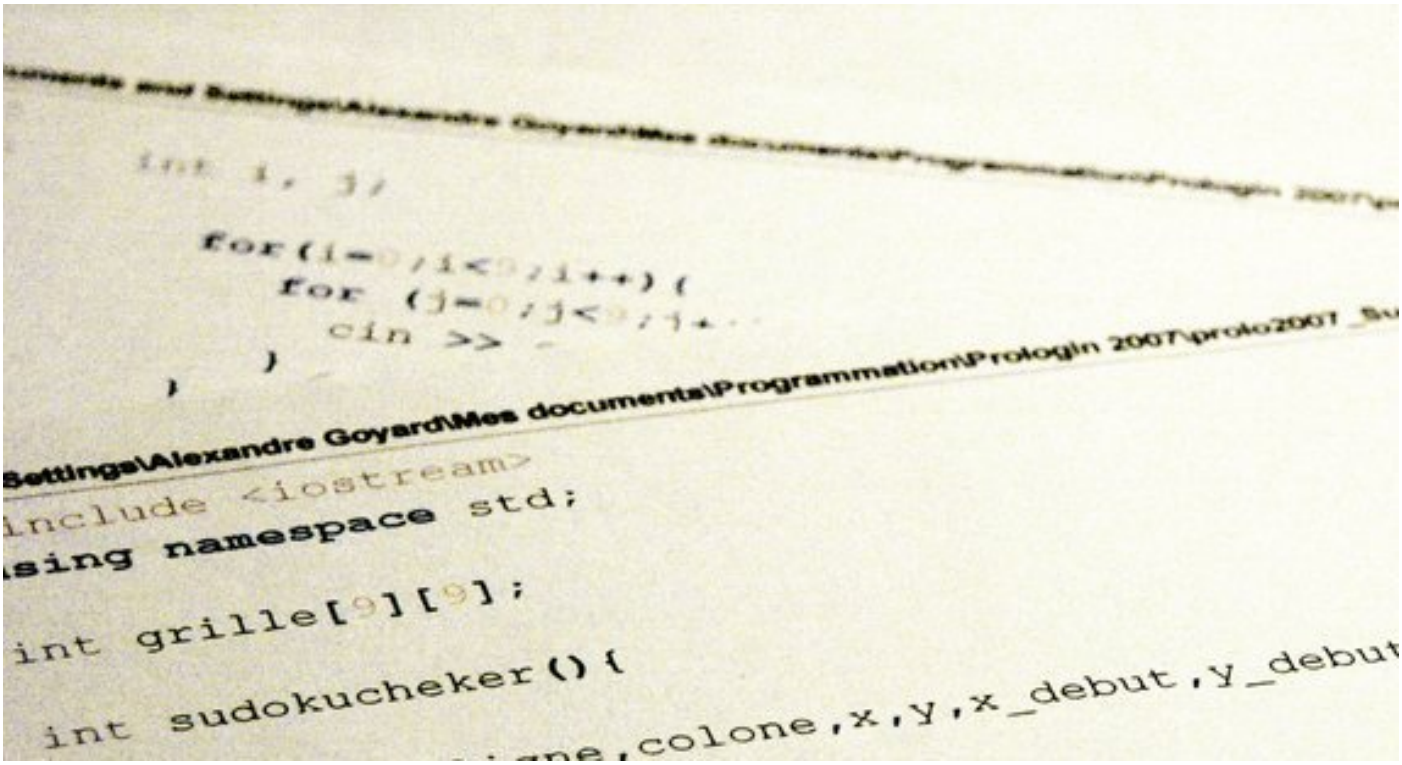
• Nem sempre se obtêm um programa circular válido, que termine.

• As várias travessias tem de seguir o mesmo padrão recursivo.

• Devido ao ponto anterior, as travessias tem de ser sempre sobre a mesma estrutura de dados.

O último ponto é bastante importante. Muitos casos tornam necessário a formação de estrutura de dados intermédia entre as várias travessias. Por exemplo, no caso do exemplo da média, se quiséssemos que a lista obtida estivesse ordenada, então o mais normal era usar como uma estrutura intermédia uma árvore binária e implementar algo tipo o quicksort ou mergesort. Este novo problema deita logo abaixo o nosso método simples. Para quem tiver interessado recomendo que pesquise um pouco, esta é uma área com alguma investigação e existem vários papers onde são abordados novos métodos (mais complexos que não se enquadram neste artigo), mas que ultrapassam estes problemas referidos.

De salientar, que apesar de ser uma área um pouco obscura, já começa a ver-se alguns casos práticos de programas circulares. Sendo os mais usuais, o "pretty-printing" em que são necessárias várias travessias por questões de formatação ou então compiladores em que também é



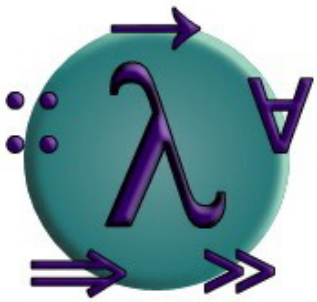
necessário várias travessias para calcular por exemplo os endereços de "jump".

Extras

• Página do Haskell:
http://haskell.org/

• Introdução a Arrows:
http://www.haskell.org/arrows/index.html

• Pequeno artigo sobre programas circulares que serviu de forte inspiração:



http://www.haskell.org/haskellwiki/Circular_programming
• The Monad Reader Issue 6 (Assembly: Circular Programming with Recursive do):
http://www.haskell.org/sitewiki/images/1/14/TMR-Issue6.pdf

SOBRE O AUTOR

Natural de Santa Maria da Feira, Miguel Bento encontra-se desde Novembro 2006 na Creativesystems, uma das principais líderes na área de RFID na península Ibérica. Tem como preferência a nível industrial a linguagem C# mas aproveita o seu tempo livre para aprofundar os seus conhecimentos na sua linguagem predilecta, Haskell.

Miguel Bento

Serialização e desserialização de Objectos em C#

Introdução

Num projecto recente, desenvolvi um sistema multi-tier em que alguns dados eram provenientes de ficheiros XML, em que também era necessário escrever dados para ficheiros em XML. Na definição dos objectos de negócio da aplicação, além das propriedades estavam incluídos dois métodos, ToXML() e FromXML(XmlNode), que convertiam os dados do objecto para XML ou recebiam os dados em XML e preenchiam o objecto com esses dados.

Inicialmente, os métodos de conversão de XML para objecto e de objecto para XML estavam incluídos na classe que definia o objecto, com a personalização necessária à classe. Esta abordagem era evidentemente má, especialmente quando o número de classes era grande pois não permitia a reutilização do código. Com tempo foi possível desenvolver uma classe generalizada, que permite a serialização e deserialização em XML dos objectos das diversas aplicações que integram a classe, aumentando a eficiência de desenvolvimento.

O artigo que segue, apresenta a estrutura típica em que é usado, a classe XmlCustomSerializer que efectua a serialização e desserialização do objecto automaticamente, e um exemplo de como utilizá-la.

Enquadramento

As aplicações que tenho desenvolvido, essencialmente na framework .NET (ASP.NET com C#), têm mantido a mesma estrutura multicamada. Aprendi este método e adaptei-o às minhas necessidades após a leitura de um conjunto de artigos do Imar Spaanjaars (http://imar.spaanjaars.com – "Building Layered Web Aplications With Microsoft ASP.NET 2.0").

Resumidamente, o método que ele descreve inclui uma camada de acesso a dados (DAL) uma camada de lógica de negócio (BLL) e a camada de apresentação. Comuns a estas 3 camadas estão os objectos de negócio (BO) que pouco mais são que contentores de objectos – classes definidas apenas com propriedades e, nalguns casos, alguns métodos que possam ser usados nas várias camadas. Podem ser incluídas mais ou menos camadas, que o conceito mantem-se - há a classe de suporte às propriedades, global às camadas, e um conjunto de classes gestoras, em cada camada.

A DAL tem tipicamente o conjunto de métodos CRUD de acesso à base de dados ou repositório de ficheiros. As classes da camada são gestores dos objectos no que diz respeito a acesso a dados. Por exemplo uma classe Pessoa, classe que é um BO com as propriedades de uma pessoa da aplicação, tem na DAL uma classe PessoaDAL, com o conjunto de métodos CRUD para gerir os dados de Pessoa na base de dados ou repositório. A classe tem os métodos Get(), GetList(), Insert(Pessoa), Update(Pessoa), Delete(Pessoa) e outro relevantes.

A BLL é composta também por classes gestoras dos objectos, mas com os métodos relevantes à camada de negócio – validações, autenticação, processamento de dados recebidos da aplicação ou da DAL, etc. Em alguns casos, os métodos da BLL são simples chamadas do método semelhante na DAL. A camada de apresentação tem o conjunto de métodos relevantes à apresentação da informação na aplicação e processamento das acções do utilizador.

Esta estrutura permite tornar transparente a forma como funcionam as diversas camadas. Apenas passa o conjunto de dados gerado pelas camadas para a seguinte, e por exemplo a camada de apresentação nunca terá acesso directo à camada de escrita de dados. Mais, é possível separar e distribuir as diversas camadas através de Remoting e/ou Web Services partilhando recursos entre aplicações ou distribuindo o esforço por diversas máquinas.

Serialização no .NET

A serialização é um mecanismo importante na framework do .NET, em que os objectos são transformados numa stream de dados, formato portável, quer para utilizar remotamente, quer para persistir dados. No caso dos WebServices os dados das respostas são convertidos para XML, e o objecto a partir desses dados pode ser reconstruído. A serialização e desserialização neste caso é realizado automaticamente pela framework.

Na framework há três modos de serialização:

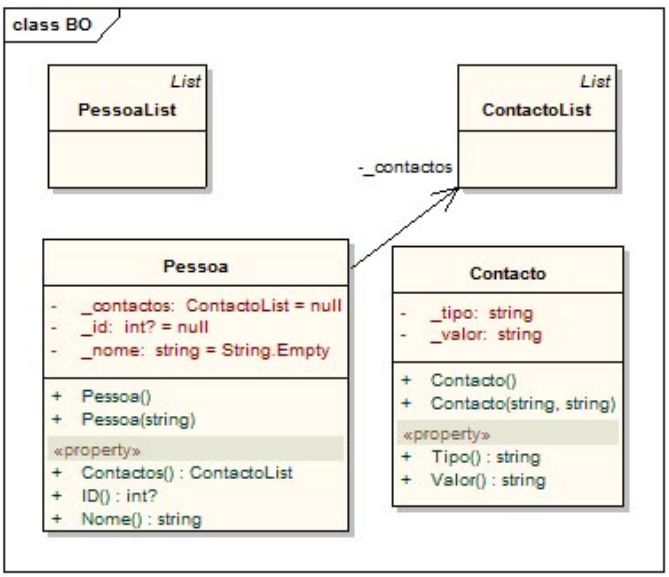
- Binária – mais leve e que apresenta a melhor performance e usado no remoting.
- SOAP – usado por Webservices para converter os objectos em XML.
- XML – serialização costumizada para um formato desejado.

Quer a serialização binária quer a SOAP efectuam Deep Serialization no sentido que serializam o objecto na sua totalidade, incluindo outros objectos contidos nele. Já o XML, porque apenas serializa os elementos públicos do objecto, e é feita da forma como nós indicamos, é chamada de Shallow Serialization.

O resto do artigo está ligado à serialização costumizada em XML. É bastante útil para passar dados que estão em ficheiros XML directamente para um objecto ou vice versa. Há aplicações em que uma base de dados é overkill, e meia dúzia de ficheiros XML resolvem para suportar os dados. Escrever código dedicado a cada objecto é também muitas vezes desnecessário (a menos que siga um formato muito próprio e os objectos em uso não admitam a conversão directa).

Para o demonstrar vou criar um pequeno exemplo. Este servirá para apresentar o modo de implementação dos objectos como contentores, a classe genérica de serialização e a sua aplicação.
Exemplo - lista de contactos

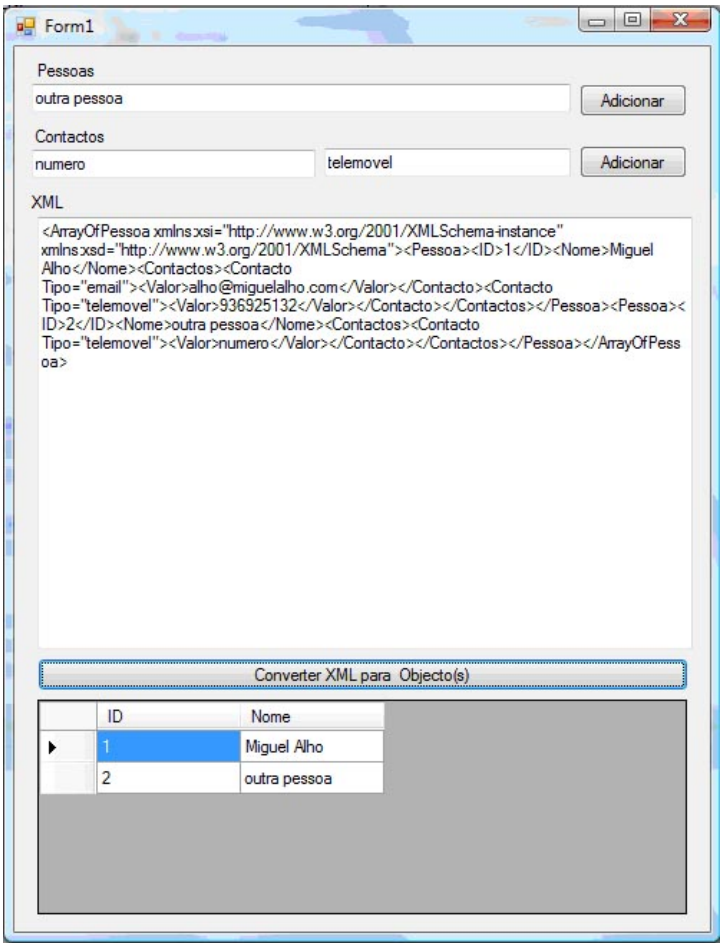
Um bom exemplo é uma lista de contactos. Os objectos de negócio que vou utilizar são:



A classe Pessoa contém os dados de uma pessoa (ID, nome e contactos) e utiliza ContactoList para associar vários contactos ao mesmo registo de Pessoa. ContactoList não é mais que uma lista genérica do tipo Contacto que contem os dados do contacto individual (tipo de contacto e o valor). PessoaList é também uma lista genérica, mas do tipo Pessoa.

A aplicação exemplo não utiliza as camadas BLL e DAL (a simplicidade não exige). No entanto declarei-os para apresentar um exemplo de como funciona. Como é possível ver no código, a DAL tem o conjunto de métodos que permite aceder à fonte de dados e retornar os objectos. Apenas esta camada tem acesso aos dados, o que torna o processo de obtenção de dados transparente para as camadas superiores. A BLL muitas vezes é uma simples interface para os métodos da DAL, podendo ter mais processamento, como validação e autenticação.

A camada de apresentação é constituída pelos vários formulários que a aplicação pode ter. Estes formulários apenas conhecem os métodos públicos da BLL que por sua vez apenas conhece os métodos públicos da DAL. A separação é implementada através de namespaces, e cada camada conhece apenas a camada seguinte. Este modo implementa uma clara separação da funcionalidade de cada camada.



Relativo à aplicação, a funcionalidade implementada é o mínimo necessário para demonstrar o serializador. Primeiro, tem um campo que permite criar uma instância do objecto Pessoa através do nome. O ID é atribuído automaticamente pelo processo de inserção na lista PessoaList. À última pessoa criada é possível associar uma lista de contactos adicionando um contacto de cada vez. A caixa de texto apresenta o XML gerado pelo serializador a partir da

instância do objecto PessoaList que suporta os dados. Por fim, é possível converter os dados do XML em numa nova instância de PessoaList.

A classe Form1 do formulário tem um membro declarado do tipo PessoaList para suportar a lista de pessoas:

```
public partial class Form1 : Form
{
PessoaList aLista = new PessoaList();
```

A adição de uma pessoa cria um objecto pessoa, com o nome preenchido e adiciona-o à lista de pessoas:

```
private void button1_Click(object
sender, EventArgs e)
{
      Pessoa p = new
Pessoa(textBox1.Text);
      aLista.Add(p);

      //o ID neste exemplo simples
passa a ser a posição na lista
      aLista[aLista.Count-1].ID =
aLista.Count;

      //Com o construtor certo, as
três linhas de cima poderiam ser
escritas numa só, do género:
      // aLista.Add(new
Pessoa(aLista.Count, textBox1.Text));
//com ID inicial 0

      //SaveList - aqui surgiria a
chamada ao processo para armazenar os
dados.
      ShowXML();
}
```

A criação do ID é simulado neste caso – o ID após a inserção na lista é criado a partir do índice do registo na lista. O processo de armazenar os dados também é simulado, apenas mostrando o XML gerado na caixa de texto (o ShowXML será descrito mais à frente.)

Para adicionar contactos temos:

```
private void button2_Click(object
sender, EventArgs e)
{
      Contacto c = new
Contacto(textBox2.Text,textBox4.Text);

      if (aLista[aLista.Count -
1].Contactos == null)
            aLista[aLista.Count -
1].Contactos = new ContactoList();

      aLista[aLista.Count -
1].Contactos.Add(c);

      ShowXML();
}
```

É criado um novo contacto com o par valor / tipo no construtor. Se a lista de contactos da pessoa é nulo, é criado nova instância da lista. Finalmente é adicionado o contacto à lista e armazenada a lista.

Serializador costumizado

Preparação dos objectos

Porque pretendemos serializar os objectos de forma automática para um formato costumizado, temos que preparar os objectos para essa tarefa. O .Net tem um conjunto de atributos que devemos adicionar ao código dos objectos para tornar a construção e desconstrução do objecto em XML possível. Vamos primeiro analisar a classe Contacto:

```
public class Contacto
{
  public Contacto() { }

        public Contacto(string
contacto, string tdc)
        {
            this.Valor = contacto;
            this.Tipo = tdc;
        }

        private string _valor;
        [XmlElement]
        public string Valor
        {
            get { return _valor; }
            set { _valor = value; }
        }

        private string _tipo;
```

```
        [XmlAttribute]
        public string Tipo
        {
            get { return _tipo; }
            set { _tipo = value; }
        }
}
```

A classe tem dois construtores, um vazio (necessário para o serializador) e outro parametrizado. Contém também as propriedades, com membros públicos e privados. O atributo essencial para o processo de serialização é o que está declarado imediatamente antes do membro público. [XmlElement] antes de Valor, indica que valor deve ser serializado como sendo um elemento XML; [XmlAttribute] antes de Tipo indica que Tipo será um atributo do elemento Contacto. É importante notar que os tipos nuláveis (ainda) não são serializáveis como XmlAttribute. O objecto serializado terá a seguinte estrutura:

```
<Contacto Tipo="telemovel">
   <Valor>912345678</Valor>
</Contacto>
```

ContactoList, que é declarado como:

```
[XmlType(TypeName="Contactos")]
public class ContactoList :
List<Contacto>
{
}
```

Será serializado com a forma:

```
<Contactos>
   <Contacto Tipo="telemovel">
      <Valor>912345678</Valor>
   </Contacto>
   <Contacto Tipo="email">

<Valor>alho@miguelalho.com</Valor>
   </Contacto>
   (...)
</Contactos>
```

Por defeito, as listas, sem qualquer modificador no seu atributo, são serializadas como "ArrayOfTipo" (no caso de ContactoList seria ArrayOfContacto). A utlização do atributo XMLType com a propriedade TypeName sobre a classe permite alterar o nome com que é serializado. O mesmo é possivel com o XmlElement através do parametro ElementName.

A classe Pessoa e PessoaList são:

```
public class Pessoa
{
        /// <summary>
        /// construtor vazio é
necessário para a serialização
        /// </summary>
        public Pessoa() { }

        public Pessoa(string nome)
        {
            this.Nome = nome;
        }

        /// <summary>
        /// id é nullo quando se
adiciona o contacto, logo usemos um
elemento nulável
        /// </summary>
        private int? _id = null;
        [XmlElement(IsNullable=true)]
        public int? ID
        {
            get
            {
                if (_id.HasValue)
return _id.Value;
                else return null;
            }
            set
            {
                if (value.HasValue)
_id = value.Value;
                else _id = null;
            }
        }

        private string _nome =
String.Empty;
        [XmlElement]
        public string Nome
        {
            get { return _nome; }
            set { _nome = value; }
        }

        private ContactoList
_contactos = null;
        [XmlArray(IsNullable=true)]
```

```
        public ContactoList Contactos
        {
            get { return _contactos; }
            set { _contactos = value;
}
        }
    }

    [XmlType(TypeName = "Pessoas")]
    public class PessoaList :
List<Pessoa>
    {


    }
```

Aqui, a primeira novidade é o uso de [XmlArray] para indicar que Contactos (do tipo ContactoList) é efectivamente uma lista e deve ser apresentado no XML como um array de Contacto.

A segunda novidade é o IsNullable no atributo de [XmlElemento] da propriedade ID e no [XmlArray] . Caso seja nulo, no XML gerado será escrito na tag xsi:nil="true" indicando explicitamente que é nulo. Sem o IsNullable, no caso de o valor da propriedade ser nulo, o respectivo XML é ignorado. Caso haja algum elemento público que não desejamos passar para o XML (como um método que retorna um valor baseado nas propriedades) podemos usar o [XmlIgnore].

Usando o programa exemplo, se adicionar uma Pessoa (instanciar uma pessoa com o nome preenchido), o XML gerado é:

```
<Pessoas
xmlns:xsi="http://www.w3.org/2001/XMLSc
hema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSc
hema">
   <Pessoa>
      <ID>1</ID>
      <Nome>Miguel Alho</Nome>
      <Contactos xsi:nil="true"/>
   </Pessoa>
</Pessoas>
```

A informação do namespace (xmlns:xsi...) é adicionada automaticamente pelo serializador e apenas no primeiro elemento.

Se adicionarmos mais pessoas e contactos, temos:

```
<Pessoas
xmlns:xsi="http://www.w3.org/2001/XMLSc
hema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSc
hema">
   <Pessoa>
      <ID>1</ID>
      <Nome>Miguel Alho</Nome>
      <Contactos>
         <Contacto Tipo="telemovel">
            <Valor>912345678</Valor>
         </Contacto>
         <Contacto Tipo="email">

<Valor>alho@migeulalho.com</Valor>
         </Contacto>
      </Contactos>
   </Pessoa>
   <Pessoa>
      <ID>2</ID>
      <Nome>outra pessoa</Nome>
      <Contactos>
         <Contacto Tipo="email">
            <Valor>outro@email</Valor>
         </Contacto>
      </Contactos>
   </Pessoa>
</Pessoas>
```

Os objectos que são arrays são correctamente convertidos para listas de objectos, as propriedades são correctamente serializadas e apresentadas da forma que escolhi. Resta ver então como fazer a serialização, propriamente.

XmlCustSerializer

A classe XmlCustSerializer é que efectua a conversão de e para Xml. Está incluída no namespace dos BO para poder ser usada nas diversas camadas da aplicação. É escrita na forma de classe genérica de modo a suportar qualquer tipo que definimos.

```
public class XmlCustSerializer<T>
{
public static XmlElement ToXml(T obj)
//XmlDocument xd) {
        {
        //cria stream na memória para
suportar o Xml
        MemoryStream memoryStream =
new MemoryStream();
```

```
        try
        {
            //Instancia o serializador
            XmlSerializer xs = new
XmlSerializer(obj.GetType());
            //Instancia o
XmlTextWriter
            XmlTextWriter
xmlTextWriter = new
XmlTextWriter(memoryStream,
Encoding.UTF8);
            //Serializa os dados da
classe para o writer

xs.Serialize(xmlTextWriter, obj);

            //Copia a stream do
TextWriter para a memória
            memoryStream =
(MemoryStream)xmlTextWriter.BaseStream;

            //Coloca o apontador para
o início da stream
            //(necessário para evitar
erros no carregamento dos XMlDoc)
            memoryStream.Position = 0;
            //Instancia o XmlDocument
            XmlDocument xd = new
XmlDocument();
            //carrega o Stream em XMl
para o XmlDocument
            xd.Load(memoryStream);
            //Obtém o nó raiz com os
dados a retorna
            XmlElement node =
xd.DocumentElement;
            //retorna o nó raiz.
            return node;
        }
        catch
        {
            throw new
InvalidCastException("Ocorreu um erro
na serialização do objecto
(1009002001)");
        }
        //return registo;
    }
```

O ToXml recebe um objecto do tipo T e converte-o em Xml, consoante os atributos da classe. O processo requer a serialização para uma stream de memória através de um XmlTextWriter. Posteriormente o stream é lido para um XMLDocument e o elemento XML é retornado. Como é possível verificar, o XmlSerializer efectua a serialização de forma automática, sem código especifico a determinado objecto.

```
    public static T FromXml(XmlNode
O_Xml)
    {

        //Instancia uma representação
de encoding UTF-8
        UTF8Encoding encoding = new
UTF8Encoding();

        try
        {
            //Instancia uma Stream de
memória, para onde é passado o texto
Xml do parâmetro de entrada
            MemoryStream memoryStream
= new
MemoryStream(encoding.GetBytes(O_Xml.Ou
terXml));
            //retorna o apontador da
stream para o início
            memoryStream.Position = 0;
            //instancia a serialização
para o tipo de dados
            XmlSerializer xs = new
XmlSerializer(typeof(T));

            T obj;
            obj =
(T)xs.Deserialize(memoryStream);

            return obj;
        }
        catch
        {
            throw new
InvalidOperationException("Ocorreu um
erro na desserialização do objecto
(1009002002)");
        }
    }
}
```

O método FromXML(XmlNode) efectua o processo inverso – a partir de um nó XML, com o encoding em UTF-8 (possivelmente o método poderá ser alterado para aceitar o encoding como parâmetro), transfere-o para a stream de memória, desserializa-o para o o objecto do tipo T e retorna o objecto com os dados.

Para o usar, basta usar a classe para o tipo desejado, por exemplo:

```
private void ShowXML()
{
 textBox3.Text =
XmlCustSerializer<PessoaList>.ToXml(aLi
sta).OuterXml;
}
```

onde aLista é do tipo PessoaList, e o uso de .OuterXml permite obter uma string. Porque os métodos são estáticos, não é necessário criar uma instância da classe XmlCustSerializer. No caso da desserialização, temos:

```
private void button3_Click(object
sender, EventArgs e)
{
 XmlDocument xDoc = new XmlDocument();
 xDoc.LoadXml(textBox3.Text);

 dataGridView1.DataSource =
XmlCustSerializer<PessoaList>.FromXml(x
Doc.FirstChild);
}
```

Na aplicação, o XML contido na área de texto é transformado e XmlDocument e desserializado para a nova PessoaList, que serve de datasource ao dataGridView. Neste exemplo, a serialização e desserialização é usada apenas para apresentar dados, mas podia ser usada para ler os dados de um ficheiro ou escrever para um ficheiro, por exemplo, tornando a conversão automática.

O processo também é bastante permissivo, no sentido que se retirar elementos do XML, posso construir o objecto na mesma (desde que a propriedade permita o nulo por defeito). Por exemplo:

```
<Pessoas >
   <Pessoa>
      <Nome>Miguel Alho</Nome>
   </Pessoa>
</Pessoas>
```

É correctamente reconstruído como objecto PessoaList, em que as propriedades ID e Contactos da única Pessoa na lista são nulos. Esta caracteristica pode ser muito útil na definição de um formato de ficheiro que uma aplicação deva receber, e que não necessite de ter os dados completos (diminuindo o tamanho do ficheiro). Depois, após recepção, o objecto desserializado é processado e os campos nulos poderão ser validados ou preenchidos pela aplicação.

Da mesma forma que podem haver ausência de tags, pode existir tags e atributos redundantes:

```
<Pessoas data="today" origem="a">
   <Pessoa numero="1">
      <Nome>Miguel  Alho</Nome>
      <criado modo="automÃ¡tico"/>
   </Pessoa>
</Pessoas>
```

Por exemplo os atributos data, origem, e numero, e o elemento criado não são relevantes para a nossa aplicação. No entanto, o objecto PessoaList é construído correctamente, ignorando o que está a mais. Podemos por exemplo guardar o ficheiro num repositório para conservação, mas usar apenas os dados relevantes na aplicação.

Conclusão

Penso que é evidente a utilidade da classe XMLCustSerializer.

• Implementa o processo de serialização e desserialização de objectos da aplicação para XML de forma automática e customizada.

• Os mesmos métodos aceitam objectos de tipos customizados, reutilizando código de forma eficiente.

• A conversão para XML permite implementar métodos mais simples de armazenamento dos dados em ficheiros, e métodos de leitura dos mesmos.

• O uso de XML é útil para interoperacionalidade da aplicação.

O uso do XML tem custos, no entanto:

• O overhead das tags é grande, e é sentida em ficheiros com grande quantidade de dados (o XML pode ocupar 10x mais que um CSV equivalente).

Uma forma de o contornar é usar attributos onde possível, e utilizar nomes curtos para os elementos.

• Em comparação aos outros modos de serialização, a binária é mais rápida.

Eu pessoalmente vou utilizando o formato XML, porque se necessário, posso editar um ficheiro gerado à mão, e a conversão directa para objecto facilita a produção de código de armazenamento e leitura dos ficheiros e facilita o

processo de declaração do formato de armazenamento.

Bibliografia e Créditos

A classe XmlCustSerializer foi desenvolvido em conjunto com Marco Fernandes e Joaquim Rendeiro.

Algumas fontes usadas para compor a parte teórica do artigo:

• Building Layered Web Aplications With Microsoft ASP.NET 2.0
http://Imar.spaanjaars.com

• Introducing Serialization in .NET , Joydip Kanjilal
http://aspalliance.com

• Serialization in .NET , Nishith Pathak
http://www.CodeProject.com

• XmlSerializer Class
http://msdn2.microsoft.com

• System.Xml.XmlSerializer, Nitin Pande
http://www.eggheadcafe.com

SOBRE O AUTOR

Natural da Murtosa, Miguel Alho é licenciado em Engenharia Electrónica e Telecomunicações pela Universidade de Aveiro. Actualmente trabalha como programador freelancer, desenvolvendo aplicações web e multimédia para entidades nacionais e internacionais, utilizando principalmente a tecnologia .NET da Microsoft.

Miguel Alho

Manipulação de Datas

Introdução

No mundo da programação, é comum necessitarmos de algum utilitário que efectue cálculos com datas do calendário gregoriano. Os ambientes de programação integrados normalmente possuem tais funções já definidas internamente, bastando ao programador apenas fazer uso das mesmas. A principal questão deste artigo é responder à questão: como podem esses cálculos ser processados? Para solucionar este tipo de problemas, é conveniente conhecer e utilizar algoritmos específicos. Existem várias formas e soluções, no entanto é aconselhável utilizar aquelas que já tenham sido testados como, por exemplo, algoritmos utilizados na área da Observação Astronómica, que possibilitam calcular qualquer data a partir do ano 4712 a.C. Para entender melhor o processo do cálculo de datas, torna-se necessário entender os conceitos de: Dia Juliano, Calendário Juliano, Calendário Gregoriano e Ano Bissexto.

Dia Juliano

O dia Juliano é um valor sequencial positivo inteiro que representa uma contagem sucessiva de dias a partir do ano 4712 a.C., ou seja, do ano -4712. A data 01/01/2000 seria representada como dia Juliano com o valor 2.451.545. Assim sendo, é possível calcular a diferença entre datas distantes. Por exemplo, quantos dias existem entre as datas 01/01/2000 e 26/04/1965? Para obter o número de dias entre as datas, é necessário converter cada uma das datas para os respectivos valores sequenciais para então efectuar o cálculo da diferença de dias entre as datas desejadas. Desta forma, a data 26/04/1965 (2.438.877) subtraída da data 01/01/2000 (2.451.545) resulta no valor 12.688 dias.

Calendário Juliano

O calendário Juliano foi instituído no ano 46 a.C. (época do imperador romano Júlio César) e utilizado até 04/10/1582. Nesta ocasião foram considerados três anos de 365 dias e um de 366, a cada quatriénio (ano bissexto). Nesse período, o mês de Fevereiro possuía 29 dias, tendo 30 dias apenas nos anos bissextos. Em homenagem ao imperador foi dado o nome de Julho ao sétimo mês do Calendário Juliano, o qual possui até hoje um ciclo de 31 dias. A partir do ano 8 a.C., tempo do imperador César Augusto, houve a mudança do nome do oitavo mês para Agosto (em sua homenagem), e como o mês de Julho possuía 31 dias, foi então atribuído mais um dia ao mês de Agosto, passando este a ter um ciclo de 31 dias, tal como o mês de Julho e, para que não houvesse diferença entre as homenagens. Este dia a mais foi retirado do mês de Fevereiro, que passou a ter um ciclo de 28 ou 29 dias nos anos bissextos. Desde então, ficaram instituídas as regras para os meses com 31 dias (Janeiro, Março, Maio, Julho, Agosto, Outubro e Dezembro), com 30 dias (Abril, Junho, Setembro e Novembro) e com 28 ou 29 dias (Fevereiro).

Calendário Gregoriano

O calendário Gregoriano foi instituído na reforma papal do pontificado do Papa Gregório XIII. Um facto curioso é que o calendário Gregoriano foi iniciado em 05/10/1582, tendo uma diferença de 10 (dez) dias em relação ao término do calendário Juliano. Observe como ficou o calendário do mês de Outubro de 1582, quando ocorreu a junção dos calendários Juliano e Gregoriano.

Outubro 1582						
Dom	Seg	Ter	Qua	Qui	Sex	Sáb
	1	2	3	4	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

A supressão de dez dias ocorreu devido a um erro no ciclo do número de dias de um ano, existente no calendário Juliano, que prevê um ano de 365,25 dias, quando este ciclo é de 364,24219271 dias, sendo reduzido a uma razão de 0,005369 segundo por ano.

Ano Bissexto

Os cálculos para determinar anos bissextos nos calendários Juliano e Gregoriano são diferentes. No calendário Juliano é ano bissexto todo ano divisível por 4 (quatro), porém no calendário Gregoriano, devido às correcções e para maior precisão astronómica, considera-se bissexto o ano divisível

por 4 (quatro), excepto os anos terminados em 00 que para serem bissextos necessitam ser divisíveis por 400 (quatrocentos). O ajuste nos anos terminados com 00 ocorre a cada 400 (anos), devido ao ciclo de um ano possuir 364,24219271 dias, sendo reduzido a uma razão de 0,005369 segundo por ano, ciclo denominado de ano trópico, quando ocorre o equinócio (quando o Sol está exactamente sobre a linha do equador no início da primavera ou no início do mês de Outubro; nesta ocasião o dia e a noite têm a mesma duração temporal).

Algoritmos

A partir da exposição dos conceitos anteriores é apresentado os algoritmos utilizados para conversão de data em dia Juliano, conversão de dia Juliano em data e verificação de ano bissexto. Conversão de Data em Dia Juliano Para converter uma data do calendário Juliano ou calendário Gregoriano em dia Juliano deve-se fazer uso de um algoritmo específico que transforma uma data num número sequencial, como o demonstrado:

```
Leia DIA, MÊS, ANO
Se (MÊS < 3) Então
  ANO <- ANO - 1
  MÊS <- MÊS + 12
Fim_se
Se (DATA >= 15/10/1582) Então {Calendário
Gregoriano}
  A <- Inteiro(ANO / 100)
  B <- Inteiro(A / 4)
  C <- 2 - A + B
Senão {Calendário Juliano}
  C <- 0
Fim_se
D <- Inteiro(365,25 * (ANO + 4.716))
E <- Inteiro(30,6001 * (MÊS + 1))
F <- D + E + DIA + 0,5 + C - 1.524,5
JULIANO <- Inteiro(F)
Escreva JULIANO
```

O algoritmo de conversão verifica primeiro se o MÊS é menor que três. Sendo a condição verdadeira, é feito um ajuste nos valores de ANO e MÊS. Em seguida, é verificado se a data fornecida pertence ao calendário Gregoriano (datas a partir de 15/10/1582) ou calendário Juliano (datas anteriores a 04/10/1852). Se a data pertencer à parte do calendário Gregoriano, a variável C é ajustada a partir das variáveis A e B; caso contrário, à variável C é atribuído o valor de zero. Após a verificação da condição e estando a variável C ajustada, são realizados cálculos para determinar o número de dias a partir de 4712 a.C.

Conversão de Dia Juliano em Data

Para converter um dia Juliano em data é necessário também seguir um algoritmo específico de conversão, que tem em consideração o facto de um ano ser ou não bissexto.

```
Leia JULIANO
A <- JULIANO
Se (A > 2.299.160) Então
  B <- Inteiro((A - 1.867.216,25) /
36.524,25)
  C <- A + 1 + B - Inteiro(B / 4)
Senão
  C <- A
Fim_se
D <- C + 1.524
E <- Inteiro((D - 122,1) / 365,25)
F <- Inteiro(E * 365,25)
G <- Inteiro((D - F) / 30,6001)
H <- D - F - Inteiro(G * 30,6001)
Se (G < 14) Então
  I <- G - 1
Senão
  I <- G - 13
Fim_se
Se (I > 2) Então
  J <- E - 4.716
Senão
  J <- E - 4.715
Fim_se
Se (J > 0) Então
  K <- J
Senão
  K <- Módulo(J + 1) {Módulo = valor
positivo}
Fim_se
DIA <- H
MÊS <- I
ANO <- K
Escreva DIA, MÊS, ANO
```

O algoritmo em questão atribui a variável A o valor de dias Juliano, e em seguida verifica se o valor de dias Juliano é maior que 2299160 (no calendário Gregoriano, equivale a 04/10/1582). Se esta condição for verdadeira, são executados os devidos ajustes de tempo para o valor da variável C; caso contrário (sendo a data pertencente ao calendário Juliano), a variável C é implicada directamente pelo valor da variável A (sem os ajustes). Seguidamente, é executada uma série de cálculos para extrair os valores do DIA, MÊS e ANO do dia Juliano fornecido. Verificação de Ano Bissexto Para verificar se o ano de uma data é ou não bissexto deve-se fazer uso do algoritmo apresentado em seguida.

```
Leia ANO
Se (ANO mod 4 <> 0) Então
```


```
  ANO NÃO É BISSEXTO
  Se (ANO mod 100 = 0) Então
    Se (ANO mod 400 = 0) Então
      Escreva "ANO É BISSEXTO"
    Senão
      Escreva "ANO NÃO É BISSEXTO"
    Fim_se
 Senão
    Escreva "ANO É BISSEXTO"
  Fim_se
Fim_se
```

Com o algoritmo de ano bissexto é possível verificar primeiro se uma data é divisível por quatro. Pode ser divisível quando o resto da divisão do ano por quatro for diferente de zero (considerando como quociente da divisão um resultado sempre inteiro). Neste caso, existe a possibilidade de o ano ser bissexto. Dentro dessa possibilidade, é utilizada uma segunda condição (para detectar os anos que terminam com 00), em que se verifica se o ano é divisível por cem. Se o ano for divisível por cem, é possível ainda ser bissexto. Para verificar esta possibilidade utiliza-se a terceira condição quando o ano é dividido por 400.

Referência Bibliográfica

MANZANO, J. A. N. G.; YAMATUMI, W. Y. Free Pascal: Programação de Computadores. São Paulo, Brasil: Editora Érica, 2007. 390 p., ISBN 978-85-365-0136-9.

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

Revista Linux

A Verdadeira Revista Portuguesa de Linux

Edição Bimestral em formato PDF

Download Gratuito

www.revista-linux.com

Internet Protocol Version 6 – IPv6

IPv6, também conhecido por IPng, é chamado por muitos de protocolo da próxima geração. E visto que estamos cada vez mais próximos do upgrade da versão 4 para a 6 convém começarmos a familiarizar-nos com este protocolo e no que traz de novo. Grande parte das pessoas não sabe que estamos a chegar ao limite do IPv4. Na verdade, dos teoricamente 4.3 mil milhões de endereços que o IPv4 permite apenas temos disponíveis cerca de 250 milhões endereços. E ainda existem muitas áreas onde o IP pode e vai ser aplicado. Além destas aplicações que se pretendem fazer, também temos que levar em conta que apenas cerca de 10% da população mundial está ligada à Internet e que países como por exemplo a China e Índia ainda agora começaram a ter acesso a ela. O IPv4 não consegue fazer face a todas estas necessidades, daí a necessidade de haver uma mudança. Muito se tem feito para tentar atenuar a escassez de endereços com o IPv4, tem-se recorrido à Variable Lenght Subnet Masks (VLSM) e ao Network Address Protocol (NAT) para tentar aproveitar ao máximo os endereços disponíveis. Mas o IPv6 não se trata apenas de ter mais espaço, não, este protocolo sofreu uma modificação radical tanto na sua estrutura como no seu funcionamento. Foi melhorado significativamente para oferecer mais capacidade, eficácia, flexibilidade e optimização para ir ao encontro das necessidades actuais.

Benefícios e Usos do IPv6

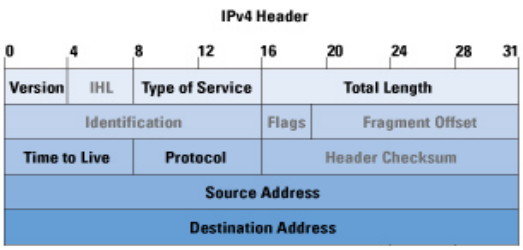
Talvez o que salte logo á vista seja o facto de ao invés dos 4 mil milhões de endereços disponíveis, passamos a poder usar cerca de 340.282.366.920.938.463.463.374.607.431.768.211.456 de endereços. Mas existem outras características que fazem com que uma mudança para o IPv6 seja benéfica e que compense o tempo, o custo e o esforço da mudança. Muitas das coisas que existem hoje numa rede e em especial na Internet não foram pensadas quando se desenvolveu o IPv4 e por isso tem-se recorrido a muitos "remendos" para tentar colmatar essas falhas. Além de que as tecnologias mais recentes começam apresentar alguma dificuldade em trabalhar com o IPv4. Existem outros benefícios que vão surgir com a utilização do IPv6:

- Auto-configuração;
- Segurança (IPsec);
- Melhor suporte para diferentes engenharias (diffserv ou RSVP);
- Multicast;
- Melhor suporte para redes ad hoc;

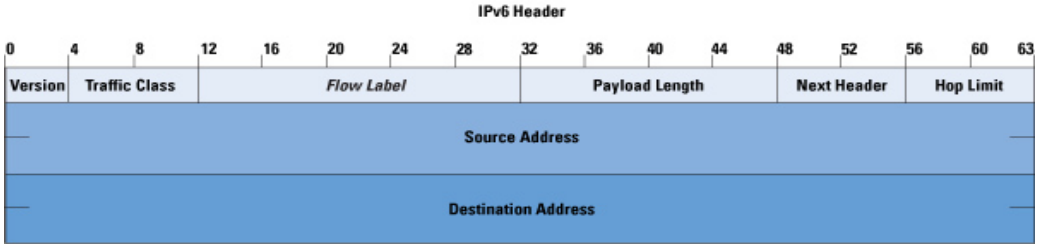
E um dos melhores e grandes benefícios que o IPv6 vai trazer é a eliminação do broadcast por completo.

Estrutura

Como foi referido anteriormente, o IPv6 apresenta mudanças profundas na sua estrutura, em especial no seu cabeçalho (header). Em primeiro lugar 5 campos do cabeçalho no IPv4 foram removidos, Header Length, Identification, Flags, Fragment Offset e Header Checksum. O Header Length foi removido porque o header já tem um tamanho fixo. Os campos Identification, Flags, e Fragment Offset estão relacionados com a fragmentação de pacotes em IPv4, mas foram removidos porque a fragmentação de pacotes em IPv6 funciona de maneira diferente. O campo Header Checksum também foi removido para aumentar a velocidade de processamento. Isto porque se os routers não precisarem de verificar e actualizar os checksums o processamento torna-se mais rápido.



Fonte: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_9-3/ipv6_internals.html



Endereços e expressões no IPv6

É importante referir que os endereços no IPv6 vão ser muito maiores tendo em conta que passamos a usar 128 bits. Por isso em vez do habitual 192.168.10.10 passamos a ter algo como 2001:0db8:3c4d:0012:0000:0000:1234:56ab. A diferença que talvez mais se destaque seja o tamanho e o facto de usarmos 8 conjuntos de números separados por um ": " e em hexadecimal. Também o endereço passa a ser dividido em 3 partes:

- Global Prefix – 2001:0db8:3c4d
- Subnet – 0012
- Interface ID – 0000:0000:1234:56ab

De salientar que o endereço que estamos a usar como exemplo pode ser resumido até ficar assim 2001:db8:3c4d:12::1234:56ab, mas isso já envolve algumas regras que tem de ser seguidas, porque talvez tenham reparado que eliminamos alguns zeros, mas isso nem sempre é possível.

Endereços especiais

Assim como no IPv4 o IPv6 também endereços especiais e que estão reservados para diversas funções: 0:0:0:0:0:0:0:0 (::.) – mesmo que 0.0.0.0 em Ipv4; 0:0:0:0:0:0:0:1 (::1) – mesmo que o 127.0.0.1 em IPv4; Estes são apenas dois exemplos de endereços reservados mais conhecidos, porque muitos outros existem para se referir a tipos específicos de endereços, como global unicast, unique local unicast, link-local unicast, multicast.

Tipos de Endereços

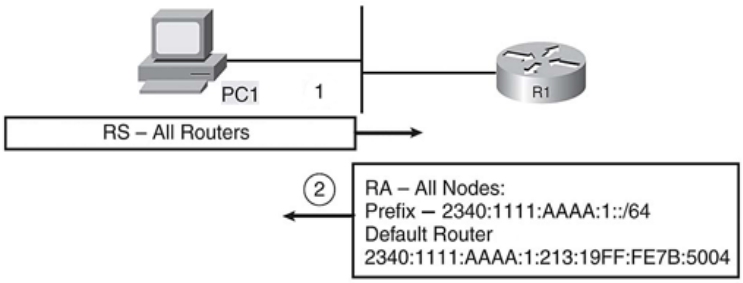
Novamente no IPv6, endereços tais como multicast, unicast que existem no IPv4 e que certamente conhecemos e estamos familiarizados, foram reaproveitados e usados também no IPv6. No entanto um endereço foi completamente eliminado, o broadcast. Lembram-se das broadcast storms? Pois é, esse foi um dos muitos problemas que foi eliminado/resolvido com a eliminação do broadcast e que certamente não vai deixar saudades. Um endereço foi eliminado e um foi criado, estou-me a referir é claro ao Anycast, mas já vamos lá. Só de salientar que os endereços Unicast, Global Unicast, Multicast, Link-Local e Unique Local funcionam basicamente como no IPv4 e sem grandes diferenças ou praticamente nenhumas no IPv6. Mas como dissemos surgiu um novo tipo de endereço, o Anycast. Este endereço identifica diversas interfaces, mas o pacote anycast apenas é entregue a um endereço. Neste caso é entregue ao primeiro endereço que "ele" encontre e que corresponda aos termos definidos pelo routing da rede. Este endereço é usado em conjunto com o protocolo BGP.

Como é que o IPv6 funciona

Pode-se começar por explicar como um host recebe um endereço IPv6 e como consegue encontrar recursos e outros hosts numa rede. Também abordar o que é a stateless autoconfiguration e stateful autoconfiguration.

Auto-configuração

Desde 1993 que se usa o DHCP em IPv4 de maneira a que os host obtenham não só o endereço como também o Gateway e o Sufixo DNS. No IPv6 existem, no entanto, duas maneiras de configuração de um endereço. Uma é usando um DHCPv6 server, que basicamente funciona como o DHCP em IPv4 apesar de existir algumas diferenças nas mensagens usadas, o conceito é o mesmo. A outra maneira é chamada de stateless autoconfiguration. Com a auto-configuração um host pode dinamicamente obter um endereço usando para isso o endereço Link-Local Unicast. Para isto acontecer o host precisa de um prefixo /64 para configurar a interface. É então usado um protocolo do IPv6 chamado de Neighbor Discovery Protocol (NDP). Assim o host envia aos routers uma mensagem NDP chamada de RS (router solicitation) usando para isso uma mensagem IPv6 multicast. Esta mensagem pergunta qual é o prefix IPv6 usado nesta subnet e qual é o endereço(s) usado(s) pelo(s) default router na subnet.. O router por usa vez envia uma resposta com esse prefixo através de um RA (router advertisement) usando novamente o ICMP.



O host pega então o prefixo que recebeu e junta-o com o seu interface ID (MAC) e no meio desse valor junta um FFFE para compensar os bits que faltam. O processo é mais complexo do que isto, porque é usado certo tipo específico de mensagens do ICMP assim como é usado endereços Multicast. Além de que existem diversas opções para a configuração do endereço.

Migrar para o IPv6

Apesar de todas as vantagens apresentadas certamente muitos se perguntem qual é que vai ser o "custo" e o

trabalho que vai dar esta migração. Não podemos pensar que vai haver uma migração em massa do IPv4 para IPv6, isto porque o número de dispositivos com IPv4 ascende aos milhões e nem todos esses dispositivos suportam IPv6 por isso é bem provável que se vá demorar uns bons anos a fazer a transição. Além de que a transição vai depender também da infra-estrutura que tenhamos, como óbvio se tivermos routers ou switches já arcaicos o custo da migração vai ser maior. Mas foram criadas algumas estratégias que permitem uma integração gradual do IPv4 para a versão 6.

- IPv4/IPv6 Dual Stacking

O termo Dual Stack significa que o host ou o router usa o IPv4 e o IPv6 ao mesmo tempo. Para os hosts significa que tem associado ao NIC tanto um endereço IPv4 como um IPv6 e que consegue enviar pacotes IPv4 para hosts com o IPv4 e o mesmo com o IPv6. Para os routers significa que além dos endereços IPv4 e routing protocols também tem configurado endereços IPv6 assim como os respectivos routing protocols. Esta é a estratégia mais usada porque é a mais fácil de se usar isto porque permite o upgrade gradual tanto dos dispositivos como das aplicações.
6to4 Tunneling

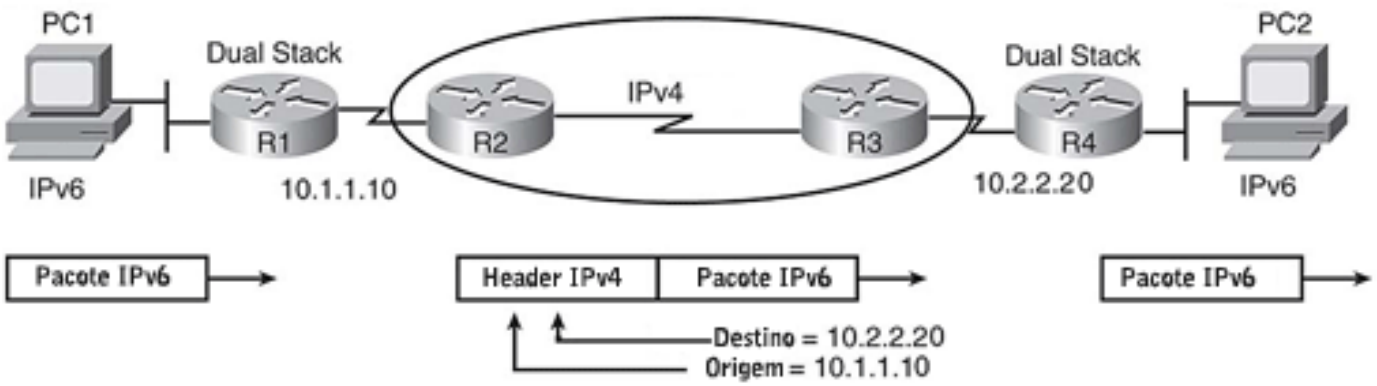
Existem diversos tipos de tunneling, mas neste caso o que é feito é pegar num pacote IPv6 e encapsula-lo dentro de um pacote IPv4. Existem diversos tipos de tunneling que podemos usar: Manually configured tunnels (MCT); Dynamic 6to4 tunnels; Intra-site Automatic Tunnel Addressing Protocol (ISATAP) e; Teredo tunneling. Um aspecto a ter em atenção é que se em algum ponto existir um NAT, este irá "destruir" esse túnel, mas existe uma maneira de dar a volta este problema e é por usarmos um método já citado – Teredo.

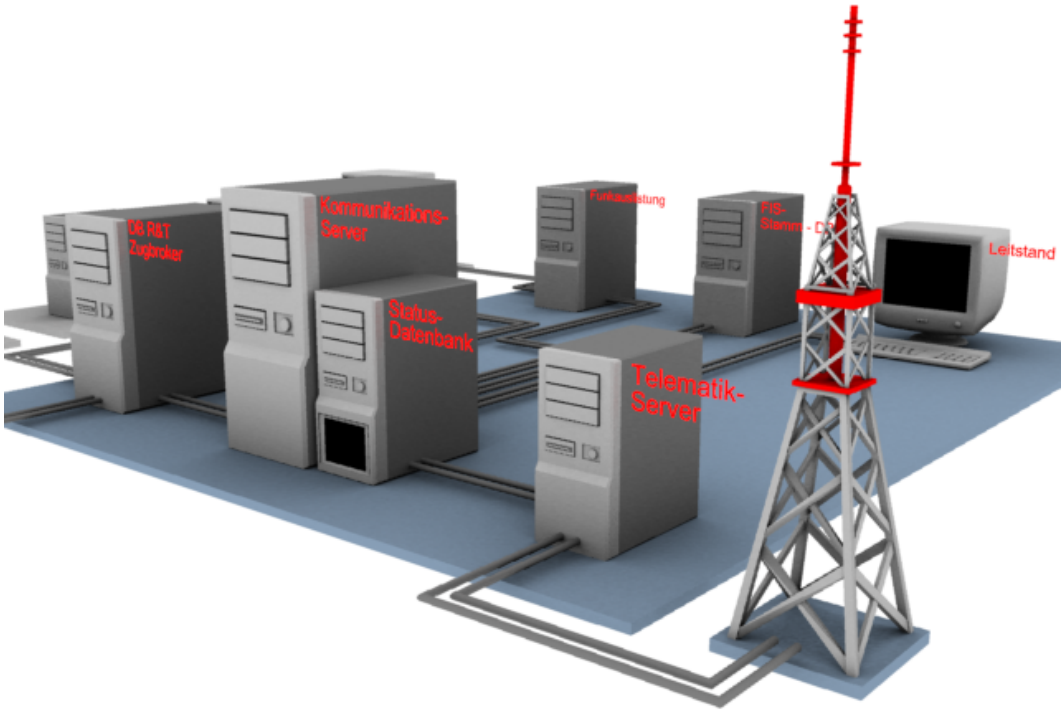
- NAT-PT

Em alguns casos teremos que usar uma terceira opção que permita que se converta um cabeçalho IPv6 em IPv4 e vice-versa. Neste caso temos que usar o Network Address Translation–Protocol Translation (NAT-PT) apesar de não ser uma grande solução. Para que se dê esta conversão é necessário que o router saiba quais os endereços IPv4 e IPv6 deve usar e assim como no NAT tradicional também temos static definition, dynamic NAT, e dynamic PAT.

Segurança

É na segurança que o IPv6 também se destaca e pela positiva. Nesta área viu-se a necessidade de redesenhar e incorporar algumas características básicas em termos de segurança, de maneira a que o IPv6 possa, pelo menos, providenciar um nível mínimo de segurança contra as muitas ameaças a que a Internet está sujeita. Por isso alguns aspectos básicos foram levados em conta como a integridade a confidencialidade e a autenticação. Como já dissemos anteriormente, o IPv4 quando foi desenhado não levou em conta as ameaças que iriam surgir. Por isso surgiram algumas falhas de segurança, por exemplo com alguns protocolos usados. Protocolos esses que "confiam" a sua segurança da autenticação dos endereços IP e das portas. No entanto não sabemos se determinado pacote IP foi modificado ou lido o que leva a uma falha de segurança. Além de que muitos protocolos que requerem autenticação enviam a password em puro texto que facilmente poderá ser descodificado. Estes são alguns dos problemas que são colmatados através de Filtros para pacotes, Firewalls, SSL, PEM, PGP, etc. Como é o IPv6 é mais seguro? Graças à Framework IPSec, que é uma Framework de segurança para a camada onde trabalha o IP. Esta Framework consiste em seis elementos distintos: requerimentos de segurança na



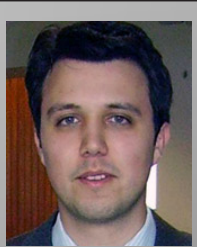


camada network, encriptação (ESP), autenticação (AH), utilização de algoritmos criptográficos para encriptação e autenticação, definição de políticas de segurança e gestão da chave IPSec. Existem diversos usos que podemos dar á IPSEC, confidencialidade nas transmissões, autenticação dos peer para os updates de routing, server lookups e auto-configuração em DHCP e também a prevenção de ataques Denial-of-Service. O IPSEC também exerce a sua influência em outros serviços e aplicações das diversas camadas, mas nem todo é um mar de rosas porque existem algumas áreas onde ainda ocorrem alguns conflitos, como por exemplo em NAT, QoS e Tunneling. Mas também ainda existem falhas no IPv6 apesar de se considerar o IPSEC estável, mas muito trabalho ainda se tem de fazer em diversas áreas de maneira a poder usufruir ao máximo aquilo que o IPv6 tem para oferecer.

Conclusão

Neste pequeno artigo foram abordados diversos aspectos relacionados com o IPv6, mas tudo isto que analisámos apenas é um pequeno resumo de toda a informação disponível. Para poderem aprofundar melhor este assunto recomendo que leiam IPv6 Essentials O Reilly e o Chapter 13 - Internet Protocol Version 6 (IPv6) do Sybex CCNA Cisco Certified Network Associate Study Guide Exam 640-802 6th Edition. Podem também consultar o site http://www.pt.ipv6tf.org/ .

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

USACO Training Pages

Site de treino das olimpíadas de informática dos EUA, recomendado a qualquer programador com interesse em testar/treinar as suas capacidades em algoritimia. Com background teórico e novos problemas desbloqueáveis à medida que se avança, o site inclui ainda um grader que nos permite saber em que testes o programa submetido errou, entre outras informações.

http://train.usaco.org/

Android

Site do projecto Android da Open Handset Alliance, uma plataforma opensource para telemóveis em que o Google participa, onde poderá encontrar o SDK e boa documentação sobre o Android.

http://code.google.com/android/

Free Software Foundation Licenses

Uma página com licenças reconhecidas pela FSF que permitem a distribuição de software e outro tipo de conteúdos de forma free as in speech. Muito útil se planeia lançar software opensource e quer saber qual a que melhor se adequa a si e ao público-alvo do seu programa/biblioteca.

http://www.fsf.org/licensing/licenses/

PHP Classes Repository

Este site é um dos maiores repositórios de código PHP, nele pode encontrar milhares de classes que o vão ajudar no desenvolvimento das suas aplicações.

http://www.phpclasses.org/

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