PROGRAMAR

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

EDIÇÃO #43 - DEZEMBRO 2013

ISSN 1647-0710

A PROGRAMAR

LISTAS DUPLAMENTE LIGADAS

EXPANDE O TEU MERCADO GLOBALIZANDO A TUA APP!

INTRODUÇÃO AO WEB2PY PARTE 2

A FERRAMENTA GENGETOPT

IMPLEMENTAÇÃO DE ÁRVORES DE VANTAGEM

COLUNAS

OS PERIGOS DAS ESTRUTURAS MUTÁVEIS C#

COMUNIDADES

LEITOR DE QRCODE PARA WINDOWS PHONE NETPONTO

ANÁLISES

TÉCNOLOGIAS DE PROGRAMAÇÃO DE JOGOS

NO CODE

É A "LÍNGUA" DO SÉCULO XXI PROGRAMAR

EDITORIAL

EQUIPA PROGRAMAR

Coordenador
António Pedro Cunha Santos

Editor
António Pedro Cunha Santos

Design
Sérgio Alves
Twitter: @scorpion_blood

Redacção
António Pedro Cunha Santos
António Pedro Cunha
Bruno Almeida
Jorge Paulino
Nuno Santos
Paulo Morgado
Rita Peres
Sara Santos
Sara Silva

Staff
António Pedro Cunha
António Santos
António Silva
Jorge Paulino
Sara Santos

Contacto
revistaprogramar@portugal-a-programar.org

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

ISSN
1 647-071 0

«Out of sync»

Fora de Sincronismo… Com o Natal quase a chegar e com cheiro "sabor a Natal" já no ar, chegamos a mais uma edição da revista PROGRAMAR.

Como tem vindo a ser "tradição" desde que escrevo os editoriais, escolho como titulo um código de erro e neste caso escolhi um genérico "out of sync", por me parecer o mais adequado nesta altura em que tudo "parece estar fora de sincronismo", desde o clima, até à tecnologia, com tantos lançamentos de novas tecnologias, consolas, updates de software, etc…

"Out of Sync" recordo-me de ver em tantas aplicações… Especialmente agora em que a moda está no uso da cloud para armazenamento e as ligações à internet não têm a "qualidade" que seria desejável! Escolhi este titulo também, porque cada vez mais a tecnologia evolui a uma velocidade que é "complexo" mantermo-nos em "sincronismo". Não é de todo "impossível", mas antes complexo! Infelizmente e pude constatar isso, existem "velocidades e velocidades", o que leva a este "out of sync" cada vez maior… Quer falemos de tecnologia, quer falemos de ensino da tecnologia e divulgação da mesma!

Este mês vi ser editado um livro que me traz grandes memórias… Vi e fiquei "nostálgico" ao escrever este editorial. Fiquei porque me recordo de ler a publicação de programação da altura, em papel, que custava uma "pequena fortuna", para os bolsos dos jovenzinhos entusiastas que as liam e pensei no facto de nós, (equipe da revista PROGRAMAR) trazermos até voz (prezados leitores), bimestralmente uma nova edição, com novos conteúdos, com novos temas, novas tecnologias, com artigos de qualidade, fruto do esforço dos autores que connosco colaboram e a quem deixo aqui o meu sincero obrigado e da boa vontade dos nosso parceiros, que connosco colaboram edição após edição.

E como nesta época o tempo é escaço, o frio muito e o Natal está às portas, não me alongo mais neste editorial, deixando a todos quantos lêem esta nossa mas sobretudo vossa edição os Votos de um Feliz Natal e um prospero 2014!

Re-encontramo-nos na edição de Fevereiro! Até lá...

António Santos

ÍNDICE

TEMA DE CAPA

A PROGRAMAR

COLUNAS

ANÁLISES

COMUNIDADES

NO CODE

EVENTOS

13 de Dezembro	APPY DAY BPI (LX FACTORY)
14 de Dezembro	44ª Reunião Presencial da Comunidade NetPonto em Lisboa (Microsoft Portugal - Auditório)

Para mais informações/eventos: http://bit.ly/PAP_Eventos. Divulga os teus eventos para o email eventos@portugal-a-programar.pt

NOTICIAS

Novo malware pode transmite-se por…colunas e microfones

Tudo o que é tecnologia é hoje "olhado" com desconfiança por parte dos utilizadores….até o ferro de engomar. Nos últimos meses, a (in)segurança é um dos temos mais debatidos na sociedade e são várias as tecnologias/produtos que têm sido alvo de análise/exploração.

No que se refere à segurança informática, o malware continua a ser uma das principais ameaças para os sistemas de informação e há agora informações de um novo conceito de malware que se consegue replicar através som áudio inaudível para o ser humano.

Há umas semanas atrás, Dragos Ruiu, investigador na área da segurança, afirmou publicamente que os computadores do seu laboratório estavam a ser infectados por tipo "estranho" de malware. Segundo Ruiu, este tipo de malware,que foi baptizado com o nome de "badBIOS", é transmitido através de microfones e alto-falantes, podendo infectar facilmente todas as máquinas que se encontrem num mesmo espaço, mesmo que estas não estejam ligadas em rede.

Desde esse momento, muitos são os investigadores que têm devolvido investigação nesta área…e já há resultados.

Uma equipa de investigadores do Fraunhofer Institute for Communication (Alemanha) publicou um novo protótipo de malware no Journal of Communications, que recorre a som inaudível para o ser humano, capaz de "transportar" dados (ex. passwords, etc), não havendo a necessidade dos sistemas estarem ligados a uma infra-estrutura de rede.

Recorrendo a microfones e colunas, os investigadores referem que é possível transmitir passwords e pequenas quantidades de informação numa distancia máxima de 20 metros (isto para a frequência ultra-sónica dos 20.000 Hz) .

A rede "mesh" criada pelos investigadores para avaliar o protótipo é constituída por 5 computadores, que permitem demonstrar que a informação sensível pode facilmente circular entre os nós infectados.

O som inaudível recorre a técnicas utilizadas na transmissão de áudio debaixo de água, e permitiram aos investigadores a comunicação, entre dois computadores Lenovo T400, recorrendo apenas a microfones e alto-falantes. Apesar da pouca largura de banda (cerca de 20 bits/segundo, na frequência ultra-sónica dos 20.000 Hz), os investigadores referem que o canal de comunicação é suficiente para transmitir informação sensível (ex. credenciais).

Apesar de ser (ainda) apenas um conceito, a criação de uma rede acústica discreta para comunicação foi já fundamentada no documento publicado no Journal of Communications. A investigação continua e nos próximos tempos haverá certamente novidades.

Licenciado em LESI criou site que tem 50.000 visitas/dia José Pedro converte qualquer música em acordes

José Pedro Magalhães, ex-aluno da licenciatura em Engenharia de Sistemas e Informática (LESI) da UMinho, criou um software que converte qualquer música (em mp3 ou do YouTube, por exemplo) nos seus acordes. A plataforma Chordify (http://chordify.net), que cofundou para o efeito, é única no mundo por ser online, gratuita, automática e muito intuitiva, atraindo 50.000 visitantes por dia. Atualmente a investigar na Universidade de Oxford, já passou ainda pelo CERN, Philips Research, Microsoft Research Cambridge e Universidade de Utrecht.

O software de José Pedro Magalhães explora os recursos mais sofisticados da linguagem Haskell (e suas extensões), que o investigador começou a desenvolver na UMinho, aprofundou no doutoramento pela Universidade de Utrecht (Holanda) e prossegue na Universidade de Oxford (Inglaterra). "O ensino de Haskell logo nos primeiros anos de Informática na UMinho despertou o meu interesse e, de facto, contribui para a formação sólida em programação e que a indústria reconhece em geral aos alunos desta academia", diz.

Funciona para qualquer canção

A start-up Chordify baseia o modelo de negócio em anúncios não intrusivos e em funções adicionais a baixo custo, como o descarregamento de ficheiros PDF ou MIDI com os acordes transcritos para piano, viola e até ukulele. Curiosamente, as canções mais procuradas são pouco conhecidas: "O serviço funciona para qualquer canção, por mais obscura que seja. E essa é uma grande vantagem, pois outros sites constroem bases de dados de acordes manualmente, tendo apenas os artistas famosos", refere José Pedro Magalhães.

O tratamento "muito correto" da transcrição é outra mais-valia, tornando o sistema ideal para compositores, músicos e não só. De momento, o tema mais popular é "Wherever you are", dos australianos 5 Seconds of Summer. O informático de 29 anos, natural de Vila Nova de Gaia, admite que não esperava "tamanha adesão" da comunidade mundial em tão pouco tempo. "Passámos de 500 visitas por dia para mais de 50.000, ou seja, atingiu-se um milhão e meio de visitas em menos de um ano de existência!", sorri.

Fonte: (Newsletter Universidade do Minho)

TEMA DE CAPA

Interação com voz no Android : Aprenda a desenvolver um aplicativo com interação por voz no Android

Interação com voz no Android : Aprenda a desenvolver um aplicativo com interação por voz no Android

Do que se trata o artigo:

Esse artigo demonstra os passos básicos para desenvolver um aplicativo que utiliza a interação por voz, posterior reconhecimento do que foi dito e com isso realizar uma determinada acção (no nosso exemplo, faremos uma simples mudança na interface).

Em que situação o tema é útil:

Será útil para os programadores que desejem ter uma experiência inicial com a API de reconhecimento de voz.

Desenvolvendo aplicações para Android:

A utilização de smartphones tem crescido no mundo todo, tendo como um dos principais sistemas operativos o Android, o qual conta com diversas aplicações para atrair novos utilizadores. Pensando nisso, a Google disponibilizou o Android SDK, uma plataforma gratuita para o desenvolvimento de aplicações. Este artigo apresenta os passos necessários para o desenvolvimento de uma aplicação que utiliza uma busca por voz, e através dessa busca retorna resultados que serão analisados e uma nova acção será tomada de acordo com essa análise.

Introdução

O SDK é uma sigla de Software Development Kit (Kit de Desenvolvimento de Software), que é normalmente disponibilizado por empresas que desejam que os programadores externos tenham uma melhor integração com o software proposto. Neste artigo utilizaremos o ADK-SDK + Eclipse (IDE) que possui todas as ferramentas necessárias para o desenvolvimento de uma aplicação Android.

Abaixo seguem os passos para o download e instalação do IDE+SDK. Neste artigo vamos mostrar a criação de uma aplicação onde as pronúncias das palavras “guitarra” ou “piano” redirecionarão a um ecrã com a imagem dos respectivos instrumentos musicais. É um exemplo simples, porém com a criatividade pode ser extendido de muitas formas para outras aplicações.

Configurando o Android SDK

O primeiro passo é fazer o download no site oficial que pode ser consultado na seção de referências.

Após a descompactação do arquivo (que deve estar no formato ZIP), basta abrir o Eclipse localizado na pasta com o seu nome.

Criando o projeto

Com o Eclipse aberto, abra o menu “File”/“New” e em seguida selecione a opção “Others”. Escolha a pasta “Android” e selecione a opção “Android Application Project”, conforme mostra a Imagem 1.

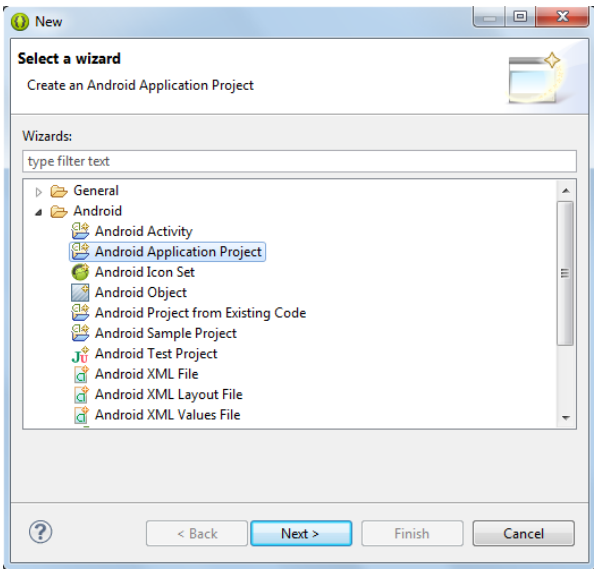


Figura 1– Ecrã inicial de criação de projecto de aplicação Android no Eclipse

Nos próximos passos iremos que dar um nome ao nosso projeto e à nossa aplicação e escolher o seu logo. No ecrã de selecção do tipo de Activity, escolhêmos o padrão BlankActivity que determina uma Activity vazia, essa opção deve ser selecionada pois faremos tudo do “zero”. De seguida definimos o nome da nossa Activity e do seu Layout e então clicamos em “Finish”.

Entendendo a Interface do Eclipse (IDE)

O Eclipse é um ambiente de desenvolvimento que facilita o desenvolvimento de software pela inclusão de inúmeras ferramentas que, dentre outras coisas, economizam tempo ao programador.

Neste artigo vamos comentar apenas os itens que serão utilizados da Interface de desenvolvimento Android fornecida pelo Eclipse. Essa explicação será somente para que o leitor possa se localizar quando citarmos um determinado ponto da interface, o significado de cada um será explicado posteriormente.

No canto esquerdo é possível observar uma hierarquia de pastas, a raiz é obviamente a pasta do seu projeto e todos os arquivos utilizados no aplicativo devem estar dentro dela.

A pasta “source” contém o arquivo “.java” que vai conter todo a parte lógica necessária para o funcionamento do aplicativo.

A pasta “res” contém outra hierarquia de pastas, que, por sua vez, contém todos os arquivos necessários para a definição da interface gráfica do aplicativo. Desde arquivos XML até aos arquivos com as imagens que serão utilizadas.

XML no Layout

XML ou eXtensible Markup Language é uma linguagem de marcação utilizada pelo Android para definir a interface gráfica do aplicativo, permissões e outras configurações do sistema operacional.

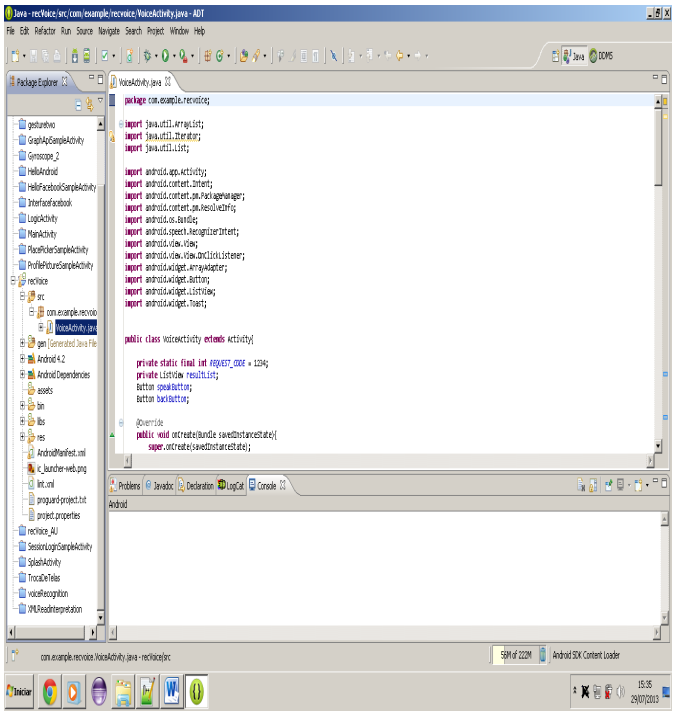


Figura 2- Tela inicial da IDE com código em Java da aplicação.

Utilizando a IDE Eclipse podemos determinar a interface gráfica do aplicativo de duas formas, uma delas é definindo o elemento via XML ou via interface do Eclipse que se assemelha a “montar” visualmente o aplicativo. É importante enfatizar o fato de que cada componente da interface gráfica possui um ID, isto é, possui um identificador único. Esse aspecto é importante, pois durante a programação vamos aceder os elementos da interface gráfica através desse ID. Dessa forma vamos deixar claro para o leitor quando estaremos definindo esse identificador e onde estaremos utilizando-o.

Iniciando cores e nomes

O uso de cores é inevitável para a construção de interfaces, porém precisamos definir as cores antes de utilizá-las. Para isso vamos ao arquivo XML chamado “string.xml”, esse arquivo está dentro da pasta “values” que, por sua vez, está dentro de “res”.

Dentro dele escreva o seguinte código XML:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">recVoice</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<color name="gray_back">#E8E8E8</color>
<color name="blue_back">#4372AA</color>
<color name="font_gray">#344152</color>
 </resources>
```

A primeira linha é padrão. Seguindo abrimos a marcação <resources> onde vão estar a definições das cores e nomes das “strings”. É muito simples entender o significado de cada linha, no que se referem as “string” temos o nome que vamos dar a uma e depois algo que podemos chamar de valor da “string”, por exemplo, a “string” app_name possui valor recVoice. O mesmo segue para as cores, primeiro atribuímos um nome a ela e depois um valor em hexadecimal, pois é dessa forma que um dispositivo electrónico “enxerga” as cores. As cores ficam à escolha do leitor, o que temos acima são somente exemplos que serão utilizados na aplicação .

Iniciando o desenvolvimento da interface

Nessa aplicação teremos 4 telas, logo temos 4 arquivos XML para descrever cada uma. Começando com a tela principal, este arquivo já estará criado na pasta “Layout” que está dentro de “res”. É possível ver essas pastas no canto esquerdo da interface. O nome desse arquivo XML é “activity_main”, porém vamos renomear para “activity_voice” a fim de manter um padrão.

Com o objetivo de deixar o mais claro possível todos os elementos que farão parte dessa tela serão primeiramente descritos, depois o código em XML será mostrado e posteriormente explicações sobre casos mais particulares.

De início temos de definir a estrutura de Layout que vamos utilizar, escolhemos a estrutura “Linear Layout”, ela organiza as componentes dentro da tela de forma que os mesmos sejam distribuídos de forma horizontal ou vertical.

Botões

Anteriormente no arquivo XML “strings” foram definidas algumas cores, para escolher a cor de plano de fundo dessa tela vamos utilizar uma delas, mais especificamente a “gray_back”.

Agora três botões serão adicionados a tela, dois deles serão para redireccionarmos a aplicação para uma tela que contém a imagem de um piano ou de uma guitarra, e o outro será o botão para falarmos e posteriormente realizar a busca.

Analisando cada um dos botões:

O botão que redireciona para a guitarra contém o id “bguitarra”, lembre-se que o id é extremamente importante, pois é através dele que conseguimos acessar um elemento da interface.

Dentro desse botão colocaremos o texto “Guitarra”. De forma semelhante acontecerá com o botão que referencia a uma tela que contém um piano como plano de fundo, esse botão terá o id “bpiano” e colocaremos um texto “Piano”.

O terceiro botão servirá para que o usuário fale a aplicação e a busca seja realizada. Esse botão terá o id “speakButton” e não teremos nenhum texto escrito nele. Nesse botão há algo novo em relação aos outros, foi definido uma imagem como fundo nele. Para fazer isso, antes de tudo temos de obter uma imagem, porém é importante saber onde salvar essa imagem. Dentro de res existem várias pastas cujo primeiro nome é “drawable”, uma cópia do arquivo que contém a imagem deve ser salvo em cada uma dessas pastas. O nome do arquivo imagem que será utilizado nesse botão é “voice_icon”.

List View

Haverá também uma list view que vai ter todos os resultados da busca do que foi dito, ela será muito importante para identificarmos o que foi dito e posteriormente realizarmos uma acção. O id da list view será “list”.

Na próxima seção teremos o código em XML da interface principal do aplicativo, tudo o que foi descrito acima está traduzido nas linhas de código a seguir, porém, para tentarmos reduzir as dúvidas, comentaremos os pontos mais importantes.

XML da interface principal

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://
schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical"
android:background="@color/gray_back">
  <Button android:id="@+id/bguitarra"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Guitarra" />
  <Button android:id="@+id/bpiano"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Piano" />
  <TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_margin="20dp"
  android:text="Clique no botão e fale"
  android:textColor="@color/font_gray"
  android:textSize="18dp" />
  <Button android:id="@+id/speakButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center_horizontal"
  android:background="@drawable/voice_icon"/>
  <ListView android:id="@+id/list"
  android:layout_width="fill_parent"
  android:layout_height="0dip"
  android:layout_weight="1"
 />
```

Primeiramente é possível observar que todas as componentes dessa interface estão associadas a um Layout, nesse caso o “Linear Layout”. É necessário comentar que em uma interface podem ser usados vários tipos de Layout, nessa situação foi utilizado o Linear pelo fato de ser mais simples e intuitivo, mas fica a critério do leitor utilizá-lo ou não. Como opção do Linear podemos definir a sua orientação (horizontal ou vertical), é possível visualizar a escolha na seguinte linha de código XML:

```
android:orientation="vertical"
```

As definições de id são extremamente fáceis de se notar e extremamente importantes também, os id’s definidos no código XML são meramente figurativos, fica à escolha do leitor utilizá-los ou não, porém deve-se atentar ao fato de que quando quiser aceder esse elemento de imagem é necessário recordar o id que foi dado a ele.

Layout tela de guitarra e piano

Por simplicidade nas telas que terão a imagem da guitarra e do piano definiremos também como “Linear Layout”. As duas interfaces são incrivelmente mais simples que a principal, pois terão somente um botão e uma imagem como plano de fundo.

Semelhante à escolha de uma imagem para o botão de “falar” na tela principal vamos manter uma para o plano de fundo de cada interface. O processo é o mesmo, escolha duas imagens que remetem a uma guitarra e a um piano e mantenha uma cópia de cada em todas as pastas cujo primeiro nome é “drawable”, a escolha da imagem fica a critério do leitor.

Nas interfaces teremos um botão que vai redirecionar a aplicação para a tela principal, esses botões possuem o id: “btback” para a tela da guitarra e “btbackp” para a tela do piano.

XML das telas guitarra e piano

Código XML guitarra

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://
schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@drawable/img_guitarra" >
  <Button
  android:id="@+id/btback"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentRight="true"
  android:layout_alignParentTop="true"
  android:layout_marginTop="32dp"
  android:text="Back" />
</RelativeLayout>
```

Código XML piano

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://
schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@drawable/img_piano">
```

```
  <Button
  android:id="@+id/btbackp"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentRight="true"
  android:layout_alignParentTop="true"
  android:layout_marginTop="14dp"
  android:text="Back" />
</RelativeLayout>
```

Nota-se que em ambos os códigos XML existem uma linha em que selecionamos o fundo da tela, no caso do piano será:

```
android:background="@drawable/img_piano"
```

Nesse ponto é importante lembrar o nome que foi dado à imagem e garantir que uma cópia dela foi armazenada nas pastas "*drawable*".

Desenvolvendo as funcionalidades

Primeiro, vamos importar os pacotes que serão usados no desenvolvimento do nosso projeto. Importe os pacotes presentes na Listagem. Os pacotes referentes ao "RecognizerIntent" e o "OnClickListener" representam as importações necessárias para que o aplicativo possa fazer o reconhecimento do que está sendo falado. Os "imports" são extremamente importantes para o funcionamento da aplicação, caso alguns deles sejam esquecidos é muito provável que o aplicativo não compile e muito menos funcione.

```
package com.example.recvoice;

        import java.util.ArrayList;
        import java.util.Iterator;
        import java.util.List;
        import android.app.Activity;
        import android.content.Intent;
        import android.content.pm.PackageManager;
        import android.content.pm.ResolveInfo;
        import android.media.MediaPlayer;
        import android.os.Bundle;
        import android.speech.RecognizerIntent;
        import android.view.View;
        import android.view.View.OnClickListener;
        import android.widget.ArrayAdapter;
        import android.widget.Button;
        import android.widget.ListView;
        import android.widget.Toast;
        import android.hardware.Sensor;
        import android.hardware.SensorEvent;
        import
              android.hardware.SensorEventListener;
        import android.hardware.SensorManager;
        import android.widget.TextView;
```

Na programação Java para Android toda classe deriva da classe *Activity*, logo, em toda declaração de classe temos de ter um **extends** para a classe *Activity*. Assim como no Java convencional, onde temos as *interfaces* que são classes que possuem somente a declaração do método, sem a sua implementação, em Java para android não é diferente. Como essas classes não possuem a implementação de seus métodos é necessário então fazer isso em outra classe. Para isso utiliza-se o **implements** na declaração da classe que vai implementar todos os métodos da classe *interface* (é obrigatório implementar todos os métodos), tal ferramenta é útil na utilização de polimorfismo, que é um conceito comum no paradigma orientado a objetos.

No caso do aplicativo que esse artigo descreve será feita a implementação da classe *interface* "SensorEventListener". Essa classe é utilizada para receber notificações da classe "SensorManager" quando os valores dos sensores presentes em um dispositivo móvel for alterado (dois exemplos de sensores são o acelerômetro e o giroscópio).

A declaração da classe VoiceActivity que implementará a classe interface SensorEventListener pode ser observada na Listagem:

Listagem 2. Declaração da classe Activity

```
public class VoiceActivity extends Activity imple-
ments OnInitListener{
```

Em seguida declare as variáveis presentes na Listagem 3. Como foram descritos na seção que trata da interface da aplicação termos elementos na "programação" que farão referência a esses elementos, que são as variáveis. Logo temos a variável relativa ao botão que redireciona para interface da guitarra do piano e ao de fala. Assim como a "ListView" que será utilizada para buscarmos a palavra guitarra ou piano.

Foi descrito acima também que a classe SensorEventListener trata as notificações emitidas por um elemento da classe SensorManager, logo faz-se necessário a existência de uma variável que faça referência a essa classe.

Abaixo temos a declaração das variáveis

Listagem 3. Declaração das variáveis.

```
private static final int REQUEST_CODE = 1234;
private ListView resultList;
private String telaAtiva;
private Button speakButton;
private Button backButton;
private Button bguitarra;
private Button bpiano;
private SensorManager sManager;
private int valor = 0;
```

O método onCreate é responsável por criar e inicializar a interface quando o aplicativo é inicializado.

Na **Listagem 4** criamos um objeto da classes SensorManager onde posteriormente chamaremos um método (voice_tela) que será responsável por inicializar e exibir a interface da tela principal. Essa "exibição" poderia ser feita também dentro desse método (e normalmente é), porém por uma questão de lógica e economia de código vamos fazer isso dentro de um outro método.

Listagem 4. Método onCreate

```
public void onCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
telaAtiva = "inicial";
sManager = (SensorManager) getSystemService
                                    (SENSOR_SERVICE);
sManager.registerListener(this,
sManager.getDefaultSensor
                        (Sensor.TYPE_ACCELEROMETER)
                    , sManager.SENSOR_DELAY_NORMAL);
voice_tela();
}
```

Criamos um método para a exibição dos elementos de interface. Nesse método acessaremos cada elemento através de seu id, que, por sua vez, foram definidos no código XML utilizados para montar a interface (finalmente vamos utilizá-lo!), a busca por esses id's será feita através do método "findViewById". É possível observar também que "setamos" a interface construída no arquivo XML, isso é feito através do método "setContentView", esse método possui como argumento o nome do XML onde foi definido a interface.

Na **Listagem 5** está a implementação do método "voice_tela", onde primeiramente carregamos a tela principal do aplicativo, definida pelo arquivo XML "activity_voice". Instancia-se os botões que redirecionam a tela de piano e guitarra("bt_piano" e "bt_guitarra" respectivamente). Logo abaixo atribuímos algumas características visuais aos botões.

Listagem 5

```
setContentView(R.layout.activity_voice);
speakButton = (Button) findViewById
(R.id.speakButton);
bguitarra = (Button) findViewById(R.id.bguitarra);
bpiano = (Button) findViewById
(R.id.bpiano);bpiano.setEnabled(false);
bpiano.setVisibility(0);
bguitarra.setEnabled(false);
bguitarra.setVisibility(0);
     resultList = (ListView);
```

Nesse momento vamos começar o desenvolvimento da parte relativa ao botão para falar para a aplicação, ou seja, é o momento do "RecognizerIntent". Primeiramente vamos criar e inicializar uma variável da classe "PackageManager", essa classe recupera informações de vários tipos sobre os pacotes da aplicação que estão instalados no dispositivo. Uma questão crucial que não pode ser esquecida é que temos de atribuir uma ação quando um botão é pressionado, caso contrário a sua existência não faria sentido, imagine pressionar um botão no aplicativo e em nada o seu comportamento for alterado.

Logo para atribuir uma ação quando um botão é pressionado utilizamos o método "setOnClickListener". Dentro desse método existe outro método chamado "onClick", é dentro dele que a ação realmente acontecerá. Exportando a listagem 6 para uma aplicação prática veremos que se o botão "speakButton" for pressionado então teremos de executar o método startVoiceRecognitionActivity, logo ele deve estar dentro do método onClick(). Nesse caso acessaremos o "PackageManager" para descobrir se a captação de áudio do dispositivo móvel está realmente ativa. Na **Listagem 6** verificamos se os serviços para a captação de voz estão disponíveis, caso não estejam é exibido um "Toast" avisando que o "Recognizer" não está disponível. Conseguimos essa informação através da classe "Package Manager" que possui dados sobre os pacotes instalados atualmente na aplicação. Abaixo é possível observar o evento relacionado ao botão que deve ser pressionado para falar ("speakButton"), está definido que no momento em que ele for pressionado o método "startVoiceRecognitionActivity" será chamado.

Listagem 6.

```
// Desabilitamos o botao caso nao tenha o    //
servico
PackageManager pm = getPackageManager();
List<ResolveInfo> activities =
pm.queryIntentActivities(
       new Intent
       (RecognizerIntent.ACTION_RECOGNIZE_SPEECH),
       0);
       if (activities.size() == 0){
              speakButton.setEnabled(false);
          Toast.makeText(getApplicationContext(),
       "Reconhecedor de voz nao encontrado",
       Toast.LENGTH_LONG).show();
       }
       speakButton.setOnClickListener(new
         OnClickListener() {
              public void onClick(View v) {
                 startVoiceRecognitionActivity();
            }
});
```

Vamos agora para a implementação do método "startVoiceRecognitionActivity". Primeiramente cria-se um objeto da classes Intent, pois é através dela que podemos enviar comandos ao sistema operacional Android para a realização de alguma ação, além de poder enviar e recuperar dados. O método "putExtra" insere na *intent* algum valor, ele possui dois parâmetros, o primeiro é um nome para identificação e o segundo um valor que pode ser de qualquer tipo (string, inteiro float, etc). Utilizamos o "putextra" na aplicação pois vamos atribuir o à *string* EXTRA_LANGUAGE_MODEL o valor LANGUAGE_MODEL_FREE_FORM, esse valor não delimita o reconhecimento a uma linguagem específica. Segue abaixo na **Listagem 7** a implementação do método "startVoiceRecognitionActivity", onde são escolhidos os parâmetros que serão usados na requisição, já que o reconhecimento não é processado pelo dispositivo móvel, mas sim por um servidor do Google.

Listagem 7

```
private void startVoiceRecognitionActivity(){
       Intent intent = new Intent
(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
                  intent.putExtra(RecognizerIntent.
                                  EXTRA_LANGUAGE_MODEL,
       RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
              "AndroidBite Voice Recognition...");
```

```
startActivityForResult(intent, REQUEST_CODE);
}
```

Finalmente chegamos ao momento em que detectamos qual a palavra que foi dita e realizamos uma ação se ela for “guitarra” ou “piano”. O mecanismo de análise é incrivelmente simples, primeiramente o usuário diz algo ao dispositivo móvel, todos os resultados possíveis são armazenados na ArrayList “matches”. Então basta percorrer essa lista e identificar se uma das palavras esperadas encontra-se lá. Nesse caso se encontrarmos a palavra guitarra ou piano redirecionamos a aplicação para uma tela cujo plano de fundo é o respectivo instrumento.

Antes de fazer isso é necessário fazer a verificação se as permissões foram realmente adquiridas, isso através do REQUEST_CODE, feito a verificação e tudo estiver certo, a busca é realizada. Na **Listagem 8** temos a implementação do método onActivityResult, onde todo o mecanismo de busca descrito acima é implementado.

Listagem 8.

```
protected void onActivityResult(int requestCode,
int resultCode, Intent data){
if (requestCode == REQUEST_CODE && resultCode ==
                                        RESULT_OK)
{
ArrayList<String> matches =
                    data.getStringArrayListExtra
(RecognizerIntent.EXTRA_RESULTS);
      resultList.setAdapter(newArrayAdapter<String>
        (this, android.R.layout.simple_list_item_1,
                                        matches));
for(int i = 0; i < matches.size();i++)
{
if(matches.get(i).equalsIgnoreCase("guitarra"))
{
      tela_guit();
        break;
}
if(matches.get(i).equalsIgnoreCase("piano"))
{
      tela_piano();
        break;
}
}
}
super.onActivityResult(requestCode, resultCode,
                                            data);
}
```

Para acessar as telas relativas ao piano e à guitarra usamos os métodos “tela_piano()” e “tela_guit()”, respectivamente. A **Listagem 9** e **10** dizem respeito aos eventos atribuídos aos botões “bguitarra” e “bpiano”. Em ambos os casos temos o mesmo conceito de evento do botão sendo pressionado para a chamada de um método.

Listagem 9.

```
bguitarra.setOnClickListener(new
View.OnClickListener() {
       public void onClick(View v) {
           tela_guit();
           telaAtiva = "guitarra";      }
});
```

Listagem10

```
bpiano.setOnClickListener(new View.OnClickListener
() {
       public void onClick(View v) {
           tela_piano();
           telaAtiva = "piano";
      }
});
```

Seguimos agora para a implementação dos métodos “tela_guit” e “tela_piano”, em ambos o layout definido no arquivo XML explicado anteriormente será acessado e “setado” na tela do dispositivo móvel. Isso é feito através do método “setContentView”, é possível observar que o padrão seguido na tela principal é o mesmo definido pra todas. Dentro das duas telas teremos botões que redirecionarão para a tela principal, pois quando pressionados chamarão o método “voice_tela”, que, por sua vez, redireciona para a tela principal da aplicação. Seguem nas **Listagens 11** e **12** os métodos relativos a “tela_piano” e “tela_guit”, que possuem o mecanismo descrito acima.

Listagem 11

```
public void tela_piano(){
      setContentView(R.layout.itf_piano);
      backButton = (Button) findViewById
                                    (R.id.btbackp);
      backButton.setOnClickListener(new
                        OnClickListener() {
            public void onClick(View v) {
                  voice_tela();
            }
       });
}
```

Listagem 12

```
public void tela_guit(){
       setContentView(R.layout.itf_guitarra);
      backButton = (Button) findViewById
                                     (R.id.btback);
      backButton.setOnClickListener(new
                                OnClickListener() {
            public void onClick(View v) {
            voice_tela();
            }
       });
}
```

Permissões e AndroidManifest.xml

Esse é o arquivo principal do projeto, pois dentro dele estão todas as configurações. A aplicação em questão necessita de conexão com a internet para que funcione, porém essa conexão tem de ser permitida primeiramente. Isso é feito adicionando uma permissão a um arquivo XML chamado “Androidmanifest.xml”.Para adicionarmos a permissão basta adicionar a seguinte linha depois da TAG “manifest”:

```
<uses-permission android:name="android.
permission.INTERNET" />
```

Dessa forma garantiremos que o dispositivo móvel permitirá que o arquivo se conecte com a internet e realiza a busca pela palavra pronunciada.

> “O RecognizerIntent” e o “OnClickListener” representam as importações necessárias para que o aplicativo possa fazer o reconhecimento do que está sendo falado.

Conclusões

Este artigo apresentou as características básicas para se construir um aplicativo que “ouve” o usuário e realiza uma busca pelo que foi dito. Após a análise da voz, uma mudança de tela pode ocorrer, mais especificamente se o usuário disser as palavras guitarra ou piano será redirecionado para uma tela que contém a imagem do instrumento como plano de fundo. Veja que isso é um exemplo ingênuo do que pode ser feito, conseguir reconhecer e atribuir uma ação através da fala abre inúmeras possibilidades de desenvolvimento!

Links

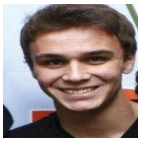
http://developer.android.com/sdk/index.html

Site oficial do Android SDK.

http://developer.android.com/training/index.html

Site com guias para iniciantes no desenvolvimento para Android.

AUTORES



Escrito por Lucas Mandotti Morotti Soares

(morotti.lucas@gmail.com). Bacharelando em Ciência da Computação na Universidade Federal de São Carlos. Faz Iniciação Científica na Universidade Federal de São Carlos na área de desenvolvimento de aplicativos com interação multimodal para Android.

Escrito por Ricardo Roberto Leme

(ricardo.leme@fatec.sp.gov.br). Mestrando em Ciência da Computação pela Universidade Federal de São Carlos - Campus Sorocaba (2014). Atualmente é professor da FATEC - Faculdade de Tecnologia de Itu. Tem experiência na área de Sistemas de Informação, com ênfase em Aplicações de Sistemas de Informação, atuando principalmente nos seguintes temas: IHC- Interface Homem Computador, RIA- (Rich Internet Applications) e aplicações acessíveis em dispositivos móveis.

Escrito por Luciana Aparecida Martinez Zaina

(lzaina@ufscar.br). É Professora Adjunta do curso de Ciência da Computação da UFSCar - campus de Sorocaba. Atualmente, trabalha com o tema de interação multimodal em pequenos dispositivos com foco em design da interação.

Escrito por Ian Ferreira Costa

(fcosta.ian@gmail.com). Bacharelando em Ciência da Computação na Universidade Federal de São Carlos. Faz Iniciação Científica na Universidade Federal de São Carlos na área de desenvolvimento de aplicativos com interação multimodal para Android.

A PROGRAMAR

Introdução ao Web2py Parte 2

A ferramenta gengetopt

Implementação de Árvores de Vantagem

Listas Duplamente Ligadas

Expande o Teu Mercado, Globalizando a Tua App!

Introdução ao Web2py Parte 2

Neste artigo, vou aprofundar um pouco mais a framework web2py, na sua aplicação mais prática, uma vez que se trata de um artigo de continuidade do anterior onde abordei a instalação da framework tanto em ambiente Windows como em ambiente GNU/Linux, e a construção de uma primeira aplicação com acesso a base de dados, utilizando a DAL fornecida pelo web2py.

Uma vez que estamos a falar num artigo de continuação, vou tentar debruçar-me mais sobre a construção de aplicações em web2py e menos sobre alguns aspectos da linguagem.

A framework web2py tem como base a linguagem Python no desenvolvimento dos models, controllers, views, embora utilize uma sintaxe de Python ligeiramente modificada nas views de forma a tornar o código mais legível, sem impor quaisquer restrições sobre o uso correcto Python.

O propósito das views é "embutir" código Python, em documentos HTML. Esta tarefa por norma é bastante árdua e complexa, além de apresentar problemas complexos de implementação.

Um dos problemas que provavelmente já vos está a passar pelas ideias é "como é que vai ser separado o código em linguagem Python do código HTML ?". A resposta é bastante simples, em web2py utilizam-se por default os caracteres de chaveta (dupla chaveta) {{ ... }} para delimitar o código da view que se encontra escrito em linguagem Python. A vantagem de utilizar a dupla chaveta, reside no facto de as chavetas serem ignoradas pela maioria dos editores de HTML, permitindo assim a nós programadores podermos usar o editor que mais gostamos sem qualquer dificuldade. No entanto estes caracteres podem ser alterados, apesar de isto ser altamente desaconselhado, por poder causar problemas não só com os editores mas com alguns browsers.

Podem-se alterar os caracteres delimitadores de código Python nas views das aplicações feitas utilizando web2py com o seguinte código:

```
response.delimiters = ('<?','?>')
```

Esta alteração tem o seu âmbito definido conforme o local onde este código é colocado. Por exemplo se for colocado num controller, apenas será aplicado às views desse controller, se for aplicado dentro do código de uma acção, será apenas aplicado à acção em causa, mas se por exemplo for colocado num model, será aplicado a toda a aplicação. Esta é uma das razões pela qual é altamente desaconselhada esta alteração, pois um erro onde se coloca, e o âmbito (scope), fica completamente alterado, podendo originar diversos erros na aplicação e levar o programador a ter de rever imenso código.

Adicionalmente em linguagem Python usa-se uma indentação padrão de 4 espaços, que neste caso não podemos utilizar. Ora para contornar essa questão usamos a palavra reservada pass da linguagem Python, no fim de cada bloco, para que o interpretador faça a indentação.

A Python Template Language, encarrega-se de fazer a indentação do código de forma a produzir código que o interpretador possa executar.

Ora vejamos o seguinte exemplo:

```
<html>
<head> </head>
<body>
{{for x in range(10):}}{{=x}} Olá</br> {{pass}}
</body>
</html>
```

Seria traduzido para:

```
response.write("""<html><body>""", escape=False)
for x in range(10):
response.write(x)
response.write("""hello<br />""", escape=False)
response.write("""</body></html>""",
                                    escape=False)
```

Quando ocorre um erro numa view com código Python o código que é visível no ticket e o respectivo report, é o código que é gerado pela view e não o código que foi efectivamente escrito pelo programador. Isto tem a vantagem de facilitar o debug, mostrando o código que é efectivamente executado e que pode ser editado directamente num editor de HTML, ou num DOM Inspector do browser, como o Firebug (do browser Firefox).

Devemos ter em atenção ainda a um pormenor de extrema importância. Quando passamos variáveis, por exemplo:

```
{{=x}}
```

Gera como resultado:

```
response.write(x)
```

As variáveis injectadas no código HTML, são "limpas" (escaped), por default, de forma a servir como "barreira" contra ataques do tipo XSS Injection. Vejamos um exemplo:

```
{{=H1(i)}}
```

É interpretado e traduzido da seguinte forma:

```
response.write(H1(i))
```

A quando da interpretação do object H1, ele é "serializado", "limpo", e por fim escrito dentro da tag <body> do código HTML que será interpretado pelo browser em cliente-side.

O método response.write(obj, escape=True), recebe dois argumentos, primeiramente o objecto a ser escrito e de seguida se ele tem de ser "limpo" (escaped), antes de ser escrito. Se o objecto contiver um método .XML(), ele é chamado e o seu resultado é escrito nos devidos lugares dentro do html body; nos restantes casos é usado o método __str__ para serializar todos os valores retornados, com a excepção dos objectos já embutidos na framework web2py, que são serializados pela própria framework, no exemplo acima o H1 é serializado como <H1></H1>, pois é um objecto que integra a framework.

Feitas estas considerações gerais sobre web2py, vamos experimentar fazer algumas coisas engraçadas numa view, de uma aplicação, começando por usar ciclos ou "loops".

Para quem vem de outras linguagens que esteja habituado a outra sintaxe do ciclo for each, ele aqui é idêntico e muito útil para iterar, um objecto, por exemplo:

```
{{revistas = ['edicao 1', ' edicao 2', 'edicao 3',
                                        'edição 4']}}
<ul>
{{for revista in revistas:}}<li>{{=revista}}</li>
{{pass}}
</ul>
```

Aqui temos um ciclo for que irá iterar um array chamado revistas que contem quatro strings. O código HTML resultante disto seria:

```
<ul>
<li>edicao 1</li>
<li>edicao 2</li>
<li>edicao 3</li>
<li>edicao 4</li>
</ul>
```

Apenas tive o cuidado de criar o array na view, para ter a certeza que era um objecto iterável, como tantos outros objectos disponíveis na linguagem python.

De notar que todos os valores foram "limpos" (escaped), antes de serem passados para HTML, novamente aqui é "erguida" uma barreira contra XSS Injection.

Vejamos agora o ciclo While, igualmente útil:

```
{{r = 5}}
<ul>
{{while r > 0:}}<li>{{=r}}{{r = r - 1}}</li>
{{pass}}
</ul>
```

Neste caso implementamos um contador descendente, com o while em que o ciclo é executado equanto r for maior que zero, usamos a palavra reservada pass para fechar o ciclo.

O resultado seria o seguinte:

```
<ul>
<li>5</li>
<li>4</li>
<li>3</li>
<li>2</li>
<li>1</li>
</ul>
```

Estes são os principais ciclos usados em web2py, para quem programa habitualmente PHP ou ASP, esta estrutura não será de todo estranha.

Além de ciclos e como o web2py foi feito para ter grande flexibilidade, temos mais algumas funcionalidades que podemos usar mesmo nas views, como por exemplo, instruções condicionais.

Abaixo apresenta-se um exemplo de if com else if e else.

```
{{
import random
r = random.randint(1, 43)
}}
<h2>
{{=r}}
{{if r % 2:}}r é impar{{else:}}é par{{pass}}
</h2>
```

Neste caso dependendo do número seleccionado entre um e quarenta e três, será mostrado o número, seguido da indicação de ser par ou ímpar. Quarenta e três é o numero de edições até ao momento publicadas da revista programar, por isso a escolha do número foi essa, mas testem com os números que quiserem.

Como podem observar só se usa a palavra reservada pass para fechar o ciclo após o else, neste caso não existe um else if (elif), que será demonstrado no exemplo seguinte:

```
{{
import random
k = random.randint(1, 43)
}}

<h2>
{{=r}}
{{if r % 6 == 0:}}é divisivel por 6 (o numero de
edições que publicamos num ano)
{{elif k % 2 == 0:}}é par
{{else:}}é impar
{{pass}}

</h2>
```

Neste caso usamos um elif para permitir uma segunda condição dentro da condição principal. Podemos encadear quantas condições quisermos, aconselha-se o uso de bom senso, mas não existe um limite definido de condições encadeadas que podem ser usadas numa view.

INTRODUÇÃO AO WEB2PY PARTE 2

Da mesma forma que nos exemplos anteriores existem outras estruturas possíveis de ser usadas, incluindo estruturas como try … excepto … else … finaly .

Vamos ver o exemplo com try … except:

```
{{try:}}
Erro {{= 1 / 0}}
{{except:}}
divisão por zero
{{else:}}
divisão não por zero
{{finally}}
<br />
{{pass}}
```

Isto resultaria em:

```
Erro
Divisão por zero.
<br />
```

> ***(…) construída tendo como ponto forte a segurança, isto é, aborda automaticamente muitas das questões que podem levar a vulnerabilidades, seguindo práticas bem estabelecidas.(…)***

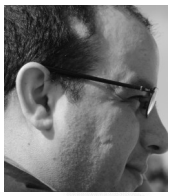
Como vimos ao longo dos exemplos existem diversas estruturas de controlo que podem ser aplicadas nas views estendendo assim a sua "funcionalidade mais elementar".

> ***Um dos problemas que provavelmente já vos está a passar pelas ideias é "como é que vai ser separado o código em linguagem Python do código HTML ?".***

Não foram apresentadas todas, pois o artigo seria ainda mais extenso e aborrecido, para quem lê, pois tanto tempo a ver estruturas de controlo e estruturas de dados torna-se maçador.

No próximo artigo, irei continuar a abordar as views mais concretamente os HTML Helpers do Web2py.

AUTOR



Escrito por António Santos

Entusiasta da tecnologia desde tenra idade, cresceu com o ZX Spectrum, autodidacta, com uma enorme paixão por tecnologia, tem vasta experiência em implementação e integração de sistemas ERP, CRM, ERM. Membro da Comunidade Portugal-a-Programar desde Agosto de 2007, é também membro da Sahana Software Foundation, onde é Programador Voluntário. Neste momento é aluno no Instituto Politécnico de Viana do Castelo, na Escola Superior de Tecnologia e Gestão.

.

A ferramenta gengetopt

Os parâmetros de linha de comando constituem um poderoso mecanismo de interação, especialmente nas aplicações de modo consola/terminal que funcionam em modo de texto. De facto, através dos parâmetros da linha de comando, torna-se possível especificar múltiplos comportamentos e funcionalidades para uma mesma aplicação de modo consola/terminal, estando cada comportamento associado a um dado conjunto de parâmetros passado pela linha de comando. Por exemplo, o bem conhecido comando **ls**, cuja função primária no UNIX é a listar nomes de ficheiros e diretórios, disponibiliza na versão GNU mais de 50 parâmetros da linha de comando, desde o **-a** (mostra todos os ficheiros) ao **-x** (mostra a listagem por linhas). Curiosamente, existe ainda a opção **-X** (maiúscula) que lista os ficheiros por ordem alfabética das respetivas extensões.

Este artigo mostra como se usa e quais as mais-valias da ferramenta **gengetopt** para o tratamento semi-automatizado das opções passadas pela linha de comando em programas escritos em linguagem C que se destinam a ambientes linux.

Acesso aos parâmetros da linha de comando na linguagem C

Na linguagem C, os parâmetros da linha de comando estão acessíveis através de dois parâmetros disponíveis na função **main**: o vetor de strings **argv** e o inteiro **argc**. O vetor **argv** contém uma string para cada conjunto de carateres passados pela linha de comando, em que argv[0] representa o nome do programa em execução, argv[1] corresponde ao primeiro parâmetro passado linha de comando e assim sucessivamente. Por sua vez, o inteiro **argc** indica o número de strings existentes no vetor **argv**. O código da Listagem 1 exemplifica a iteração do vetor argv. A Listagem 2 apresenta o resultado da execução do código quando o programa é chamado da seguinte forma: **./mostraArgv um dois --tres**

Conforme indicado anteriormente, o primeiro elemento do vetor argv (**argv[0]**) corresponde sempre ao nome do programa que está a ser executado, como se denota na Listagem 2 (**./mostraArgv**).

```
int main(int argc, char *argv[])
{
  int i;
  for (i = 0; i < argc ; i++)
  {
    printf("argv[%d]='%s'\n", i, argv[i]);
  }
  return 0;
}
```

Listagem 1: iteração do vetor argv (mostraArgv.c)

```
./mostraArgv um dois --tres
argv[0] = './mostraArgv'
argv[1] = 'um'
argv[2] = 'dois'
argv[3] = '--tres'
```

Listagem 2: exemplo da execução do programa mostraArgv

Opções em formato curto e opções em formato longo

Em ambiente UNIX, é comum as aplicações de modo consola disponibilizarem as opções da linha de comando em dois formatos: formato curto e formato longo. Uma opção curta está limitada a uma letra precedida de um sinal menos (e.g., **-a**). Por sua vez, uma opção longa é composta por uma palavra precedida por dois sinais menos (por exemplo, **--all**). O formato longo é obviamente mais explícito. É frequente ainda uma opção ser disponibilizada no formato curto e no formato longo. Exemplo disso é a opção **-a** e **--all** do comando **ls**. Uma opção pode requerer elementos adicionais, como, por exemplo, o nome de um ficheiro.

Para além das opções e dos respetivos parâmetros, uma aplicação pode ainda solicitar argumentos livres, isto é, que não estejam diretamente associados a opções. É o caso do comando **ls** que pode receber uma lista com os nomes dos ficheiros a serem processados (**ls -l a.txt b.txt c.txt**).

Processamento manual/explícitos dos parâmetros da linha de comando

Para o programador da aplicação, o processamento dos parâmetros da linha de comando suportados por uma aplicação é frequentemente uma tarefa fastidiosa. De facto, o processamento manual dos parâmetros da linha de comando requer que seja implementado código para a validação das opções. Esse código deve incidir sobre a deteção de opções inválidas, seja porque não existem, seja porque falta um parâmetro complementar à opção, por exemplo, o nome de ficheiro sobre o qual se aplica a opção. Adicionalmente, quando são acrescentadas funcionalidades em novas versões da aplicação, é frequente o aumento do número de opções, requerendo atualizações e acréscimos ao código que trata do processamento das opções da linha de comando e dos respetivos parâmetros.

A Listagem 3 apresenta código rudimentar para o tratamento manual de opções da linha de comando. Concretamente, o código processa as opções **-a**, **--all** (equivalente à opção **-a**), --**help** e **-f**. A opção **-f** requer um nome de ficheiro como parâmetro adicional. A opção --**help**, que existe na maioria das aplicações, apresenta ajuda sucinta sobre a aplicação, nomeadamente sobre as opções da linha de comandos que disponibiliza. Apesar de lidar somente com quatro opções, e apenas uma delas ter um parâmetro adicional obrigatório, o

código mostrado na Listagem 3 tem um tamanho apreciável e uma relativa complexidade devido às várias estruturas de controlo **if**. Importa ainda notar que o código apresentado não atende a todas as situações, pois não deteta a múltipla indicação de uma opção (e.g., **-a -a -a** ou **-a** --**all**), ou a indicação de uma opção em lugar do nome do ficheiro para a opção **-f** (e.g., **-a -f --all**).

O código não considera ainda a possibilidade de uma ou mais opções poderem ser obrigatórias, no sentido que a aplicação só deve prosseguir a execução caso as opções definidas como obrigatórias tenham sido indicadas pelo utilizador. Por fim, o acréscimo de mais opções à aplicação requer uma substancial extensão do código, diminuindo a legibilidade do mesmo e aumentando os custos de manutenção. Por exemplo, o acrescentar de uma nova opção requer não só escrever o código no ciclo **while** para tratamento da opção, mas também a atualização da função **Help**.

Processamento assistido das opções da linha de comando - as funções *getopt*, *getopt_long* e *argp*

Sendo o processamento dos parâmetros da linha de comando uma tarefa relativamente entediante, não é de estranhar que tenham surgido várias metodologias com o objetivo de reduzir o volume de código e trabalho do programador. Umas dessas metodologias assenta no uso da API (*Application Programming Interface*) **getopt** [getoptOnline]. Essa API assenta no uso da função **getopt** que recebendo uma string especialmente formatada com as possíveis opções curtas a suportar, devolve as opções que foram indicadas pelo utilizador. Por exemplo, para suportar as opções **-a** e **-f**, especifica-se a string "**af:**", com os dois pontos a indicarem que a opção **-f** requer obrigatoriamente um parâmetro adicional. Um exemplo do uso da função getopt pode ser encontrado em [getoptExemplo]. Para suprir as limitações da função **getopt**, existe a função **getopt_long** que suporta o processamento de opções curtas e longas, mas a custo de uma acrescida complexidade como se denota pelo protótipo da função mostrado na Listagem 4.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/** Programa que processa (sumariamente) as op-
ções:
 *  -a/-all, --help,  -f <ficheiro>
 */
void Help(const char *msg){
  if( msg != NULL ){
    fprintf(stderr,"[ERRO]: %s\n",msg);
  }
  printf("Opções\n");
  printf("-a / -all   processa tudo\n");
  printf("-f nome     processa nomeficheiro\n");
  printf("--help      exibe ajuda sucinta\n");
}

int main(int argc, char *argv[]){
  int i;
  char Err_S[256];
  i=1;    /* argv[0] --> nome do programa */
  while (i < argc){
    if (strcmp(argv[i],"-a") == 0) {
      printf("opção -a\n");
    }
    else if (strcmp(argv[i], "--all") == 0) {
      printf("opção --all\n");
    }
    else if(strcmp(argv[i], "-f") == 0) {
      /* opção -f requer ficheiro */
      i++;
      if (i == argc) {
        Help("Falta fich opção -f");
        exit(EXIT_FAILURE);
      }
      else {
        printf("-f fich='%s'\n", argv[i]);
      }
    }
    else if (strcmp(argv[i], "--help") == 0) {
      printf("Opção --help\n");
      Help(NULL);
      exit(EXIT_SUCCESS);
    }
    else {
      sprintf(Err_S,"Erro '%s'",argv[i]);
      Help(Err_S);
      exit(EXIT_FAILURE);
    }
    i++;
  }/* while */
  return 0;
}
```

Listagem 3: processamento rudimentar de opções

```
int getopt (int argc, char *const *argv,
            const char *options);

int getopt_long (int argc, char *const *argv,
                 const char *shortopts,
                 const struct option *longopts,
                 int *indexptr);

error_t argp_parse (const struct argp *argp,
                    int argc, char **argv,
                    unsigned flags, int *arg_index,
```

Listagem 4: protótipos das funções getopt, getopt_long e argp

A função **argp** é uma outra forma de automatizar algum do processamento dos parâmetros da linha de comando, similar ao **getopt** e ao **getopt_long**, embora com mais alguma funcionalidade. Ao leitor interessado recomenda-se a consulta de [argp2010].

Processamento automatizado dos parâmetros da linha de comando: o gerador de código gengetopt

O maior entrave ao uso das funções **getopt** e **getopt_long** deve-se à sua relativa complexidade e à necessidade de se recorrer a um formato pouco amigável para a especificação dos parâmetros a tratar. O objetivo da ferramenta **gengetopt** (***generate getopt***) é precisamente libertar o programador da interação direta com as pouco amigáveis funções da família **getopt** [gengetoptOnline]. Para o efeito, o **gengetopt** constroi, a partir de um simples ficheiro de configuração fornecido pelo programador, o código em linguagem C para interagir com o **getopt/getopt_long**. Através do ficheiro de configuração, o programador expressa, no formato próprio e relativa-

mente simples do **gengetopt**, os parâmetros da linha de comando reconhecidos que devem ser aceites como válidos pela aplicação. Quando executado com o ficheiro de configuração, o **gengetopt** cria um ficheiro de código C e o respetivo ficheiro **.h** que devem ser integrados no projeto da aplicação. Na sua utilização mais simples e mais usual de validação dos parâmetros da linha de comando, o programador apenas tem que chamar a função **cmdline_parser** criada pelo **gengetopt**. Para além de proceder à validação das opções da linha de comando, a função **cmdline_parser** preenche a estrutura do tipo **gengetopt_args_info**, estrutura criada pelo **gengetopt** e cujos campos refletem a configuração expressa pelo programador.

Exemplo basico.ggo

A listagem do ficheiro de texto **basico.ggo** (Listagem 5) exibe um exemplo básico de ficheiro de configuração **gengetopt**. Embora não seja obrigatório, é usual o nome de um ficheiro de configuração **gengetopt** ter a extensão **.ggo**. Na primeira parte do ficheiro são descritos o pacote de software (***package***), a versão (***version***), o propósito do software (***purpose***) e ainda uma descrição do software (***description***). As opções propriamente ditas são identificadas através da palavra-chave **option**. Concretamente, cada opção curta/longa é descrita por uma linha iniciada por **option**. Finalmente, o símbolo **#** serve para iniciar um comentário que se prolonga até ao fim da linha.

No exemplo basico.ggo são definidas as opções **-a/--all** e a opção **-f/--ficheiro** através das respetivas linhas option. Para o efeito, e após a palavra-chave **option**, cada linha identifica o nome da opção longa entre aspas (uma opção longa não pode ter espaços no nome, embora admita o símbolo _), seguido da letra empregue para a opção curta ou de um sinal menos (-) que indica que a opção não tem versão curta. Segue-se a descrição da opção, através de texto delimitado por aspas. Esta descrição é empregue pelo gengetopt para gerar o texto de ajuda sucinta disponibilizado através da opção --**help**. A configuração que segue a descrição da opção depende do tipo de opção. No caso do par de opção **-a**/--**all** é apenas especificado que se trata de uma opção optional, isto é, não obrigatória. Por omissão, uma opção é considerada obrigatória (*required*) pelo **gengetopt**, isto é, se a opção não for especificada na execução da aplicação, a execução da aplicação termina com uma mensagem de erro a indicar a falta da opção obrigatória.

A opção **-f**/--**ficheiro** é um pouco mais elaborada, dado que requer que seja acrescentado, logo a seguir à opção, uma string correspondente a um nome de ficheiro. Essa informação é transmitida na configuração do **gengetopt** através da indicação **string** (tipo de elemento a acrescentar). Por sua vez, a especificação **typestr="filename"** serve para indicar que o valor a especificar após a opção se destina a ser interpretado pela aplicação como o nome de um ficheiro. Importa notar que o **gengetopt** não produz código para a validação do nome indicado como nome de ficheiro, sendo a especificação **typestr="filename"** somente empregue para a documentação acessível através da opção **--help**.

```
package "gengetopt_basico"
version "1.0"
purpose "Exemplificar gengetopt"
description "Exemplo básico do gengetopt"

# Definição das opções
option "all" a "opção -a / -all" optional
option "ficheiro" f "requer nome de ficheiro"
typestr="filename" string optional
```

Listagem 5: ficheiro de configuração basico.ggo

Execução do gengetopt

Se o **gengetopt** não estiver disponível no sistema, é expectável que possa ser facilmente instalado através do sistema de gestão de software da distribuição de linux em uso, sobretudo no caso de se estar a usar uma distribuição popular de linux (Ubuntu, Fedora, etc.). Por exemplo, numa distribuição **Ubuntu**, a instalação é efetuada através da execução de **sudo apt-get install gengetopt**.

Finalizado o ficheiro de configuração, executa-se o gengetopt da seguinte forma: **gengetopt < basico.ggo**. A necessidade de se especificar o ficheiro de configuração **.ggo** através do redirecionamento de entrada (**< basico.ggo**) deve-se ao facto do **gengetopt**, por omissão, processar a entrada padrão (**stdin**). Caso se pretenda evitar o redirecionamento de entrada, deve-se especificar na execução do gengetopt a opção **-i** (ou a versão longa --**input**) seguida do nome do ficheiro de configuração a processar (**gengetopt -i basico.ggo** ou **gengetopt --input=basico.ggo**).

A execução do **gengetopt** leva à criação de dois ficheiros de código fonte: **cmdline.c** e **cmdline.h**. Estes ficheiros contêm o código (ficheiro **cmdline.c**), as estruturas e os protótipos (ficheiro **cmdline.h**) criadas pelo **gengetopt** para processamento das opções da linha de comando suportadas pela aplicação. Através da opção **-F nome** (ou --**file-name nome**) é possível indicar ao **gengetopt nome** como alternativa a **cmdline** para a criação dos ficheiros de código fonte. Por exemplo, executando-se **gengetopt -i basico.ggo -F params**, o **gengetopt** cria os ficheiro **params.c** e **params.h** em lugar dos ficheiros **cmdline.c** e **cmdline.h**.

Utilização do código gerado pelo gengetopt

Para integrar a funcionalidade de processamento das opções especificadas na linha de comando, o programador da aplicação deve chamar a função **cmdline_parser** que foi criada pelo **gengetopt**. Esta função recebe como parâmetros, para além de **argc** e **argv**, o endereço de memória de uma estrutura do tipo **struct gengetopt_args_info**. O protótipo da função **cmdline_parser** e a declaração da estrutura **gengetopt_args_info** são mostrados na Listagem 6.

```
struct gengetopt_args_info{
  const char *help_help;
  const char *version_help;
  const char *all_help;
  char *ficheiro_arg;
  char *ficheiro_orig;
  const char *ficheiro_help;
```

A FERRAMENTA GENGETOPT

```
  unsigned int help_given;
  unsigned int version_given;
  unsigned int all_given;
  unsigned int ficheiro_given;
} ;

int cmdline_parser (int argc, char **argv,
     struct gengetopt_args_info *args_info);
```

Listagem 6: estrutura struct gengetopt_args_info e protótipo da função cmdline_parser

O código da função main (Listagem 7) limita-se a chamar a função **cmdline_parser**, sem ter em consideração os valores da **struct gengetopt_args_info** que são preenchidos pela função. Com essa utilização simples, o programador está somente a fazer uso da validação de parâmetros. A compilação da aplicação, requer que sejam primeiro criados os ficheiros de código objeto **main1.o** e **cmdline.o** a partir dos respetivos ficheiros de código fonte C, sendo os ficheiros de código objetos posteriormente *linkados* para a criação do ficheiro executável. A execução do **gengetopt**, bem como os passos de compilação e ligação necessários à criação do executável (**main1.exe**) da aplicação encontram-se na Listagem 8.

```
#include <stdio.h>
#include <stdlib.h>
#include "cmdline.h"

int main(int argc, char *argv[]) {
  struct gengetopt_args_info ArgsInfo;
  if (cmdline_parser(argc, argv, &ArgsInfo)) {
    fprintf(stderr,"Erro: execução de
                                cmdline_parser\n");
    exit(EXIT_FAILURE);
  }
  return 0;
}
```

Listagem 7: main1.c

```
gengetopt < basico.ggo
gcc -Wall -W -c cmdline.c
gcc -Wall -W -c main1.c
gcc -Wall -W cmdline.o main1.o -o main1.exe
```

Listagem 8: execução do gengetopt e criação do ficheiro executável main1.exe

Apesar de ainda não ter nenhuma funcionalidade implementada pelo programador, a aplicação **main1.exe** já deteta opções inválidas, isto é, opções não especificadas no ficheiro de configuração. Por exemplo, a execução **./main1.exe -j** leva ao término abrupto do programa com uma mensagem de erro a indicar que a opção **-j** é inválida. Igualmente, caso seja especificada a opção **-f/--ficheiro** sem que seja seguida por uma string (que seria interpretada como nome de ficheiro), a aplicação termina reportando que a opção **-f/--ficheiro** requer um argumento. A aplicação suporta ainda as opções **-h/--help** e **-V/--version**. Estas opções são automaticamente criadas pelo **gengetopt** a partir da informação fornecida no ficheiro de configuração **.ggo**. A saída produzida pela opção **--help** é mostrada na Listagem 9.

```
./main1.exe --help
RevistaProgramar 1.0
Exemplificar gengetopt
Usage: RevistaProgramar [OPTIONS]...
Exemplo básico do gengetopt
  -h, --help                Print help and exit
  -V, --version             Print version and exit
  -a, --all                 opção -a / -all
  -f, --ficheiro=filename   requer nome de fichei-
ro
```

Listagem 9: saída da execução de ./main1.exe --help

Utilização dos parâmetros passados pela linha de comando

Obviamente, o uso do **gengetopt** não se esgota na validação de parâmetros. O programador necessita de ter acesso a informação que permita determinar quais foram as opções indicadas na linha de comando, e no caso de opções com parâmetros, qual o valor dos parâmetros. Essa informação é disponibilizada pelos vários campos da **struct gengetopt_args_info** (Listagem 6). De facto, para cada opção especificada no ficheiro de configuração **.ggo**, o **gengetopt** cria um conjunto de campos na **struct gengetopt_args_info**. Por exemplo, para a opção **--all**, existem os campos **unsigned int all_given** e **const char *all_help**. O primeiro apresenta um valor inteiro positivo caso a opção **-a/--all** tenha sido indicada na linha de comando, ao passo que o campo **all_help** contém o texto de ajuda definido para a opção **-a/--all**.

Por sua vez, a opção **-f/--ficheiro**, que requer um argumento string adicional, está representada por quatro campos na **struct gengetopt_args_info**: **unsigned int ficheiro_given, const char *ficheiro_help**, **char *ficheiro_arg** e **char *ficheiro_orig**. Os dois primeiros campos tem funcionalidade análoga aos anteriormente vistos para a opção **-a/--all**. O campo **char *ficheiro_arg** representa o parâmetro adicional requerido pela opção, ao passo que o campo **char *ficheiro_orig** representa a string que foi passada na linha de comando. Neste caso, e por via do parâmetro adicional para o par de opções **-f/--ficheiro** ter sido declarado como string, os campos **ficheiro_arg** e **ficheiro_orig** tem o mesmo valor.

Através do uso dos campos da estrutura **struct gengetopt_args_info**, o programador pode determinar quais as opções que foram indicadas e levar a aplicação a agir em concordância com o especificado na linha de comando. Assim, para detetar se uma determinada opção foi indicada, a aplicação testa o campo **OPCAO_given**, em que **OPCAO** corresponde ao nome da opção. Caso a opção tenha sido definida como tendo um parâmetro, a aplicação pode aceder ao parâmetro através do campo **OPCAO_arg**. A listagem **main2.c** exemplifica o uso dos campos **OPCAO_given** e **OPCAO_arg**. A aplicação **main2.exe** pode ser compilada da seguinte forma: **gcc -Wall -Wextra cmdline.c main2.c -o main2.exe**

```
#include <stdio.h>
#include <stdlib.h>
#include "cmdline.h"
```

```
int main(int argc, char *argv[]) {
  struct gengetopt_args_info ArgsInfo;
  if (cmdline_parser(argc, argv, &ArgsInfo)) {
    fprintf(stderr,"Erro: execução de
                                cmdline_parser\n");
    exit(EXIT_FAILURE);
  }
  if (ArgsInfo.all_given) {
    printf("-a/-all\n");
  }
  if (ArgsInfo.ficheiro_given)
{
    printf("-f/--ficheiro com '%s'\n",
                           ArgsInfo.ficheiro_arg);
  }
  printf("Finito!\n");
  return 0;
}
```

Listagem 10: main2.c

Explorando as capacidades do gengetopt

O **gengetopt** possui um vasto leque de funcionalidades para além das empregues no exemplo **basico.ggo**. O ficheiro de configuração **funcionalidades.ggo** apresenta mais algumas das funcionalidades do **gengetopt** (Listagem 11).

```
package "RevistaProgramar"
version "2.0"
purpose "Exemplificar gengetopt "
description "Exemplo do gengetopt"
# Definição das opções
option "obrigatoria" o "obrigatoria sem parâmetro"
option "opcional" - "opcional sem parâmetro" optio-
nal
option "flag" - "flag (estado inicial: off)" flag
off
option "preenchido" p "string com valor por omis-
são"                  string default="AAA"
option "argopcional" - "opção com argumento opcio-
nal"
          float argoptional default="3.14"
option "numero" n "requer numero inteiro
(opcional)"
                       long optional
option "multi" m "opção que pode ser especificada
mais do que uma vez" optional multiple(1-3)
option "multi2" - "opção múltipla até 4 vezes com
parâmetro string" string optional multiple(1-4)
option  "enumerada"   - "Uma opção cujo possíveis
valores são especificados por values"
values="zero","um","dois" default="zero" optional
```

Listagem 11: ficheiro de configuração funcionalidades.ggo

De seguida são analisadas algumas das funcionalidades que constam do ficheiro funcionalidades.ggo (Listagem 11).

-o/--obrigatoria: opção obrigatória (por omissão, uma opção é do tipo *required*, isto é, é obrigatória)

--opcional: opção opcional (uso da palavra-chave optional). A opção não tem versão curta

--flag: opção do tipo **flag** que apenas tem dois estados: **on** e **off**. Neste caso está a **off** por omissão. Se for especificado **--flag** na linha de comando, o valor de **flag_arg** da estrutura **struct gengetopt_args_info** passa a **on**

-p/--preenchido: opção obrigatória do tipo string preenchida com o valor por omissão "AAA". O **gengetopt** requer que o valor por omissão deve ser sempre indicado entre aspas.

--argopcional: opção que tem um argumento do tipo double que é opcional, sendo ainda definido o valor por omissão de "3.14".

-n/--numero: opção que requer um número do tipo **long** quando é especificada. O campo **numero_arg** da estrutura **struct gengetopt_args_info** é do tipo **long**. Para além dos tipos de dados **long**, **string** e **double**, o **gengetopt** suporta ainda **int**, **short**, **float**, **longdouble** e **longlong**

-m/--multi: opção que pode ser especificada múltiplas vezes na mesma linha de comando. No exemplo (**multiple(1-3)**), pode ser indicada entre uma a três vezes. Nesta forma sem parâmetro, a opção **multiple** é empregue nalguns programas para reforçar o nível de verbosidade das mensagens de depuração (**-v** algumas mensagens de debug, **-v -v** mais mensagens de debug, etc.). O campo **unsigned int multi_given** da estrutura **struct gengetopt_args_info** devolve o número de vezes que a opção foi especificada

--multi2: Opção múltipla mas que requer parâmetro. Neste caso, a opção **--multi2** pode ser indicada com um a quatro parâmetros (**multiple(1-4)**) do tipo string através da seguinte forma **--multi2=str1,str2,str3** ou **--multi2=str1 --multi2=str2 --multi2=str3**. Em termos de código, o valor dos vários parâmetros está acessível através do vetor de strings **char ** multi2_arg** da estrutura **struct gengetopt_args_info**, sendo o número de elementos do vetor indicado pelo campo **unsigned int multi2_given**.

--enumerada: opção cujo possíveis valores são especificados pelo campo "values". Neste caso, os valores possíveis são "zero", "um" ou "dois", sendo "zero" o valor por omissão.

A Listagem 12 mostra a estrutura **struct gengetopt_args_info** que foi gerada pelo **gengetopt** a partir do ficheiro de configuração **funcionalidades.ggo**.

Opções avançadas: *groups* e *mode*

Grupos

Para além da possibilidade de configuração individual de cada opção, o **gengetopt** suporta ainda a definição de grupo de opções. Para o **gengetopt**, um grupo de opções é um conjunto de opções que estão em exclusão mútua, isto é, na execução da aplicação apenas pode ser especificada uma das opções pertencente a um dado grupo.

A definição de um grupo chamado **NomeGrupo** faz-se através de **defgroup NomeGrupo**. Cada opção pertencente ao

grupo é definida pela palavra-chave **groupoption** em que o parâmetro **group** indica o nome do grupo a que pertence. Para além da necessidade de indicação do grupo a que pertence, a definição de uma opção via **groupoption** é similar à definição de uma opção simples, com exceção que a opção não pode ser definida como obrigatória, pois tal contradiz o propósito de um grupo. Importa ainda referir que caso um grupo seja definido como obrigatório (**required**), então uma e uma só das opções do grupo terá que ser especificada aquando da execução da aplicação. A listagem mostra o ficheiro **grupos.ggo** que exemplifica a definição de grupos (**grupo_1** e **grupo_2**), bem como da opção independente "**normal1**". As linhas iniciadas pela palavra-chave section correspondem a divisões que apenas têm efeito na documentação acessível através da opção --**help**, com uma separação visual por secções. Um exemplo de utilização da funcionalidade de grupos é mostrado na Listagem 13, com o ficheiro de configuração **grupos.ggo**.

```
struct gengetopt_args_info{
  const char *help_help;
  const char *version_help;
  const char *obrigatoria_help;
  const char *opcional_help;
  int flag_flag;
  const char *flag_help;
  char *ficheiro_arg;
  char *ficheiro_orig;
  const char *ficheiro_help;
  char *preenchido_arg;
  char *preenchido_orig;
  const char *preenchido_help;
  long numero_arg;
  char *numero_orig;
  const char *numero_help;
  float argopcional_arg;
  char *argopcional_orig;
  const char *argopcional_help;
  char *enumerada_arg;
  char *enumerada_orig;
  const char *enumerada_help;
  unsigned int multi_min;
  unsigned int multi_max;
  const char *multi_help;
  char **multi2_arg;
  char **multi2_orig;
  unsigned int multi2_min;
  unsigned int multi2_max;
  const char *multi2_help;
  unsigned int help_given;
  unsigned int version_given;
  unsigned int obrigatoria_given;
  unsigned int opcional_given;
  unsigned int flag_given;
  unsigned int ficheiro_given;
  unsigned int preenchido_given;
  unsigned int numero_given;
  unsigned int argopcional_given;
  unsigned int enumerada_given;
  unsigned int multi_given;
  unsigned int multi2_given;
};
```

Listagem 12: esturutura struct gengetopt_args_info criada pelo gengetopt a partir da configuração funcionalidades.ggo

```
section "grupo 1"
defgroup "grupo_1" groupdesc="definição do grupo_1"
required
groupoption "opA_grupo1" - "opção opA do grupo 1"
      group="grupo_1" string argoptional
groupoption "opB_grupo1" - "opção opB do grupo 1"
      group="grupo_1" string multiple
groupoption "opC_grupo1" - "opção opC do grupo 1"
       group="grupo_1" string default="BBB"

section "grupo 2"
defgroup "grupo_2" groupdesc="definição do grupo 2"
groupoption "op1_grupo2" - "opção op1 do grupo 2"
group="grupo_2" float argoptional
groupoption "op2_grupo2" - "opção op2 do grupo 2"
group=”grupo_2” int multiple(1-3)
groupoption "op3_grupo2" - "opção op3 do grupo 2"
group="grupo_2" string default="CCC"

# Opções independentes
section "opções independentes"
option "normal1" n "Opção normal que não pertence a
nenhum grupo" double default="2.71828"
```

Listagem 13: ficheiro de configuração grupos.ggo

Modos

O **gengetopt** suporta ainda outra forma de agregação de opções, através da funcionalidade de "modes". Como o nome sugere, a funcionalidade de **mode** permite a definição de vários modos de execução da aplicação. Assim, por exemplo, é possível definir um **modo A** que implementa determinado tipo de funcionalidade, um **modo B** que implementa outro tipo de funcionalidade e um **modo C** que executa outro tipo de operações. Os modos são mutuamente exclusivos, no sentido em que a execução da aplicação com a indicação de uma ou mais opções de um modo é incompatível com o uso de opções de qualquer outro modo. Mesmo com a existência de modos, o **gengetopt** continua a suportar opções independentes (não pertencentes a modos nem a grupos) que são definidas da forma habitual.

A definição de um modo faz-se com a palavra-chave **defmode** atribuindo-se um nome ao modo. A definição de uma opção pertencente a um modo é feita com a declaração **modeoption**, usando-se o parâmetro **mode=nomeModo** para associação da opção ao modo **nomeModo**. A Listagem 14 **modes.ggo** exemplifica a definição de modos e de opções independentes.

```
package "RevistaProgramar"
version "1.0"
purpose "Exemplificar gengetopt/ modos"
description "Exemplo de modos do gengetopt"
versiontext "Patricio R. Domingues - Revista Pro-
gramar"

# opções "independentes"
section "opções independentes"
option "normal1" n "Opção normal que não pertence a
nenhum modo" double default="2.71828"
option "normal2" i "Outra opção que não pertence a
nenhum modo" optional int default="20"

section "definição de modos"
# modo_1
defmode "modo_1" modedesc="modo 1"
modeoption "opt_A" - "opção A do modo 1" mo-
```

```
de="modo_1" string optional
modeoption "opt_B" - "opção B do modo 1" mo-
de="modo_1" string typestr="filename" optional de-
fault="modo1.txt"
modeoption "opt_C" - "opção C do modo 1" mo-
de="modo_1" long argoptional default="5"

defmode "modo_2" modedesc="modo 2"
modeoption "opt_X" - "opção X do modo 2" longlong
default="123456789" mode="modo_2"
modeoption "opt_Y" - "opção Y do modo 2" string de-
fault="modo2.txt" mode="modo_2"
modeoption "opt_Z" - "opção Z do modo 2" string
multiple(1-2) mode="modo_2"
```

Listagem 14: ficheiro de configuração modes.ggo

Notas finais

O utilitário **gengetopt** simplifica significativamente a gestão das opções da linha de comando de um programa escrito em linguagem C. Em lugar de desenvolver código à medida das opções da linha de comando que pretende que sejam suportadas pela aplicação, o programador define as necessidades da aplicação através do ficheiro de configuração **.ggo** a passar ao **gengetopt**. A intuitiva linguagem de configuração do **gengetopt** e as funcionalidades que a ferramenta disponibiliza possibilitam ao programador o criar de soluções poderosas e facilmente extensíveis para o processamento das opções passadas pela linha de comando.

O **gengetopt** é suportado para ambiente UNIX, nomeadamente para distribuições de Linux. Embora exista um porte do **gengetopt** para ambiente Windows [gengetoptWindows], está um pouco desatualizado, dado que corresponde à versão 2.20, ao passo que a versão corrente do gengetopt é a 2.22.6. Acresce-se que dado o **gengetopt** gerar código dependente das funções **getopt** e **getopt_long** leva a que também seja necessário o porte dessas funções para ambiente Windows. Isso ocorre, por exemplo, no sistema Cygwin [cygwinOnline].

Saber mais

Este artigo apresentou o gengetopt e algumas das suas funcionalidades. Para um conhecimento mais aprofundado da ferramenta gengetopt, como por exemplo, das opções da linha de comando suportadas pela própria ferramenta, recomenda-se a leitura do sítio web [gengetoptOnline] e da própria página do manual do **gengetopt**, acessível através de **man gengetopt**. Recomenda-se ainda uma análise atenta ao código **.c** e **.h** gerado pelo **gengetopt**.

> ***De facto, através dos parâmetros da linha de comando, torna-se possível especificar múltiplos comportamentos e funcionalidades para uma mesma aplicação de modo consola/terminal***

Bibliografia

[argp2010] Ben Asselstine, “Step-by-Step into Argp”, 2010. http://bit.ly/argp2010

[cygwinOnline] Cygwin Online, http://cygwin.com/

[gengetoptOnline] Documentação online do gengetopt. http://bit.ly/gengetopt

[gengetoptWindows] Porte do gengetopt para windows. http://bit.ly/gengetoptwin32

[getoptOnline] Documentação online do getopt. http://bit.ly/getopt

[getoptExemplo] Exemplo do uso da função getopt. http://bit.ly/getoptExemplo

AUTOR

Escrito por Patricio Domingues

Patrício Domingues é professor do Departamento de Engª Informática da Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria (IPLeiria). Leciona, entre outras, a disciplina de Programação Avançada ao curso de Licenciatura em Engenharia Informática, onde tem o privilégio de trabalhar com os colegas docentes Gustavo Reis e Vítor Carreira.

Implementação de Árvores de Vantagem

Nesta edição, em que estamos em pleno Outono e muitas das nossas ruas se cobrem de mantos de folhas, propomos ao leitor uma vista mais reforçada às árvores… Mais concretamente pelas árvores binárias e pelas árvores de vantagem.

Os que me conhecem, sabem que esta foi uma temática que me deu algumas dores de cabeça, mas no fim de contas, este foi um assunto que confesso, ter valido a pena ter aprofundado.

Antes de prosseguirmos vamos recordar, ao leitor, uma característica fundamental das árvores binárias. Este esquema de representação de dados torna-se bastante útil quando queremos armazenar um grande número de dados, contudo as árvores binárias são sempre ordenadas tendo em consideração uma chave ou atributo específico. Todos os elementos à esquerda de um elemento da árvore têm uma chave inferior ao elemento em que estamos e todos os elementos à direita têm uma chave superior.

No exemplo implementado para este artigo, simulámos uma aplicação disponível numa rede de stand de automóveis, em que cada automóvel que essa rede detém, é identificado por um número de série (único), pela cor, marca, ano e ainda qual o número de jante que esse automóvel tem.

Utilizando, então já como exemplo a estrutura principal do problema que ilustra este artigo:

```
typedef struct CARRO{
      long int nSerie;
      int cor;
      int marca;
      int ano;
      int nJante;
      float distancia; //para calcular euclideana
entre pivot
      struct CARRO *nseg;
}Carro;
```

Se construíssemos uma árvore binária para armazenar a informação sobre todos os automóveis que tivéssemos disponíveis, uma escolha lógica seria guardar a informação ordenada pelo número de série pois é a única característica que identifica com precisão cada automóvel. Seria também bastante útil na implementação do sistema, balancearmos essa mesma árvore cada vez que lhe adicionamos um novo elemento. A árvore seria então considerada uma árvore balanceada pela altura (AVL), ou seja a diferença entre o número de elementos no ramo direito e o número de elementos presentes no ramo esquerdo é no máximo 1.

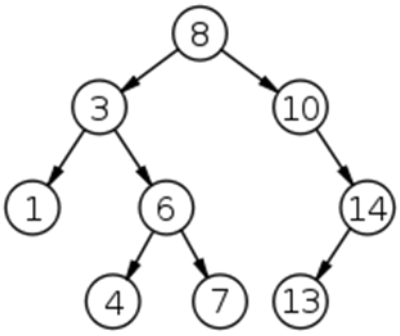


Ilustração 1 - Exemplo de AVL

Tendo em conta que as árvores binárias têm que estar sempre ordenadas por um atributo específico, só há vantagem se pesquisarmos por esse atributo, neste caso apenas teríamos vantagem em pesquisar pelo atributo número de série do carro que queríamos.

Mas em que é que este esquema nos ajudaria caso entrasse no nosso stand um cliente interessado num automóvel com determinadas características? Como poderia o nosso sistema dar-nos uma resposta em tempo útil caso não soubéssemos de antemão todos os números de série dos automóveis que achávamos que se adequavam a esse cliente?

Praticamente em nada. Porque teríamos sempre que ir pesquisar recursivamente todos os elementos da árvore considerando todas as características dentro de cada elemento. Como cada subárvore dentro da árvore principal tem um "filho direito" e um "filho esquerdo", tendo em conta que teríamos que procurar em todos os elementos, isto seria o mesmo que fazer uma pesquisa sequencial numa lista duplamente ligada. Ou seja, perderíamos a que é considerada a grande vantagem de utilizar árvores binárias, a rapidez de pesquisa pelo facto de não termos que passar por todos os elementos do nosso conjunto de dados.

Voltando então ao nosso problema inicial… como poderíamos dar uma resposta rápida a um cliente que entrasse no stand e estivesse interessado numa série de características de um automóvel? Como poderíamos dar-lhe um conjunto de veículos que correspondessem o mais possível à procura deste nosso cliente? É esta resposta que o artigo desta edição se propõe a implementar. Para isso vamos recorrer às árvores de vantagem, também conhecidas como Vantage Point Trees.

Para começar vamos considerar toda a informação que temos na árvore binária, a nossa base de dados ou a nossa coleção de objectos. Quando queremos pesquisar por uma

série de características na base de dados, fazemos uma query a essa mesma base de dados, e obtemos então um subconjunto da informação armazenada. Ou seja, ao contrário das AVL, que apenas nos permitem fazer uma pesquisa exacta, as árvores de vantagem (VPT) permitem que seja feita uma pesquisa por semelhança. A construção destas árvores é baseada num espaço métrico, que neste exemplo é baseado numa métrica de distância (a distância euclideana) em que quanto maior for a distância menos parecido é esse elemento. Para construirmos a nossa árvore vamos usar o Principio de partição, ou seja, para um ponto P, calculam-se os pontos exteriores e os pontos interiores num raio de vizinhança do conjunto.

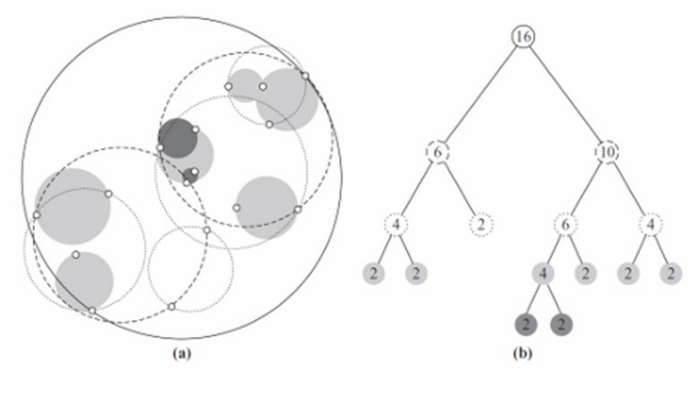


Ilustração 2 - Implementação VPT

Assim, de uma forma simples, a criação da árvore de vantagem é feita com a ajuda da mediana, usando a distância euclideana de um determinado conjunto em relação a um elemento. A este elemento dá-se o nome de Pivot. A escolha do Pivot é feita ao acaso, de forma aleatória. Despois desta escolha é calculada a distância euclidiana do resto do conjunto em relação a esse Pivot. Seguidamente é calculada a mediana do conjunto e os elementos desse conjunto são separados nas listas in e out (elementos com distância euclideana com valores inferiores e superiores à mediana, respectivamente).

A criação da VPT é feita de forma recursiva, chamando de novo a função getMediana para cada uma das listas in e out, de forma a ir subdividindo o espaço em subconjuntos cada vez mais pequenos até que todos os elementos do conjunto tenham o seu próprio subespaço. (Devo recordar ao leitor que este é apenas um dos métodos que pode ser usado para criar a VPT, outros métodos podem ser utilizados para este fim.)

Como cada subconjunto é feito em relação ao Pivot aleatoriamente escolhido e é o Pivot que é também chamado o elemento de vantagem desse subconjunto.

Este tipo de árvores permitem-nos dois tipos de pesquisas bastante conhecidos: a Pesquisa de Intervalo e a Pesquisa dos Vizinhos Mais Próximos (KNN).

Uma outra vantagem de usarmos este tipo de representação de conhecimento, é que além da VPT permitir realizar pesquisas por semelhança, há também forma de realizar uma pesquisa exacta (como fazemos nas AVL).

E como? – Poderá perguntar o leitor, caso queiramos uma pesquisa exacta basta-nos realizar uma pesquisa em que o raio máximo de distância seja zero (o raio corresponde à diferença de características, logo todas as características serão iguais às inseridas na query).

Para este artigo foram ainda usadas mais duas estruturas:

```
typedef struct AV{
      Carro *pivot;
      float mediana;
      struct AV*fe;
      struct AV*fd;
}Av;

typedef struct RKNN{
      long int nSerie;
      float distancia;
}Rknn;
```

A estrutura de dados AV é a forma como a informação fica armazenada na VPT, em que cada elemento tem um elemento do tipo Carro (o pivot), o valor da mediana dos conjuntos pertencentes a esse mesmo pivot e dois apontadores, o apontador para o filho esquerdo (que contem os elementos menores que a mediana) e o apontador para o filho direito (que contem os elementos do conjunto com mediana superior), ou seja, recapitulando mais uma vez, temos o pivot que é o elemento de vantagem do conjunto, temos a mediana dos restantes elementos a esse pivot e temos o apontador do filho esquerdo (a lista in) e o apontador do filho direito (a lista out). Cada elemento da árvore de vantagem contém esta informação.

Continuando o nosso raciocínio, para apresentar os resultados da query do cliente, usamos também a estrutura RKNN para irmos armazenando facilmente os dados que nos interessam à medida que vamos realizando a pesquisa. Para isso usamos um vector com tantas posições quanto o número de resultados que o nosso cliente deseje. Quando acabarmos a pesquisa teremos sempre os carros que mais se aproximam das características que o cliente pediu. (Nesta implementação optamos por guardar apenas nesta estrutura temporária a distância e o número de série do carro que corresponde a essa mesma distância. Esta opção foi tomada para simplificar a apresentação de resultados neste artigo, caso o leitor queira pode também optar por mostrar todas as características relativas ao veículo, de uma forma simples, bastaria implementar uma nova função que recebendo um determinado número de série de um veículo, procuraria na AVL por esse mesmo número de série e apresentaria as características desse mesmo elemento.)

Implementação da Árvore de Vantagem

No programa que ilustra este nosso artigo quando inserimos os carros no programa, estes são armazenados numa lista, que é uma estrutura de acesso sequencial. Essa lista de carros é passada como parâmetro à função criaArvoreVanta-

IMPLEMENTAÇÃO DE ÁRVORES DE VANTAGEM

gem de forma a que a VPT seja implementada recorrendo ao método anteriormente explicado.

Assim, para esta implementação são usadas as seguintes funções principais:

```
Av* criaArvoreVantagem(Carro *C);
Carro* escolhePivot(Carro *C);
float getMediana(Carro *C, Carro *pivot);
Carro* divideLista(Carro *C, float m, Carro **I,
                          Carro**O, Carro *pivot);
```

Passemos agora ao código em C, onde comentámos a maior parte das funções principais.

```
Av* criaArvoreVantagem(Carro *C){

      Carro *in=NULL;
      Carro *out=NULL;
      Av *novoElemento;

      if(C==NULL)
            return NULL;

      //aloca a memória para o novo elemento da AV
      novoElemento=makeNodeArvore();
   //Escolhe aleatoriamente um pivot da lista de
   //carros
      Carro *pivot=escolhePivot(C);
//Calculo da mediana do pivot em relação aos
//restantes elementos da lista
      float mediana=getMediana(C,pivot);
        //Divide a lista de carros em duas
//sublistas, a lista dos elementos com a mediana
inferior e a lista de elementos com a mediana
//superior ao pivot
      C=divideLista(C,mediana, &in, &out,pivot);
      //Copia da informação da caracteristicas
//do elemento
      novoElemento->pivot->nSerie=pivot->nSerie;
      novoElemento->pivot->distancia=
                                 pivot->distancia;
      novoElemento->pivot->cor=pivot->cor;
      novoElemento->pivot->marca=pivot->marca;
      novoElemento->pivot->ano=pivot->ano;
      novoElemento->pivot->nJante=pivot->nJante;

      novoElemento->mediana=mediana;

      novoElemento->fe=criaArvoreVantagem(in);
      novoElemento->fd=criaArvoreVantagem(out);
      in=clearList(in);
      out=clearList(out);

      return(novoElemento);

}

//Função em que é escolhido aleatoriamente o pivot
//a usar
Carro* escolhePivot(Carro *C){
      srand(getpid());
      int total=countNodes(C);
      int s=rand()%total;

      for(;s>0;s--)
            C=C->nseg;

      return(C);

}

//função para subdividir os elementos da lista em
duas sublistas
Carro* divideLista(Carro *C, float m, Carro **I,
Carro **O, Carro *pivot){

      float d;
      Carro *novo;
      Carro *Carro=C; //cabeça da lista

      while(C!=NULL){
            if(C!=pivot){
                  d=euclideana(C,pivot);
                  novo=makeNode();
                  novo=copiaNodo(novo,C);
                  if(d<=m)
                        *I=insertLast(*I, novo);
                  else
                        *O=insertLast(*O, novo);
            }//fim do if
            C=C->nseg;
      }//fim do while
      return(Carro);
}


//calculo da mediana dos elementos da lista, devo
//recordar o leitor que esta função de calculo é
//apenas um exemplo, o leitor pode usar outra
//caso assim  o deseje.
float getMediana(Carro *C, Carro *pivot){

      int total, totMaiores;
      float maiorMenores;
      Carro *Car=C;
      Carro *aux=C, *aux2;

 while(Car!=NULL){
            if(Car!=pivot)
                  Car->distancia=euclideana
(Car,pivot); //calc da mediana entre a Lista e
//Pivot
            Car=Car->nseg;
      }


      while(aux!=NULL){
            total=0;
            totMaiores=0;
            aux2=C;
            maiorMenores=0.0;
            while(aux2!=NULL){
                  if(aux->distancia>=
                                    aux2->distancia)
                        totMaiores++;
                  else{
                        if(aux2->distancia>
                                    =maiorMenores)
                              maiorMenores=
                                    aux2->distancia;
                                    }
                  total++;
                  aux2=aux2->nseg;
            }//fim while aux2

            if(total%2==1){
                  if(totMaiores-1==total/2){
                        return(aux->distancia);
                        }
                  }
                  else{
                        if(totMaiores==total/2){
                              return((
                        aux->distancia+maiorMenores)/2);
                        }
                  }
```

```
            aux=aux->nseg;
      }//fim do while aux

}
```

Para que o método da criação da VPT seja claro, vamos mais uma vez recapitular, a função criaArvoreVantagem recebe como parâmetro de entrada a lista de carros, dessa lista é escolhido um elemento aleatoriamente, usado como Pivot. Após essa escolha, a função getMediana é chamada de forma a que possa ser calculada a distância euclideana entre todos os elementos restantes da lista em relação ao Pivot. Após essa operação é calculada a mediana desse conjunto. Uma vez sabendo a mediana, é chamada a função divideLista que percorre a lista de carros, de forma a que possam ser criadas duas novas sublistas de forma a ter na lista in os elementos com uma distância menor que a mediana do conjunto e na lista out são inseridos os elementos com a distância superior à mediana. Assim que estes subconjuntos são criados é então copiada a informação respeitante às características do pivot e a mediana desse conjunto para o novo nodo Av. Seguidamente é chamada novamente a função criaArvoreVantagem para cada filho (fe e fd) desse nodo, de forma a ir criando a VPT até termos todos os elementos "arrumados".

A função copiaNodo que é usada dentro da função divideLista, serve para copiar a informação de cada nodo Carro presente na lista (que foi previamente escolhido como Pivot) para o novo nodo Av de forma a que esse nodo seja inserido na árvore de vantagem.

A criação desta árvore é feita no início da execução do programa e devido à quantidade de cálculos que são efectuados, a criação desta árvore é mais lenta que a criação de uma árvore binária normal. Contudo, e como apenas precisamos de a criar no arranque do programa, continua a ser vantajoso usar esta representação de dados pois as queries posteriormente efectuadas serão feitas de forma mais rápida.

```
//Cópia das características do elemento da lista
//para um novo elemento
Carro* copiaNodo(Carro *nv, Carro *C){

      nv->nSerie=C->nSerie;
      nv->cor=C->cor;
      nv->marca=C->marca;
      nv->ano=C->ano;
      nv->nJante=C->nJante;

      return nv;
}
```

Ora e então chegados "à hora da verdade", agora que já temos a VPT como podemos usá-la em nosso proveito?

Considerámos dois tipos de pesquisa neste artigo:

- Pesquisa de Intervalo: em que o utilizador introduz as características a pesquisar e introduz também o raio de procura - devem ser devolvidos todos os objectos com distância d() inferior ou igual ao valor do raio.
- Pesquisa de Vizinhos Mais Próximos: O utilizador introduz as características a pesquisar e introduz também o número de resultados que quer ver apresentados. Devem ser devolvidos os "k" objectos com menor distância d() ao objecto. (Esta pesquisa é também conhecida como a Pesquisa KNN).

O algoritmo usado na pesquisa de intervalo é simples, para cada elemento AV da árvore deve verificar-se qual o valor da mediana (que foi previamente calculado quando na função criaArvoreVantagem) e se a distância euclideana entre a query pedida e o elemento em questão for menor ou igual ao raio pedido, então devemos incluir esse elemento no resultado final. E como tomamos a decisão de continuar a procura no ramo esquerdo ou no ramo direito da árvore, pergunta o leitor? Simples… à distância entre a query e o elemento de vantagem subtraímos o raio pedido pelo utilizador. Se o resultado for menor ou igual que a mediana desse mesmo elemento então devemos pesquisar apenas na subárvore esquerda. Para sabermos se devemos pesquisar na subárvore direita, somamos o raio pedido à distância entre a query e o elemento de vantagem, se o resultado for maior ou igual à mediana desse elemento, neste caso a pesquisa deve ser feita na subárvore direita.

No caso da pesquisa dos vizinhos mais próximos (KNN) para cada elemento da árvore devemos calcular a distância euclideana entre a query pedida e o elemento da árvore em que estamos. Caso a distância seja menor que a distância que calculámos anteriormente então devemos atribuir à variável de controle (que inicialmente está inicializada ao INT_MAX da linguagem C) essa distância e incluir esse elemento na nossa resposta final, ou seja, naquele vector que já falámos anteriormente neste artigo. A decisão de percorrer a subárvore direita ou a subárvore esquerda é feita de forma quase análoga ao algoritmo anterior; Caso o elemento em que estamos e a query pedida tenham uma distância que, subtraindo o valor da variável de controlo (a variável dnn), seja menor ou igual à mediana desse mesmo elemento, então devemos procurar na subárvore esquerda. Se a distância, somando o valor da variável de controle, for maior ou igual à mediana do elemento então percorremos a subárvore direita.

Estes dois algoritmos podem parecer complicados e confusos à primeira vista, mas quando o leitor verificar mais calmamente cada um deles, rapidamente vai também achá-los mais simples.

Para clarificar as dúvidas que ainda possam existir, vamos agora então às duas funções principais deste artigo…

Pesquisa KNN (K-nearest Neighbor)

```
void procuraKNN(Av *arvore, Carro *query, int
                        carros, Rknn *maisProximos){

      int i, flag=0, controle=0;

      float distancia, dnn; //dnn é a variavel de
//controle
```

IMPLEMENTAÇÃO DE ÁRVORES DE VANTAGEM

```
      if(arvore==NULL)
            return;

      distancia=euclideana(arvore->pivot, query);

      for(i=0;i<carros;i++){
            if(distancia<=maisProximos[i].distancia)
{
                  if(maisProximos
                    [carros-1].distancia>distancia){
                        maisProximos
                     [carros-1].distancia=distancia;
                        maisProximos
            [carros-1].nSerie=arvore->pivot->nSerie;
                              dnn=distancia;
                              controle++;
                  }
                  bubbleSort(maisProximos, carros);
                  break; //para não inserir em todas
                         //as posicoes do vector....
            }
             if(controle<carros)
                  dnn=maisProximos
                              [controle-1].distancia;

                  dnn=maisProximos
                              [carros-1].distancia;
      }

                  if(distancia-dnn<=
                                    arvore->mediana)
                        procuraKNN(arvore->
                  fe, query,carros, maisProximos);

            if(distancia+dnn>=arvore->mediana){
                  procuraKNN(arvore->fd,
                        query,carros, maisProximos);

      }
}


void trocar(Rknn *vector, int j, int y){

      long int nserie;
      float d;

      nserie=vector[j].nSerie;
      d=vector[j].distancia;

      vector[j].nSerie=vector[y].nSerie;
      vector[j].distancia=vector[y].distancia;

      vector[y].nSerie=nserie;
      vector[y].distancia=d;

}

void bubbleSort(Rknn *vect, int carros){

      int i, j;

      for(i=carros-1; i>0;i--){
            for(j=0;j<i;j++){
                  if(vect[j].distancia>vect
[j+1].distancia)
                        trocar(vect,j,j+1);
            }
      }
}
```

Como o leitor pode verificar, esta função recebe como parâmetros a query introduzida pelo utilizador, assim como um apontador para a árvore de vantagem criada no início da execução do programa. Recebe ainda um apontador para um vector onde serão guardados os valores dos N indivíduos mais próximos da query introduzida. Este vector tem tantas posições como o número de carros que o cliente peça ao nosso programa, desta forma o vector vai ser ordenado a cada iteração, caso seja necessário. Quando o programa "avança" na árvore de vantagem, é calculada a distância euclideana entre a query e o pivot, caso essa distância seja menor do que as que já estão no vector, então essa distância (assim como o atributo numero de Série do pivot), são introduzidos na última posição do vector (neste caso no vector maisProximos [carros-1]), pois como encontrámos uma distância mais baixa do que alguma presente no vector, automaticamente sabemos que já não nos interessa guardar o valor que está na última posição. Seguidamente o vector volta a ser ordenado pelo campo distância recorrendo à famosa função bubbleSort.

A única diferença entre a função procuraKNN e uma função que fosse implementada de forma recursiva é que a função recursiva passaria por todos os nós da árvore de vantagem e a função procuraKNN tira partido da mediana, ou seja é usado o algoritmo KNN de forma a que se tire partido do elemento de vantagem, decidindo qual dos ramos (esquerdo ou direito) se vai continuar a pesquisar os N indivíduos mais próximos. Isto torna-se vantajoso pois decidindo qual dos ramo se pesquisa, automaticamente estamos a eliminar metade das possibilidades de procura, tornando assim a nossa aplicação mais rápida, dando desta forma uma resposta em menor tempo útil ao utilizador.

Pesquisa por Intervalo

```
void procuraRaioVantagem(Av*arvore, Carro *query,
float raio){


      float distancia;

      if(arvore==NULL)
            return;

      distancia=euclideana(arvore->pivot, query);

      if(distancia<=raio){
            printf("\nDistancia = %f.",
                                    distancia);
            printf("\nCaracteristicas do Carro:");
            mostraPivot(arvore->pivot);
            }
      if(distancia-raio<=arvore->mediana)
            procuraRaioVantagem(arvore->fe,
                                    query, raio);

      if(distancia+raio>=arvore->mediana)
            procuraRaioVantagem(arvore->fd, query,
raio);

}
```

Na procuraRaioVantagem é calculada a distância entre a

query introduzida e o pivot e caso a distância seja menor que o raio pedido, as características do carro são mostradas. A função recorre à mediana do elemento de vantagem de forma a escolher em qual ramo (esquerdo ou direito) vai continuar a sua pesquisa.

> ***"A característica fundamental das árvores binárias (...) Todos os elementos à esquerda de um elemento da árvore têm uma chave inferior ao elemento em que estamos e todos os elementos à direita têm uma chave superior."***

Em jeito de conclusão, quando o programa inicia a sua execução, logo após o "administrador de sistema" inserir toda a informação na nossa base de dados improvisada (neste caso a lista), a árvore de vantagem é calculada e criada pelo programa. Em pequenos conjuntos de dados, o utilizador não irá conseguir verificar que o programa realmente dá uma resposta em tempo útil. Contudo quanto maior for o volume de informação a constar da base de dados mais se vai notar a diferença de resposta entre uma implementação recursiva ou usando o elemento de vantagem. Para este artigo foi considerado um pequeno conjunto de dados, o stand, contudo pegando num exemplo do mundo real, imaginando uma base de dados em que constam todos os carros existentes no nosso pais, poderíamos ter que efectuar uma pesquisa nesse mesmo sistema procurando um carro com determinadas características. Neste caso, as VPT conseguiriam continuar a dar uma resposta em tempo útil enquanto a procura recursiva iria demorar mais tempo a responder.

> ***"(...) ao contrário das AVL, (...) as (VPT) permitem que seja feita uma pesquisa por semelhança. (...)"***

A única demora que poderá ser visível é aquando do cálculo da própria árvore de vantagem visto que há várias suboperações que estão presentes na sua criação mas, quando a VPT estiver criada, o sistema estará apto a responder sem qualquer problema e em tempo útil.

(O código fonte na linguagem C está disponível para download no fórum do Portugal-a-Programar para todos os leitores interessados. Usamos para este exemplo características do tipo int para que fosse mais simples a explicação, e não foram tomadas em atenção a verificação de valores pelo que o leitor deve levar isso em consideração).

Aos leitores mais curiosos deixo esse mesmo desafio. O de criar a vossa própria árvore de vantagem e tirarem as vossas próprias conclusões num universo de dados que seja considerável. Poderão até usar a função *Clock()* em C para melhor medirem as diferenças do tempo de desempenho do vosso programa. As árvores de vantagem podem ser um pouco complicadas à primeira vista, principalmente quando as compararmos às ditas "árvores binárias normais", mas em muitos casos podem sim ser uma forma de implementação a considerar porque "tudo" o que nos ajudar a dar ao nosso utilizador final uma resposta mais rápida, deverá ser sempre levado em conta!

AUTOR

Escrito por Rita Peres

Natural de Castelo Branco, licenciou-se em Engenharia Informática pela Universidade da Beira Interior. Membro do P@P desde Janeiro de 2010.

Listas Duplamente Ligadas

Conforme o prometido no artigo "Listas Simplesmente Ligadas e Exemplo de Implementação em C" da passada edição, aqui está o artigo das listas duplamente ligadas. Uma vez que este se engloba num mesmo tema, certos aspectos podem tornar-se um pouco repetitivos, isto porque a única diferença no nó entre os dois tipos de listas referidos é um apontador adicional nas segundas.

Para começar, vamos estudar as vantagens das listas duplamente face às simplesmente ligadas. Se bem se lembram uma lista simplesmente é possível percorre-la apenas num sentido. Nas que vamos ver de seguida é possível percorre-las em ambos os sentidos e é essa a principal diferença entre o modo de funcionamento das segundas. Esta diferença é muito vantajosa. Imaginemos que estamos no nó número 1000 e queríamos imprimir o conteúdo do nó 999, com as listas simplesmente ligadas teríamos de voltar à cabeça da lista e percorrer os 999 nós até chegarmos ao pretendido, com as listas duplamente ligadas basta andar uma posição para trás.

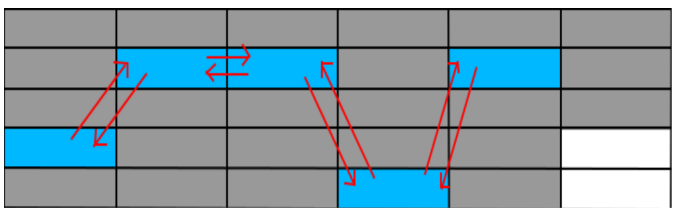
O que muda na estrutura de cada nó relativamente às simplesmente ligadas, como já foi referido, é apenas a adição de um novo apontador, o qual irá apontar para o elemento da posição imediatamente anterior da lista.

Já que estamos a falar de apontadores convém alertar para uma situação muito específica desta estrutura de dados. Vamos focar-nos na primeira posição da lista. Se todas as posições têm um apontador para os elementos anterior e posterior, como será que definimos o apontador para elemento anterior ao da cabeça?

Este apontador tem de ficar sempre a NULL e deve ser usado como critério de paragem quando percorremos a lista no sentido inverso.

Segue-se a figura ilustrativa de cinco elementos de uma lista duplamente ligada alocados em memória.

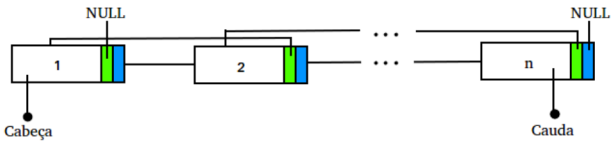
Figura 1 – Exemplo da alocação em memória de uma lista com 5 elementos



Na figura 1, os rectângulos cinzentos são espaços de memória não disponíveis, os brancos estão disponíveis e os azuis são os elementos da lista. Para quem não sabe como uma lista funciona, deve de estar a perguntar-se o que fazem as setas na figura. Como referido no artigo anterior, as listas são estruturas de dados de acesso sequencial, ou seja, as setas representam apontadores de um elemento da lista para o seguinte ou para o anterior, formando assim uma sequência.

A próxima imagem dá uma ideia gráfica de uma lista com N elementos:

Figura 2 – Exemplo visual da estrutura de uma lista duplamente ligada



Na figura 2, cada conjunto de rectângulos maiores representa um nó da lista, sendo que os rectângulos menores azuis representam o apontador do nó para o próximo elemento e os verdes representam o apontador do nó para o elemento anterior. Note que na cabeça da lista o apontador para o nó anterior está a apontar para NULL e que na cauda da lista o apontador para o próximo elemento está a NULL de igual forma.

Para percebermos como se implementam as listas duplamente ligadas, nada melhor do que definirmos um problema para o resolver e vamos nos basear no problema do artigo anterior: uma escola que precisa de um sistema básico para registar os seus alunos. O nosso programa de exemplo deverá ter 3 opções, uma para adicionar alunos, outra para os remover e outra ainda para imprimir duas vezes. a lista de todos os alunos A primeira vez será fazer uma impressão do elemento de menor valor para o de maior valor. Já a segunda fará uma outra impressão mas no sentido oposto ao anterior. Nesta última opção vamos perceber a verdadeira utilidade das listas duplamente ligadas. Vamos ter ainda em conta que a inserção na lista deve ser feita por ordem numérica.

Código:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TAM 100

typedef struct no {
  int numero;
  char nome[TAM];
  struct no *nseg, *nant;
} No;
```

```
No* insereNaLista(No *lista, No *novoNo){
  No *auxiliar, *cabeca = lista;

  //caso a lista esteja vazia
  if (lista == NULL)
    return novoNo;

  // 1º elemento > elemento a inserir
  if (lista->numero > novoNo->numero){
    novoNo->nseg = lista;
    lista->nant = novoNo;
    return novoNo;
  }
  while (lista != NULL){
    auxiliar = lista;
    lista = lista->nseg;

    // inserir no fim da lista
    if (auxiliar->nseg == NULL){
      auxiliar->nseg = novoNo;
      novoNo->nant = auxiliar;
      return cabeca; //retorna a cabeça
    }

    // inserir no meio da lista
    if (lista->numero >= novoNo->numero) {
      auxiliar->nseg = novoNo;
      novoNo->nseg = lista;
      lista->nant = novoNo;
      novoNo->nant = auxiliar;
      return cabeca;
    }
  }
  return cabeca;
}

No* removeDaLista(int numero, No *lista){
  No *auxiliar, *auxiliar2, *cabeca = lista;

  //lista vazia
  if (lista == NULL) return lista;

  if (lista->numero == numero){
    // caso seja o primeiro a remover
    cabeca = lista->nseg;
    free (lista);
    // único elemento
    if (cabeca == NULL) return cabeca;
    cabeca->nant = NULL;
    return cabeca;
  }
  while (lista != NULL){
    auxiliar = lista;
    lista = lista->nseg;
    if (auxiliar->nseg == NULL){
      // remove o ultimo elemento
      auxiliar2 = auxiliar->nant;
      auxiliar2->nseg = NULL;
      free (auxiliar);
      return cabeca;
    }
    if (auxiliar->numero == numero){
      // remove no meio da lista
      auxiliar2 = auxiliar->nant;
      auxiliar2->nseg = auxiliar->nseg;
      lista->nant = auxiliar2;
      free(auxiliar);
      return cabeca;
    }
  }
  return cabeca;
}

No* criaNo(char *nome, int numero, No *lista){
  No *novoNo;
  novoNo = malloc(sizeof(No));
  novoNo->nseg = novoNo->nant = NULL;
  novoNo->numero = numero;
  strncpy(novoNo->nome, nome, TAM);
  lista = insereNaLista(lista, novoNo);
  return lista;
}

void imprimeLista(No *lista){
  No *auxiliar = NULL;
  if (lista == NULL) {
    printf("A lista está vazia!\n");
    return;
  }
  printf("\nAscendente:\n");
  while (lista != NULL){
    printf("%6d\t", lista->numero);
    puts(lista->nome);
    auxiliar = lista;
    lista = lista->nseg;
  }
  lista = auxiliar;
  printf("\nDescendente:\n");
  while (lista != NULL){
    printf("%6d\t%s\n", lista->numero, lista-
>nome);
    lista = lista->nant;
  }
}

int main(void){
  int menu = 1;
  int numeroAluno;
  char nome[TAM];
  No *lista = NULL;
  No *aux = NULL;

  while (menu != 0){
    printf("1 - Inserir Aluno\n");
    printf("2 - Remover Aluno\n");
    printf("3 - Mostrar Lista Dos Alunos\n");
    printf("4 - Sair\n");
    scanf("%d", &menu);

    switch(menu){
      case 1:
        printf("Nome: ");
        scanf("\n");
        scanf("%99[^\n]", nome);
        printf("Numero: ");
        scanf("%d", &numeroAluno);
        printf("\n");
        lista = criaNo(nome, numeroAluno, lista);
        break;

      case 2:
        printf("Número do aluno a remover: \n");
        scanf("%d", &numeroAluno);
        lista = removeDaLista(numeroAluno, lista);
        break;

      case 3:
        imprimeLista(lista);
        break;

      case 4:
        menu = 0;
        break;

      default:
        ;  // não fazer nada
    }
  }

  while (lista != NULL){
    aux = lista;
```

```
    lista = lista->nseg;
    free(aux);
  }

  return 0;
}
```

No código acima os ponteiros nant e nseg são utilizados para apontar para as estruturas anterior e posterior, respectivamente.

> ***Se bem se lembram uma lista simplesmente ligada é possível percorre-la apenas num sentido. Nas que vamos ver de seguida é possível percorre-las em ambos os sentidos e é essa a principal diferença entre o modo de funcionamento das segundas (...)***

No código podemos ainda ver que as funções adicionar e remover alunos na lista são um pouco complexas, isto porque precisamos de tratar as quatro situações que podem ocorrer

> ***(...) como já foi referido, é apenas a adição de um novo apontador, o qual irá apontar para o elemento da posição imediatamente anterior da lista.***

neste problema. Elas são: a lista estar completamente vazia; querermos inserir ou remover na cabeça da lista; querermos inserir ou remover na cauda da lista; e ainda, queremos inserir ou remover um novo elemento no meio da lista.

Na função imprimeLista(), podemos então verificar a utilidade que este novo tipo de listas tem. Caso nos deparássemos com a impossibilidade de voltar aos elementos anteriores do modo como foi exemplificado com este tipo de estrutura da lista, então teríamos de optar uma uma solução mais complexa e computacionalmente mais exigente.

Espero ter agradado os leitores!

Bibliografia

Websites:

http://www.di.ubi.pt/~hugomcp/estruturas/ 07/10/2013

http://c.learncodethehardway.org/book/ex32.html 07/10/2013

AUTOR

Escrito por Cristiano Ramos
Estudante de Engenharia Informática na Universidade da Beira Interior.
cdgramos@live.com.pt

www.cdgramos.com

Expande o Teu Mercado, Globalizando a Tua App!

Uma das atracções da Store do Windows é a possibilidade de disponibilizar a nossa aplicação, não apenas em Portugal, mas também em qualquer um dos mais de 200 países* em que a Store de Windows está disponível. Se o leitor considerar que o número de habitantes de Portugal ronda os 10,6 milhões de habitantes, e Reino Unido, por exemplo, cerca 61,7 milhões, França 64,3 milhões, ou EUA 300 milhões, podemos ver a diferença do potencial de mercados. No entanto, para tal ser possível, é necessário disponibilizar a aplicação nas várias línguas. Neste artigo vou falar de como podemos desenvolver a aplicação de forma a podê-la traduzir nas várias Línguas, para que a aplicação detecte a língua do Windows e mude automaticamente os textos para a linguagem correspondente, sem intervenção do utilizador.

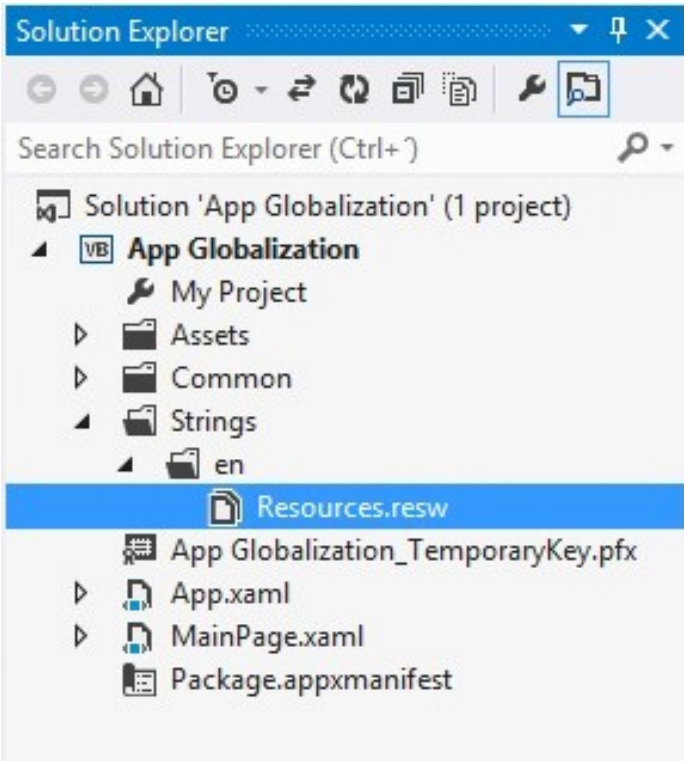
Para efeitos de demonstração vou usar uma aplicação básica, (chamemos-lhe Hello World!), que tem como propósito cumprimentar o utilizador na sua Língua.

Temos então com um Aplicação (em Português) com um Título: “Hello World!”, uma TextBox, onde pedimos para o utilizador introduzir o seu Nome, um botão para dizer “Olá!” e um TextBlock onde o utilizador vai ser saudado.

Antes de começar a tradução vamos definir qual é a língua por defeito da aplicação (a língua que será apresentada, caso de a língua do utilizador não faça parte do nosso pacote de línguas). Para isso acedemos ao nosso package.appxmanifest, tab Application UI, e vamos definir a língua por defeito da nossa aplicação como “en” (Inglês, língua universal).

O passo seguinte é criar uma pasta que vai conter as traduções. Vamos ao Solution Explorer, clicamos lado direito rato no Projeto, Add -> New Folder e chamamos Strings. Dentro desta pasta vamos criar outra pasta (uma por cada língua), que vai conter o Resource File, a que vamos chamar “en”.

Note-se que o nome das pastas criadas, deve obedecer a uma nomenclatura pré-determinada, chamada “BCP-47 language tag” **. Para usar Strings diferentes para Inglês dos Estados Unidos e Inglês Reino Unido, basta criar uma pasta en-US e outra en-UK.



Vamos adicionar o nosso Resource File, que vai conter as Strings em Inglês. Clicamos com lado direito no rato na pasta “en”, Add > New Item > Resources File (.resw) e clicamos Add (recomenda-se usar o nome por defeito).

Começamos então o nosso processo de tradução. Temos uma Grid com três colunas, uma para o Nome da String, a segunda com o texto que será apresentado e uma terceira coluna para observações. Para a nossa aplicação vamos usar três strings: Uma para a TextBlock “Introduza o seu nome” chamada “EnterName”, uma o botão, chamada “SayHello” outra chamada “Hello”, com o texto: “Olá”. Os nomes das strings devem ser usados sem espaços e sem caracteres especiais e acentos, preferencialmente. (Nota: existe também a possibilidade de definir propriedades, como largura do controlo, adicionando uma string “SayHello.Width” com a largura).

Name	Value	Comment
EnterName	Enter your name:	String para TextBlock Introduza o Nome
SayHello	Say Hello!	String para o Botão
Hello	Hello	String para TextBlock Saudação
*		

EXPANDE O TEU MERCADO, GLOBALIZANDO A TUA APP!

Temos então as nossas *Strings* traduzidas para Inglês, vamos carrega-la para aplicação. Podemos fazê-lo de duas formas:

Em *Desgin Mode* no XAML, adicionando a seguinte linha ao controlo x:Uid="EnterName" Text="", ficando, por ex. : <TextBlock x:Uid="EnterName" Text="" />

Ou em modo dinâmico, em Code Time, que é como iremos fazer nesta aplicação:

Acedemos ao código da página que vamos traduzir (o código que deste artigo está em VB.NET, mas está também disponível para cada uma das linguagens disponíveis para desenvolvimento de aplicações para Windows 8), e usamos a seguinte linha de código para carregar os recursos (o meu conselho é usar logo após Public Class (nome página) / Inherits…, de forma a estar disponível em todos os eventos):

```
Dim loader = new
Windows.ApplicationModel.Resources.ResourceLoader()
```

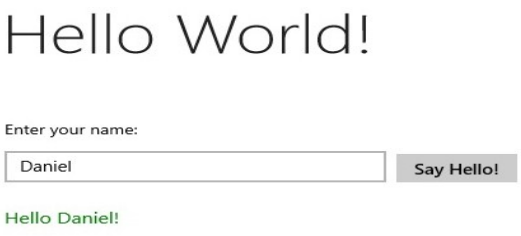
Ao evento LoadState da Página, adicionamos o seguinte código:

```
TextBlockHello.Visibility = Win-
dows.UI.Xaml.Visibility.Collapsed        Text-
BlockEnterName.Text = loader.GetString("EnterName")
btnSayHello.Content = loader.GetString("SayHello")
```

Para o código do evento Click do botão SayHello (para criar eventos, em Design, selecionamos o botão, vamos a Events, no evento Click, escrevemos btnSayHello_Click e premimos Enter):

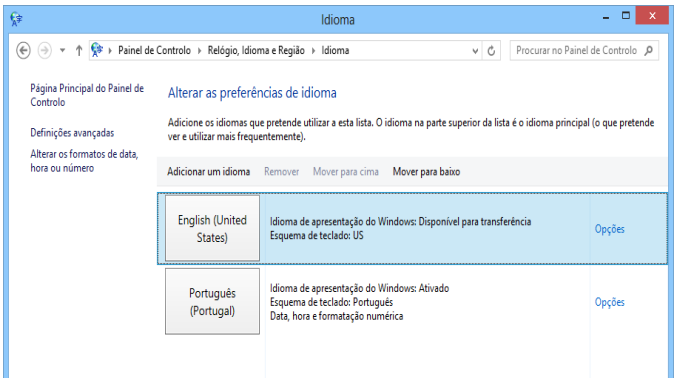
```
TextBlockHello.Text = loader.GetString("Hello") &
TextBoxName.Text & "!"
TextBlockHello.Visibility =
                Windows.UI.Xaml.Visibility.Visible
```

Quanto a código é tudo, vamos testar, corremos a aplicação, escrevemos o nosso nome, clicamos no botão e podemos ver que a App está em Inglês (visto que ainda não adicionamos a tradução Portuguesa).



Para que a nossa App volte a ficar em Português, teremos de criar uma subpasta pt na pasta Strings, e criar um Resource File semelhante ao que criamos para o Inglês, mas com as Strings em Português. Quando novamente corrermos a App, ele detectará o Computador em Português e irá apresentar os textos em Português.

Agora é só repetir o processo para as restantes línguas, criando uma pasta para cada língua (ou dialecto) sempre com um ficheiro ResourcesFiles, com os mesmos Nomes de Strings, mudando apenas o Value para o equivalente na língua correspondente.



Para simular um computador noutras línguas, acedemos às Opções Regionais, adicionamos a língua, transferimos os pacotes, e movemos para o topo para predefinir.

Não esquecer, na hora da publicação, de escolher os países para os quais traduzimos a aplicação.

Para informação adicional sobre Globalização de Apps, poderá consultar: http://msdn.microsoft.com/en-us/library/windows/apps/hh965328.aspx

* Lista de países disponíveis http://msdn.microsoft.com/en-us/library/windows/apps/jj863494.aspx)

** Para a lista de nomenclaturas, poderá usar a tabela disponível em http://msdn.microsoft.com/en-us/library/windows/apps/jj657969.aspx

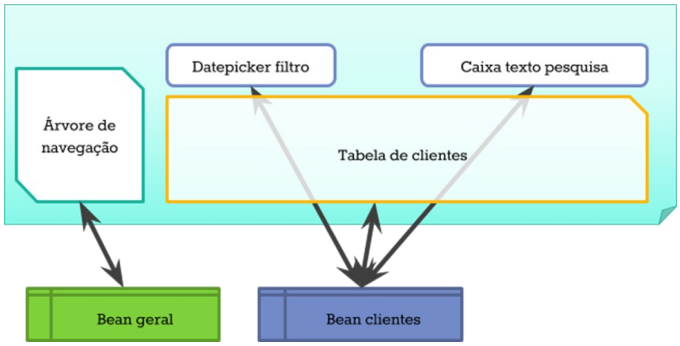
AUTOR

Escrito por Daniel Marques

JSF - Parte 1

O JSF (JavaServer Faces) é uma especificação para criar interfaces gráficas de aplicações web. É um standard do mundo Java EE, atualmente na versão 2.2. A ideia base é, após alguns anos de evolução (esta especificação nasceu em 2004), sistematizar o que as interfaces web têm em comum, apoiando o programador nesse sentido. Com relativa facilidade permite desenvolver interfaces web ricas e interativas (cujas aplicações são denominadas RIA - Rich Internet Application). É server-side e totalmente orientada ao componente: para além de uma série de componentes (abstrações dos componentes HTML), prevê validadores, gestão de estado, conversores, modelo de eventos, etc.

Entenda-se por componente (UI) um 'pedaço' reutilizável da interface (ex: dropdown box).



Além disso, o JSF, considera capacidades de internacionalização, templating, acessibilidade, navegação, Ajax, entre outras. É de alto nível, pelo que o modelo HTTP, servlets e outros os conceitos da página são implícitos.

Para clarificar melhor o enquadramento da especificação JSF, há que considerar que:

O que o JSF não é:

- Uma framework ou biblioteca
- Uma alternativa ao JSP

O que o JSF é:

- Uma especificação/standard
- Uma alternativa ao Struts (embora com objetivos e filosofias diferentes)

Sendo o JSF "apenas" uma especificação, as suas duas implementações mais comuns (essas sim, as frameworks) são o Apache MyFaces e o Oracle Mojarra.

Tanto o JSF como o Struts pertencem ao mundo Java e ambos apoiam o desenvolvimento de interfaces gráficas web. A maior diferença é que o JSF foi pensado para aplicações e o Struts para sites. Apesar do JSF, poder servir para websites, esse não é o seu uso mais comum (exceto quando se tratam de sites muito orientados à tarefa, com necessidades de componentes ricos e variados e diversas interações Ajax).

	JSF	Struts
Tipo	*Component framework*	*Action framework*
Orientado a	Componente	Página
Modelo HTTP	Implícito (alto nível)	Explícito (baixo nível)
Ideal para	*Web app*, *backoffice* (tarefa)	*Website*, *intranet* (informação)
Palavras-chave	CRUD, Ajax, interagir, fazer, RIA, SPI, controlos, funcionalidade, etc.	Navegar, consultar, *permalink*, *refresh*, *back*, ler, etc.

O facto de ser server-side resulta em que os componentes sejam codificados numa linguagem, neste caso markup XHTML (facelets), que depois é convertida para HTML, ao ir para o cliente, o browser. O conceito de facelet é importante na medida em que substitui os já velhos JSP, tornando-os obsoletos. Sendo MVC, não é suposto haver código Java nesses facelets.

```
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0
                                    Transitional//EN'
    'http://www.w3.org/TR/xhtml1/DTD/xhtml1-
                                    transitional.dtd'>
<html xmlns='http://www.w3.org/1999/xhtml'
      xmlns:f='http://java.sun.com/jsf/core'
      xmlns:h='http://java.sun.com/jsf/html'>
   <h:head>
      <title>Boa tarde</title>
   </h:head>
   <body>
      <h:inputSecret redisplay='false' value='#
{utilizadorBean.password}' />
      <h:outputLink id='link1' value='http://
                          www.google.com/search'>
         <f:param name='q' value='#
{utilizadorBean.musicaFavorita' />
         <h:outputText value='Pesquisar
                                    no Google' />
      </h:outputLink>
   </body>
</html>
```

> ***É server-side e totalmente orientada ao componente: para além de uma série de componentes (abstrações dos componentes HTML), prevê validadores, gestão de estado, conversores, modelo de eventos, etc...***

> ***O JSF (JavaServer Faces) é uma especificação para criar interfaces gráficas de aplicações web.***

O JSF considera 3 taglibs de base: core, html e ui (veremos concretamente o que são no próximo artigo desta série). É facilmente extensível com as chamadas 'bibliotecas de componentes', entre as quais constam o *PrimeFaces*, o *RichFaces* e o *ICEFaces*. O PrimeFaces é uma excelente aposta, dado que contém mais de 100 componentes, com imensas funcionalidades, entre as quais excelentes capacidades Ajax e skinning (mais de 30 temas). Entre os componentes, está o calendário, o rich text editor, o wizard, o menu, a toolbar, a tabela, a árvore, etc. Inclui a biblioteca jQuery assim como a Atmosphere (PrimePush).

Existe ainda um grande suporte por parte da comunidade, desde fóruns, demos, samples, livros, etc. Tem como irmão mais novo o PrimeFaces Mobile, que como o nome indica, oferece componentes para o mundo móvel.

Assim em resumo, e de forma a aguçar a curiosidade do leitor para os próximos artigos sobre esta especificação, as principais vantagens do JSF resumem-se principalmente a:

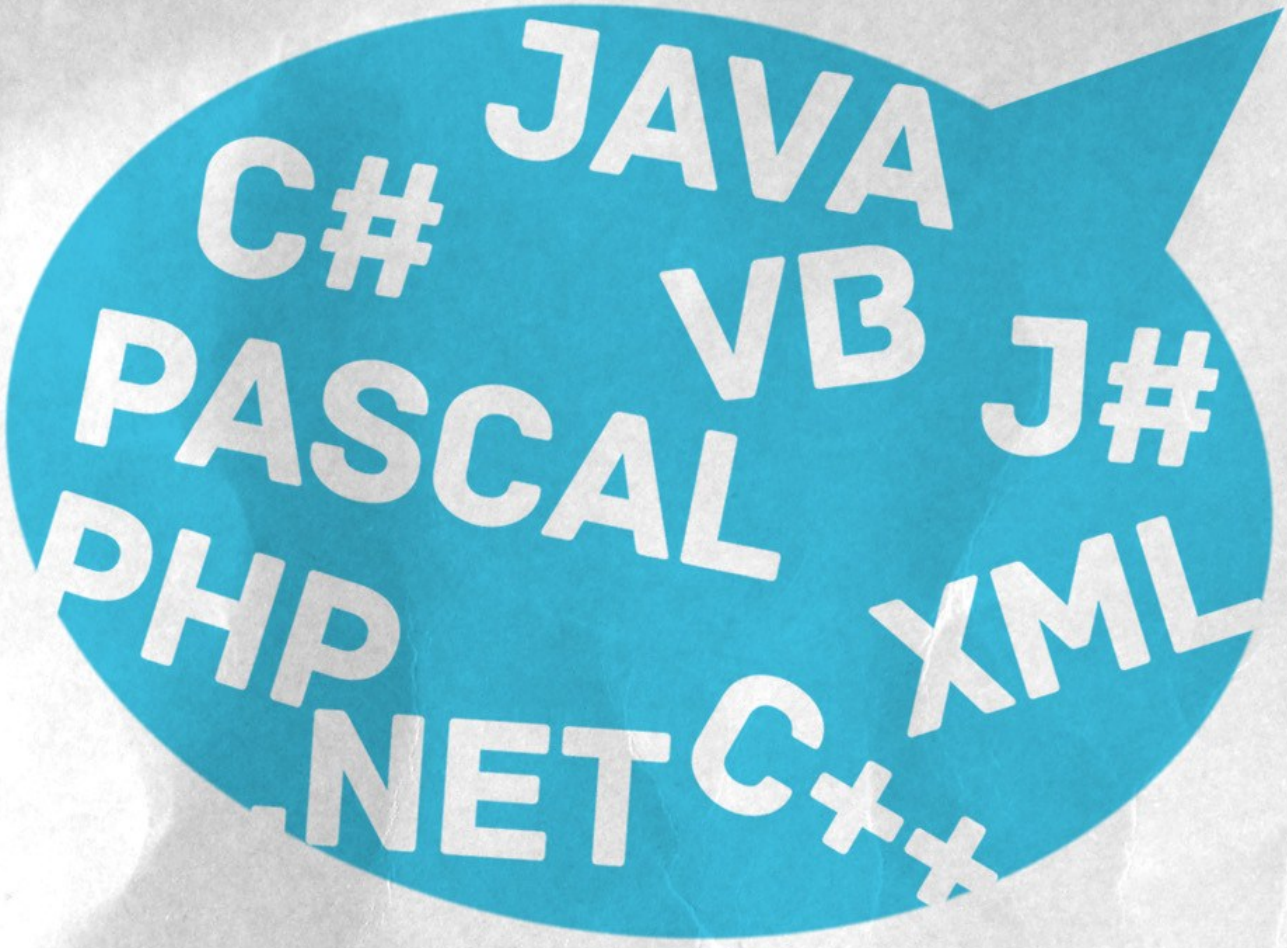
- Componentes fáceis de usar e muito parametrizáveis, reduzindo assim a necessidade de recorrermos a JavaScript;
- Fácil criação de componentes personalizados (embora seja raramente necessário, dada a oferta de bibliotecas de componentes);
- Fácil transferência de dados entre aplicação e UI;
- Fácil gestão do estado da página (alta abstração);
- Modelo de eventos muito simples;
- Possibilidade de uso de anotações Java em vez de ficheiros XML para configuração (mais contextualidade e encapsulação);
- Convention over configuration: muita coisa não precisa de ser configurada, assumindo valores por omissão.

Estas vantagens serão entendidas mais facilmente nos próximos artigos. Em conclusão, é certo que inicialmente é estranho lidar com o facto de conceitos HTTP/HTML serem tão abstraídos e de alto nível. Mas após vencer este pequeno desafio, a velocidade de desenvolvimento aumenta largamente e o leitor dará por si a criar aplicações muito complexas em relativamente pouco tempo, principalmente quando começar a criar e reutilizar os seus próprios componentes...

AUTOR

Escrito por Luís Soares

Formado em Engenharia Informática e de Computadores no Instituto Superior Técnico (Licenciatura e Mestrado). Sou *web developer, _tendo já colaborado em projetos de telecomunicações e dos _media*. Gosto de linguagens de alto nível, de reutilizar código, de refactoring para simplificar. Gosto de ensinar. Escrevi um livro sobre jQuery (goo.gl/nw2Zb).
Os meus contactos estão em cv.luissoares.com, para qualquer dúvida sobre o artigo ou outra informação.



ENTÃO, SÓ FALAS EM CÓDIGO?

TEMOS O REMÉDIO CERTO PARA TI!

portugal-a-programar.pt

A MAIOR COMUNIDADE PORTUGUESA DE PROGRAMAÇÃO, APARECE!

COLUNAS

C# - Os Perigos Das Estruturas Mutáveis

C# - OS PERIGOS DAS ESTRUTURAS MUTÁVEIS

Dada a seguinte estrutura:

```
public struct S1 : IDisposable
{
    public int Value { get; private set; }

    public S1(int value) : this()
    {
        this.Value = value; }

    public void SetValue(int value)
    {
        this.Value = value; }

    public void Dispose()
    {
        Console.WriteLine("Disposing: {0}",
                                    this.Value);
        this.Value = 0; } }
```

Qual é o resultado da execução do seguinte código?

```
using (var s1 = new S1(1))
{ s1.SetValue(-1); }
Console.WriteLine();

var s2 = new S1(2);
using (s2) { s2.SetValue(-1); }
Console.WriteLine("Disposed: {0}", s2.Value);
Console.WriteLine();

var s3 = new S1(3);
Try { s3.SetValue(-1); }
Finally { s3.Dispose(); }
Console.WriteLine("Disposed: {0}", s3.Value);
Console.WriteLine();

var s4 = new S1(4);
Try { s4.SetValue(-1); }
Finally { ((IDisposable)s4).Dispose(); }
Console.WriteLine("Disposed: {0}", s4.Value);
Console.WriteLine();
```

Resultado

O resultado da execução é:

```
Disposing: -1
Disposing: 2
Disposed: -1
Disposing: -1
Disposed: 0
Disposing: -1
Disposed: -1
```

Explicação

Para entender o que se está aqui a passar é necessário ter em mente que as estruturas (**struct**) são tipos valor, o que quer dizer que quando se copia um estrutura não se está apenas a copiar o se endereço no *heap*, mas está-se a copiar toda a estrutura. No primeiro caso, o compilador está a lidar apenas com a variável **s1**. No entanto, dado o escopo da variável não é possível validar que o valor final de **s1.Value** é **0**. No segundo caso, como se está a usar na instrução **using** um valor previamente criado, o compilador usa uma cópia desse valor para no final invocar o método **Dispose**. A razão para a utilização desta cópia é para evitar que o valor do escopo da instrução **using** seja alterado. E como se tratada de um tipo valor, é feita uma cópia integral do objeto o que faz com que o objeto contido na variável s2 não seja aquele a que foi invocado o método Dispose mas sim a cópia feita anteriormente. É por esta razão que o valor do campo Value do objeto em que foi invocado o método Dispose ainda é 2 e o valor do campo Value do objeto contido na variável s2 ainda é -1. O terceiro caso é um caso simples em que não existe qualquer "truque". No quarto caso existe uma operação explícita de *cast* de um tipo valor para uma interface. Esta operação de *cast* é feita à custa de uma operação de *boxing* do objeto do tipo valor (que consiste em alocar memória no *heap* e copiar para lá o valor do objeto) e posterior acesso como se tratando do objeto de um tipo referência. É por existir esta cópia no troço **finally** que o objeto em que está a ser invocado o método **Dispose** não é o mesmo contido na variável **s4** mas uma cópia feita no momento da invocação do método **Dispose**, pelo que, a variável **s4** mantém o estado que tinha antes da invocação feita à sua cópia.

Conclusão

Os tipos valor, nomeadamente as estruturas (**struct**), têm algumas vantagens (como por exemplo alocação no *stack* evitando o peso do GC (grabage collector)) mas quando se altera o seu estado interno pode ter consequências imprevisíveis.

Recursos

[To box or not to box, that is the question]

AUTOR

Escrito por Paulo Morgado
Bacharel em Engenharia Electrónica e Telecomunicações (Sistemas Digitais) pelo Instituto Superior de Engenharia de Lisboa e Licenciado em Engenharia Informática pela Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa exerce variadas funções relacionadas com o desenvolvimento, distribuição e manutenção de software há mais de 10 anos. Participa em diversas comunidades nacionais e internacionais (pontoNETpt, NetPonto, SharePointPT, SQLPort, Portugal-a-Programar, CodeProject, CodePlex, etc.). Pelo seu contributo para com estas comunidades, a Microsoft premeia-o com o prémio MVP (C#) desde 2003. É ainda co-autor do livro "LINQ Com C#" da FCA. http://PauloMorgado.NET/ - @PauloMorgado

Media Partners da Revista PROGRAMAR

Análises

Técnologias de Programação de Jogos

Gestão Moderna de Projetos - Melhores Técnicas e Práticas (7. Edição Atual. e Aument.)

Técnologias de Programação de Jogos

Título: Tecnologias de Programação de Jogos

Autor: José Braga de Vasconcelos / Nuno Magalhães Ribeiro

Editora: FCA - Editora de Informática, Lda.

Páginas: 320

ISBN: 978-972-722-782-2

Comecei a leitura deste livro com um pé atrás. No índice era mencionado XNA (coisa que a meu ver cria maus hábitos aos iniciantes) e assim um livro que à primeira vista me parecia ser um livro de iniciação rapidamente me surpreendeu pela positiva, não só não era o caso como era um excelente ponto de referência acompanhado de exemplos de várias linguagens e ferramentas mais usadas pela indústria de entretenimento.

Após uma análise atenta, vejo um livro que se destaca pela divisão em três partes de forma a abordar o leitor independentemente da sua formação específica. É um livro que como muitos outros, inicia com uma introdução histórica de referência que tem sempre o seu interesse, um guia organizacional (a meu ver de leitura obrigatória, mesmo para o programador experiente) e uma compilação de tutoriais de introdução às principais técnicas e ferramentas atualmente mais utilizadas na indústria. Com uma estrutura concisa e de fácil leitura, apresenta ainda um sistema de diagramas e pequenos exemplos de código, claros o suficiente para facilmente dispensar o uso da cor sem comprometer a compreensão dos mesmos. Chamou-me em especial à atenção as notas de leitura recomendada no final de cada capítulo, como uma ideia brilhante, dado o quão vasto é o tema de programação de jogos.

Este é um livro que não só beneficia quem está agora a iniciar programação, graças a uma descrição pormenorizada do necessário a uma fundação sólida e funcional, mas também um programador experiente que podia arrancar da página 77 em diante que continuava a ter um excelente livro a rever no início de cada novo projeto; isto sem querer menosprezar os restantes capítulos importantes ao iniciante, pois apesar deste já saber como iniciar um projeto da perspetiva técnica assim como também poder preferir uma linguagem ou ferramenta não abrangida neste livro, as regras de organização e manutenção assim como a criação do gdd (game development document) e diagramas estruturais a cada etapa do projeto é algo da maior importância e que deve ser estudado ao pormenor antes de se iniciar um projeto, especialmente se tivermos em conta um trabalho em equipas multi-disciplinares, onde a interação entre elementos de áreas distintas é fundamental para o sucesso. Em suma, é um livro que devia acompanhar todos os elementos de uma equipa, independentemente da sua área funcional, mesmo já tendo sido lido.

Na minha opinião, este livro seria também, caso os autores assim o entendam, uma excelente oportunidade para iniciar uma coletânea. Como já mencionei, o tema da programação de jogos é demasiado vasto para ser abrangido na sua totalidade num único livro e, como tal, fico na esperança que os autores venham a lançar outros, da mesma qualidade, onde possam aprofundar cada uma das áreas em específico, como por exemplo a criação de conteúdos e a sua importância neste tipo de projetos assim como a forma de interação entre as equipas artísticas e programadores, pois a otimização e troca de informação entre ambas a fim de contornar limitações é fundamental. Outro exemplo seria como organizar um projeto orientado a multi-plataforma e respetivo reaproveitamento de código, onde a escolha do IDE e/ou as plataformas -alvo são grandes condicionantes. Outro ponto também importante é a escolha das bibliotecas a usar caso assim o leitor o entenda e a escolha das mesmas em função das licenças, pois irão condicionar a do produto final. Outros pontos seriam cálculos de física por software ou por hardware, a escolha de protocolos de comunicação em rede, vantagens e limitações que cada um tem dependendo do projeto, e como estes, há muitos outros temas que podiam ser aprofundados numa coletânea de qualidade superior onde este seria o primeiro.

AUTOR

Escrito por Ricardo Velasquez Lé
Técnico de informática programador crossplatorm e expert linux por formação profissional.
Investigador independente de micro-electrónica como auto-didacta e presentemente core engine developer para a Digital Soul Games (DSG) .

Review

Gestão Moderna de Projetos - Melhores Técnicas e Práticas (7. Edição Atual. e Aument.)

Título: Gestão Moderna de Projetos - Melhores Técnicas e Práticas (7.ª Edição Atual. e Aument.)

Autor: António Miguel

Editora: FCA

Páginas: 504

ISBN: 978-972-722-766-2

Uma leitura agradável e de extrema utilidade, tanto para estudantes dos diversos cursos onde tenham disciplinas que impliquem a gestão dos seus projectos académicos, como a própria disciplina de Metodologias de Projecto e Projecto (nos seus diversos níveis), como para programadores e gestores de projecto.

Aborda de forma simples e apelativa todos os aspectos da gestão de um projecto, fazendo uma introdução cativante e interessante.

Ao ler pensei “Ok, já li bastante, já ouvi imenso, o meu dia a dia acaba passando por uma gestão de projecto. Mas com a leitura percebi que existe mais para ser lido e aprendido.

O papel do Gestor de Projecto é subestimado e visto de forma não propriamente correcta, tal como no início do livro é clarificado de forma simples, e assertiva.

A apresentação das normas de mercado, foi uma das partes mais interessantes, pois não só dá toda a informação relevante num único local, como a explica de forma clara e coerente, sem necessidade de remeter o leitor para outras leituras complementares.

Toda a estrutura do livro, leva a que o leitor a determinado ponto chegue a ver a leitura do livro como um projecto em que o maior stakeholder é o próprio leitor, pois é ele quem beneficia da leitura.

A gestão do tempo, as questões de ética, os conflitos pessoais extrapolados para o projecto “gestão do tempo para ler o livro”, ajudam a encaixar melhor o teor do que é explicado.

Contrariamente a outras leituras anteriores, neste livro são apresentadas questões legais, particularmente pertinentes, numa altura em que “o que hoje é, ontem não era” e uma busca sobre um assunto na legislação por vezes acaba por despender demasiado esforço numa única tarefa sem conclusão à vista, quase causando falhas em cascata.

A apresentação de normas como a PMBOK e respectiva extensão para a construção civil, é particularmente útil por oferecer uma visão menos “fechada” daquilo que é a gestão de um projecto, independentemente do seu âmbito.

Ao longo de todo o livro, são apresentados todo os aspectos de uma gestão de projectos, desde o momento em que é pensado, ao momento em que é dado por encerrado, apresentando as diversas etapas, tarefas e objectivos de cada uma delas, para um sucesso na execução de um projecto.

Uma outra grande vantagem que pude constatar está no facto deste livro não se focar demasiado num software, mas ser abrangente o suficiente para o leitor, poder aplicar o que lê em qualquer software de gestão de projectos, seja ele software livre, software proprietário, software desenvolvido internamente para a entidade em questão.

Nos capítulos 22 e 23, são apresentadas de forma “leve” aquilo que por vezes são as maiores “threats” de um projecto, nomeadamente a resistência à mudança, a gestão dessa resistência, a execução da mudança que incontáveis vezes são uma das maiores dificuldades da gestão eficiente de um projecto, pois ninguém gosta de sair daquilo que é a sua “zona de conforto”, tão pouco o ser Humano que sendo uma criatura de hábitos, sente desconforto em sair da sua “zona de conforto, ou rotina”.

É de facto um livro recomendável, tanto para consulta como para estudo, a todos os estudantes dos diversos graus de ensino.

AUTOR

Escrito por Sara Santos

Licenciada em Docência, tem um grande fascínio pelas tecnologias de informação e comunicação. Desenvolve de momento aplicações informáticas ligadas à educação, tendo já participado em projectos de programação e tradução, incluindo a gestão para língua portuguesa do software Sahana-Eden, cujo projecto de tradução gere, desde o seu começo até ao momento.

COMUNIDADES

Comunidade NetPonto – Leitor de QRCode para Windows Phone

LEITOR DE QRCODE PARA WINDOWS PHONE

Este artigo tem como objetivo criar um leitor de QRCode para aplicações Windows Phone.

O código QR ou QRCode como é mais conhecido é um código de barras bidimensional que pode ser lido facilmente por um dispositivo móvel que contenha câmara fotográfica. O código pode representar texto, um url, número de telefone, localização georreferenciada, um e-mail, um contato ou um sms. O QRCode é muito comum em revistas, propaganda, cartões de visitas, entre outros.

Em Windows Phone é possível fazer o reconhecimento de QRCode usando várias bibliotecas:

- ZXing.Net
- MessagingToolkit.Barcode
- Silverlight_ZXing_Core

Neste exemplo vamos apenas usar a biblioteca ZXing.Net e a biblioteca Silverlight_ZXing_Core. Num teste realizado MessagingToolkit.Barcode apresentou-se ser lenta.

Comecemos por desenhar a interface com o utilizador!

Vamos ter um Canvas cujo background vai ver VideoBrush com o conteúdo que a câmara está a captar no momento:

```
<phone:PhoneApplicationPage
    x:Class="PhoneApp.MainPage"
    xmlns="http://schemas.microsoft.com/
                    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
                                winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.
              Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.
                Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/
                           expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/
                        markup-compatibility/2006"
    mc:Ignorable="d"
    DataContext="{Binding Main, Source=
                        {StaticResource Locator}}"
    FontFamily="{StaticResource
                          PhoneFontFamilyNormal}"
    FontSize="{StaticResource
                           PhoneFontSizeNormal}"
    Foreground="{StaticResource
                          PhoneForegroundBrush}"
    SupportedOrientations="Portrait"
                          Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

  <Grid x:Name="LayoutRoot"
                        Background="Transparent">
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="100" />
    </Grid.RowDefinitions>

    <Image Name="resulImage"
                              Visibility="Collapsed"/>
    <Canvas x:Name="viewfinderCanvas">

      <!--Camera viewfinder -->
      <Canvas.Background>
        <VideoBrush x:Name="viewfinderBrush">
          <VideoBrush.RelativeTransform>
            <CompositeTransform
                x:Name="viewfinderTransform"
                CenterX="0.5"
                CenterY="0.5"
                Rotation="90"/>
          </VideoBrush.RelativeTransform>
        </VideoBrush>
      </Canvas.Background>
    </Canvas>
    <StackPanel Grid.Row="1" Margin="20, 0">
      <TextBlock x:Name="tbBarcodeType" Visibi-
lity="Collapsed" FontWeight="ExtraBold" />
      <TextBlock x:Name="tbBarcodeData"
FontWeight="ExtraBold" TextWrapping="Wrap" />
    </StackPanel>
  </Grid>
</phone:PhoneApplicationPage>
```

Em code behide iremos ter:

```
public partial class MainPage
{
    private IBarcodeReader _barcodeReader;
    private PhotoCamera _phoneCamera;
    private WriteableBitmap _previewBuffer;
    private DispatcherTimer _scanTimer;

    public MainPage()
    {
        InitializeComponent();
    }

    protected override void OnNavigatedTo
                            (NavigationEventArgs e)
    {
        // Inicializa o objeto câmara
        _phoneCamera = new PhotoCamera();
        _phoneCamera.Initialized +=
                                Camera_Initialized;
        tbBarcodeType.Text = string.Empty;
        tbBarcodeData.Text = string.Empty;
        CameraButtons.ShutterKeyHalfPressed +=
               CameraButtons_ShutterKeyHalfPressed;

        //Permite mostrar o resultado da câmara na
        //interface com o utilizador
        viewfinderBrush.SetSource(_phoneCamera);


        // Este time irá permiter fazer o scan do
       //"buffer" da câmara a cada 250ms e irá
       //verificar se existe algum código de
       //barras presente
```

```
        _scanTimer = new DispatcherTimer {
      Interval = TimeSpan.FromMilliseconds(250) };
        _scanTimer.Tick += (o, arg) =>
                                ScanForBarcode();
        base.OnNavigatedTo(e);
    }

    protected override void OnNavigatingFrom
                    (NavigatingCancelEventArgs e)
    {
        _scanTimer.Stop();


        if (_phoneCamera != null)
        {
            _phoneCamera.Dispose();
            _phoneCamera.Initialized -=
                              Camera_Initialized;
            CameraButtons.ShutterKeyHalfPressed -=
              CameraButtons_ShutterKeyHalfPressed;
        }
    }

    private void BcReader_ResultFound(Result obj)
    {
        // se um novo código de barras é
        //encontrado o dispositivo irá vibrar
        if (!obj.Text.Equals(tbBarcodeData.Text))
        {
            VibrateController.Default.Start
                 (TimeSpan.FromMilliseconds(100));
            tbBarcodeType.Text =
                      obj.BarcodeFormat.ToString();
            tbBarcodeData.Text = obj.Text;

            _scanTimer.Stop();
            viewfinderCanvas.Visibility =
                             Visibility.Collapsed;
            resulImage.Visibility =
                              Visibility.Visible;
            resulImage.Source = _previewBuffer;
        }
    }

    private void Camera_Initialized(object sender,
              CameraOperationCompletedEventArgs e)
    {
        if (e.Succeeded)
        {
            this.Dispatcher.BeginInvoke(delegate
            {
                _phoneCamera.FlashMode =
                                    FlashMode.On;

                _previewBuffer = new
                              WriteableBitmap((int)
       _phoneCamera.PreviewResolution.Width, (int)
           _phoneCamera.PreviewResolution.Height);

                _barcodeReader = new BarcodeReader();

                // por omissão o BarcodeReader irá
// fazer "scan" de todos os tipos de códigos de
// barras, para limitar o tipo de código de barra
// deveremos definir quais são os formatos
// possíveis
    var supportedBarcodeFormats = new
     List<BarcodeFormat> { BarcodeFormat.QR_CODE };

_barcodeReader.Options.PossibleFormats =
                            supportedBarcodeFormats;

                _barcodeReader.ResultFound +=
                              BcReader_ResultFound;
                _scanTimer.Start();
            });
        }
        else
        {
            Dispatcher.BeginInvoke(() =>
            {
                MessageBox.Show("Impossível
                          inicializar a câmara.");
            });
        }
    }


    private void CameraButtons_
  ShutterKeyHalfPressed(object sender, EventArgs e)
    {
        _phoneCamera.Focus();
    }

    private void ScanForBarcode()
    {
        // aplica o focus para obter melhor imagem
        if (_phoneCamera.IsFocusSupported)
        {
            _phoneCamera.Focus();
        }

        // recolhe uma imagem da câmara
        _phoneCamera.GetPreviewBufferArgb32
                            (_previewBuffer.Pixels);
        _previewBuffer.Invalidate();

        // analisa a imagem capturada
        // caso seja encontrado um código de barras
         //o evento ResultFound irá ser dispuletado
        _barcodeReader.Decode(_previewBuffer);
    }
}
```

Por fim, e com base no artigo A simple Windows Phone control for reading QR codes é possível obter através deste projeto https://github.com/jeffwilcox/wpqr-control um controlo que tem a capacidade de ler QRCode e é muito simples de usar.

Nota: Este controlo é o mesmo que é usado no Foursquare – 4th & Mayor

Depois de fazer o download ou fazer "git clone https://github.com/jeffwilcox/wpqr-control", adicione a referência deste projeto ao projeto Windows Phone App. Em seguida na página principal adicione o controlo da seguinte forma:

```
<phone:PhoneApplicationPage
    x:Class="PhoneApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/
                                    xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
                                      winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.
                  Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.
                     Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/
                                 expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/
                            markup-compatibility/2006"
    xmlns:jwqr="clr-namespace:JeffWilcox.Controls;
                    assembly=JeffWilcox.Controls.QR"
    mc:Ignorable="d"
```

LEITOR DE QRCODE PARA WINDOWS PHONE

```
    FontFamily="{StaticResource
                          PhoneFontFamilyNormal}"
    FontSize="{StaticResource
                           PhoneFontSizeNormal}"
    Foreground="{StaticResource
                           PhoneForegroundBrush}"
    SupportedOrientations="Portrait"
                           Orientation="Portrait"
    shell:SystemTray.IsVisible="True"

    SizeChanged="MainPage_OnSizeChanged">

  <Grid x:Name="LayoutRoot"
                         Background="Transparent">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <TextBlock Grid.Row="0"
                       x:Name="ApplicationTitle"/>

    <jwqr:QRCodeScanner Grid.Row="1"
                        Width="480"
                        Height="740"
                        x:Name="QRCodeScanner"
                        ScanComplete=
                    "QRCodeScanner_ScanComplete"
                        Error=
                          "QRCodeScanner_Error"/>

  </Grid>

</phone:PhoneApplicationPage>
```

E em code behide

```
using System.Windows;

public partial class MainPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    private void QRCodeScanner_ScanComplete
(object sender,
     JeffWilcox.Controls.ScanCompleteEventArgs e)
    {
        ApplicationTitle.Text = e.Result;
    }

    private void QRCodeScanner_Error(object
                    sender, JeffWilcox.Controls.
                         ScanFailureEventArgs e)
    {
        throw e.Exception;
    }

    private void MainPage_OnSizeChanged(object
                   sender, SizeChangedEventArgs e)
    {
        QRCodeScanner.Width = this.ActualWidth;
        QRCodeScanner.Height = this.ActualHeight;
    }
}
```

Em conclusão, conclui-se que é muito simples fazer a leitura de QRCode em Windows Phone existindo uma diversidade de alternativas para implementar um leitor de QRCode.

Faça o teste com o seguinte QRCode:

AUTOR

Escrito Por Sara Silva

é licenciada em Matemática – Especialidade em Computação, pela Universidade de Coimbra e é Microsoft Certified Professional Developer. Atualmente o seu foco de desenvolvimento incide em Windows Phone e Windows 8 Store Apps. O seu Blog **é** www.saramgsilva.com e o twitter é @saramgsilva

No Code

Programar é a “Língua” do Século XXI

A Concepção de uma Ideia Programável

Programar é a "Língua" do Século XXI

Estamos em pleno século XXI e todos nós usamos tecnologia e informação diariamente. Usamo-las de tal modo que arrisco dizer que é impossível vivermos sem esta presença nas nossas vidas

É inevitável o uso de telemóveis, smartphones, tablets e computadores, entre milhares de outros dispositivos no nosso dia-a-dia, visto que quase todos os electrodomésticos de última geração já possuem o seu próprio sistema embutido.

Mas para que o mais comum dos mortais possa usar essa mesma tecnologia, alguém teve que desenvolver essa tecnologia e sobretudo teve que a programar.

As aplicações para smartphones, tablets, computadores e todos e mais alguns dispositivos que possamos imaginar tiveram de ser programadas por um Humano. Quantas vezes cada um de nós já deu por si a pensar que uma "determinada aplicação" seria tão útil para o decorrer do nosso dia-a-dia ou para a nosso trabalho?

Ou que aquele programa que usamos todos os dias, ficaria perfeito com esta ou aquela nova funcionalidade?

Acredito que muitos já tiveram este pensamento, mas não tiveram os conhecimentos necessários sobre programação para materializar essa mesma ideia.

Não faltam ideias a fervilhar pelo mundo fora, falta é sobretudo conhecimentos técnicos suficientes para poder colocar essas mesmas ideias em prática.

Programar não é uma ciência exacta, é sobretudo uma arte que permite dar asas à nossa imaginação. Talvez seja até demasiado arriscado dizer que programar é a forma mais completa de aprender, sim aprender, porque programar aguça a lógica do pensamento tal como a matemática o faz e isto é o mais importante no processo de aprendizagem!

Num mundo onde a tecnologia é a palavra de ordem, basicamente vivemos num mundo criado pelos programadores, pois são estes que decidem o que podemos ou não fazer em determinado dispositivo tecnológico, dispositivos esses os quais já não nos imaginamos a viver sem eles.

Recentemente temos assistido a um crescente incentivo por parte de alguns Gurus da tecnologia, tais como Mark Zuckerberg. Bill Gates e Jack Dorsey ao ensino da programação aos mais jovens, tentando fazer crer que estes serão os génios de amanhã e que serão capazes de comunicar seja com quem for através da programação.

Alguns dos projectos que incentivam a programar e ensinam a programar de uma forma fácil e intuitiva são:

http://academy.dei.uc.pt/

https://www.codecademy.com/pt

www.code.org

A todos os leitores que possam estar a começar a dar os primeiros passos no mundo da programação, ou que conheçam alguém que queira entrar neste mundo, lanço-vos esse mesmo repto, o de experimentarem estes desafios e de incentivarem todos os que o queiram fazer.

Não quero estar a dizer que programar se possa tornar numa língua essencial para comunicar a curto prazo, mas sem qualquer tipo de dúvida programar está a tornar-se um recurso muito importante, poderoso e acima de tudo, está ao alcance de todos

Estará então, a Programação a tornar-se na linguagem mais importante do mundo?

"Everybody in this country should learn how to program a computer… because it teaches you how to think." – Steve Jobs

"Coding have the Magical Power to Make Your Devices do What You Want them to do!"

AUTOR

Escrito por Nuno Santos

Curioso e autodidacta com uma grande paixão pela programação e robótica, frequênta o curso de Engenharia Informática na UTAD alimentando o sonho de ainda vir a ser um bom Engenheiro Informático. Estudante, Blogger, e moderador no fórum Lusorobótica são algumas das suas actividades. Os seus projectos podem ser encontrados em: http://omundodaprogramacao.com/

A Concepção de uma Ideia Programável

Numa altura em que estamos perante o "BOOM" das aplicações para tablets e smartphones, e aproveitando o mote de um outro artigo desta edição, o "Programar é a linguagem do Século XXI", convido agora os leitores a conhecer ou a recordar uma outra vertente da programação. A fase inicial, aquela em que passamos pela concepção de uma ideia e a trabalhamos antes de a colocarmos em prática. Acredito piamente, tal como escreveu Nuno Santos, que "Quantas vezes cada um de nós já deu por si a pensar que uma "determinada aplicação" seria tão útil para o decorrer do nosso dia-a-dia?"

Se nos a consideramos útil, podem outras pessoas também considerar útil essa mesma ideia e podem mesmo vir a ser possíveis utilizadores dessa nova ideia. Mas o que precisamos para que a nossa ideia ganhe forma? E quando falo numa ideia, incluo as ideias para aplicações funcionais ou ideias para um jogo, por exemplo. E porque tanta insistência na palavra "ideia"? Porque no início do processo, uma ideia é tudo o que temos e por mais que trabalhemos ou pensemos nela, é apenas uma ideia. Antes de passarmos à acção, ou seja, ao código, temos ainda que nos preocupar em "estudar o mercado"… qual o perfil do utilizador que iremos ter? Em que poderá ele estar interessado? Em que temos que nos focar para que continuemos a garantir a sua preferência? Depois de termos todas estas variáveis em consideração, de termos feito uma observação mais a fundo, ai sim, passamos à fase de dar forma à nossa ideia, de a materializarmos e dar-lhe vida. Isto é talvez a fase da criação que todos nós programadores mais gostamos. Mas mais do que uma aplicação que seja totalmente funcional, hoje em dia e cada vez mais, o utilizador procura por aplicações simples, sem complicações. E fazendo jus aos princípios do Design, "o melhor design é o menos design possível", e a simplicidade marca pontos. Quase todos nós quando experimentamos uma aplicação ou um jogo pela primeira vez, queremos algo intuitivo, algo que fale por si e nos permita utilizar todas as funcionalidades sem ler a "Bendita parte do LEIA-ME" ou da ajuda… Assim e depois de colocarmos a nossa aplicação no Google play ou no Windows Store, podemos fazer publicidade à mesma ou podemos coloca-la como uma aplicação gratuita para que o utilizador possa experimentar e possa mais tarde contribuir para novos desenvolvimentos da mesma ou fazer o upgrade para uma versão paga (isto se assim o entendermos e quisermos ter lucro com o nosso produto). Na fase de lançamento há várias opções que podemos tomar para mostrar ao Mundo que a nossa aplicação está disponível e que pode ser uma mais valia ou um boa opção de entretenimento a quem a utiliza. Então mas e depois de criarmos uma aplicação ou o nosso jogo o que fazemos? Poderemos nós "sentarmo-nos à sombra" e esperar que os nosso utilizadores continuem fieis a um produto que continua igual com o passar do tempo? "Não" – deverá estar o leitor a pensar - "Não", concordo eu! Esta é possivelmente a fase em que mais temos que nos "preocupar" e empenhar. Se o primeiro esforço que temos é como trazer pessoas a ver e a usar a nossa aplicação, o segundo esforço e no meu entender o mais importante, é como mantermos a preferência de quem a experimentou! Num apanhado geral, há três palavras "mágicas" que devemos ter sempre em consideração. Aquisição, Retenção e Monitorização. Ou seja, na primeira fase adquirirmos utilizadores à nossa aplicação, depois de conseguirmos captar a atenção dos mesmos, deveremos conseguir retê-los e conseguir que voltem de novo a utiliza-la. Nenhum de nós gostaria de desenvolver algo que após uma ou duas utilizações caísse em esquecimento e desuso pois não? E finalmente, a fase da monitorização, que deve ser permanente. É importante não deixarmos o nosso projecto, a nossa ideia, de lado só porque já a materializamos. É importante continuarmos sempre a pensar nela, em formas de a tornar melhor e mais atractiva. Usando como exemplo um jogo e, recordando o tema do primeiro artigo que escrevi na nossa revista (edição 34), o Farmville conseguiu durante muito tempo ter a afluência que teve porque ainda hoje é um jogo que usa épocas festivas ou pequenas missões (dando pequenos prémios em caso de objectivo atingido) para prender a atenção do jogador, o que faz com que o jogador continue a voltar e em alguns casos até mesmo a investir dinheiro em créditos. Se passarmos frequentemente por todas estas fases, teremos mais probabilidades de ter uma ideia vencedora. Pensar está na alma do ser humano, ter ideias também. E cabe a cada um de nós materializar as nossas ideias. Cabe a cada um de nós lutar por essas mesmas ideias. Porque todas as aplicações e programas que usamos hoje em dia começaram por ser uma ideia, todos sem excepção! E em jeito de despedida desta edição, mais uma vez relembro as fases que este artigo quis recordar: a concepção de uma ideia, a configuração da forma e do design do projecto e a monitorização e o uso da aplicação, mas principalmente, que a materialização de uma ideia depende apenas da nossa vontade e do nosso empenho!

AUTOR

Escrito por Rita Peres

Natural de Castelo Branco, licenciou-se em Engenharia Informática pela Universidade da Beira Interior. Membro do P@P desde Janeiro de 2010.

Elege o melhor artigo desta edição

Revista PROGRAMAR

http://tiny.cc/ProgramarED43_V

Veja também as edições anteriores da Revista PROGRAMAR

42ª Edição - Setembro 2013

41ª Edição - Junho 2013

40ª Edição - Abril 2013

39ª Edição - Fevereiro 2013

38ª Edição - Dezembro 2012

37ª Edição - Outubro 2012

e muito mais em ...

www.revista-programar.info

DUVIDAS?

IDEIAS?

AJUDAS?

PROJECTOS?

portugal-a-programar.org