PROGRAMAR

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

EDIÇÃO #42 - SETEMBRO 2013

ISSN 1647-0710

A PROGRAMAR

DEBUG DE APLICAÇÕES EM C

ARDUINO ACCIONAMENTO E CONTROLO DE SERVOS VIA TECLADO

INTRODUÇÃO AO WEB2PY

LISTAS SIMPLESMENTE LIGADAS E EXEMPLO DE IMPLEMENTAÇÃO EM C

PASCAL TIPO DE DADOS VARIANT E TIPOS GENÉRICOS

COLUNAS

EVENTOS & HANDLERS **VISUAL(NOT)BASIC**

COMUNIDADES

IMPLEMENTANDO PUBLICIDADE USANDO NOKIA NAX EM APLICAÇÕES WINDOWS PHONE **NETPONTO**

NO CODE

SER OU SER RECONHECIDO PELA GOOGLE **DISPOSITIVO ANDROID PREMIUM**

NAVICAT 11

ANÁLISES

WINDOWS SERVER 2012 CURSO COMPLETO

REDES DE COMPUTADORES CURSO COMPLETO

EDITORIAL

EQUIPA PROGRAMAR

Coordenador
António Santos

Editor
António Santos

Design
Sérgio Alves
Twitter: @scorpion_blood

Redacção
António Pedro Cunha
António Santos
Cristiano Ramos
Igor Nunes
Nelson Lima
Nuno Santos
Rita Peres
Sara Silva
Sérgio Ribeiro
Telmo Marques

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

Contacto
revistaprogramar@portugal-a-programar.org

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

ISSN
1 647-071 0

try {Thread.sleep(1000); }

Como em tudo na programação, por vezes acontecem *exceptions*, que não antecipamos, nem pensamos que pudessem acontecer.

Assim na Revista PROGRAMAR, aconteceu uma no passado mês de Agosto, pelo que a edição que seria publicada o mês passado, apenas agora é trazida até vós, com grande pena nossa. Aconteceu uma *exception,* para a qual não estávamos preparados, mas longe de isso significar o fim da revista, significa uma melhoria para a continuidade.

Antes de me desculpar pelo incidente e evitando politizar um texto que não é politizável, prefiro transmitir a todos os leitores que a revista se encontra de saúde, e que esta *exception* já tem um "*handler"* para que não volte a acontecer.

Recordo-me de ler, que todas as "ameaças" devem ser convertidas em oportunidades. No caso da revista, esta "*exception*" será convertida numa oportunidade de melhorar, e também de alterar um pouco os cronogramas, uma vez que a edição de agosto foi "suprimida", trazemos até vos, agora a edição de Setembro, e a próxima será 60 dias após esta, ou seja em Novembro (bem antes do Natal).

Dito isto e dada a devida explicação, peço desculpa a todos vós, leitores, autores, parceiros, por este incidente, que alterou o cronograma da revista. Não foi algo que pudéssemos prever, foi uma situação completamente imprevista, mas para a qual já nos preparamos para que nunca mais volte a acontecer.

A todos vocês que fielmente seguem a revista, o nosso muito obrigado.

Antes de terminar o editorial desta edição, gostava de deixar uma breve palavra de agradecimento a todos aqueles que combatem o flagelo dos incêndios, lembrando os que não voltaram do cumprimento do dever, não esquecendo todos os outros que ainda combatem as chamas e deixando uma especial lembrança todos aqueles que apesar de terem as suas vidas profissionais no mundo da tecnologia, (alguns até participam regularmente na revista, tanto na autoria de artigos, como nas tarefas de staff), são também Bombeiros Voluntários, de quem não nos podemos esquecer.

Obrigado a todos vós leitores, por seguirem esta nossa revista, Obrigado a todos os Bombeiros que combatem os Incêndios Florestais em Portugal.

Até uma próxima edição,

António Santos

ÍNDICE

TEMA DE CAPA

A PROGRAMAR

COLUNAS

ANÁLISES

COMUNIDADES

NO CODE

EVENTOS

06 de Setembro CoderDojo LX #18 22 de Junho

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

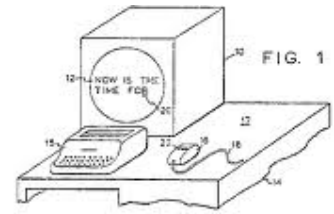
NOTICIAS

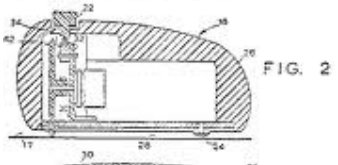
Morreu o pai do rato e do hipertexto

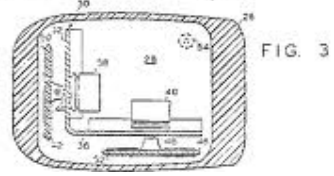
Douglas C. Engelbart, inventor do rato para computador e pioneiro da internet, morreu na terça-feira à noite, na Califórnia, Estados Unidos. Tinha 88 anos.

Engelbart tinha problemas de saúde e faleceu tranquilamente durante o sono, segundo a filha, Christina.

O momento alto da sua carreira aconteceu em 1968, em São Francisco, numa conferência onde apresentou ao mundo um dispositivo em forma de cubo, com dois discos, a que chamou "indicador de posição X-Y para um sistema de exibição", mais conhecido como o rato para computador.







Nessa mesma apresentação, o cientista informático do Instituto de Investigação de Stanford também realizou a primeira vídeoconferência da história, ao falar com um colega a 50 quilómetros de distância.

Como se ainda não bastasse, ainda explicou à audiência composta por peritos em tecnologia a teoria de como páginas com informação podiam estar relacionadas através de "hiperlinks" de texto, ideia que mais tarde seria um dos pilares da arquitectura da internet.

Os dólares não moviam Douglas C. Engelbart, mas sim o seu trabalho de pesquisa relacionado com a interacção entre o homem e os computadores.

Nunca recebeu direitos de autor pela invenção do rato, que foi patenteando pelo Instituto de Investigação de Stanford e mais tarde licenciado à Apple.

Fonte: Radio Renascença / Sapo

Supercomputador Raijin entra em funcionamento na Austrália

Entrou já em funcionamento na Austrália o supercomputador baptizado como Raijin, coincidindo com a estreia do centro de computação National Computational Infrastructure (NCI) localizado na Australian National University (ANU).

Segundo a notícia do site Ubergizmo, o supercomputador Raijin consegue, por hora, realizar o mesmo número de cálculos que levaria cerca de 20 anos a sete mil milhões de pessoas munidas de calculadoras.

Este supercomputador é o maior da Austrália, e irá permitir aos cientistas processarem grandes volumes de dados que poderiam levar anos a ficarem concluídos.

O Raijin é capaz de atingir uma velocidade de 1,2 petaflops.

Em comparação, o Tianhe-2, o supercomputador chinês mais rápido do mundo, pode atingir uma velocidade de 33,86 petaflops.

Fonte: PCGuia /Sapo

App nacional põe throttle de Flight Simulator no smartphone

Chama-se Flight Simulator Throttle e foi desenvolvida pelo português Ricardo Sequeira. Esta app para Android permite controlar o acelerador do avião a partir do smartphone e está disponível para download apenas no site do autor.

Os fãs do jogo Flight Simulator para Windows PC que tenham um terminal Android, podem agora expandir os comandos do jogo para o smartphone ou tablet. Esta app permite controlar por toque o throttle do avião e comunica com o computador através de uma ligação Wi-Fi.

Smartphone com Android e computador têm de estar na mesma rede, sendo necessário correr o pequeno programa FS Control (incluído na pasta de download) no mesmo computador do Flight Simulator.

Depois de abrir a app no smartphone Android, temos de colocar o endereço IP do computador para que a ligação se inicie. Ricardo Sequeira promete colocar em breve a app na loja Google Play.

Fonte: PCGuia / Sapo

IBM assina parceria com a Câmara de Tomar para criação de pólo tecnológico

A IBM, através da sua empresa SoftInsa, assinou um protocolo com a Câmara Municipal de Tomar que tem como objectivo o trabalho em conjunto para a criação do Centro de Inovação Tecnológica (CENIT). Este pólo tecnológico será responsável pela criação de mais de 200 empregos em Tomar.

Este novo centro de inovação e tecnologia vai ser «um pólo de prestação de serviços e de manutenção de aplicações de software altamente qualificado», confirmou António Raposo Lima, presidente da IBM Portugal, à agência Lusa. Segundo o mesmo, o centro deve ser inaugurado entre Outubro e Dezembro deste ano.

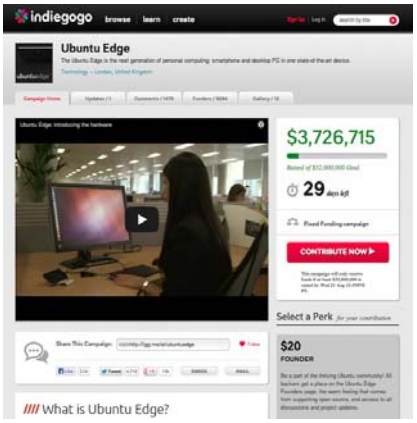
Com gestão assegurada pela SoftInsa, empresa detida pela IBM Portugal, o CENIT será o sexto na Península Ibérica. Ao contrário do que era esperado, António Raposo Lima não revelou o valor total deste investimento, mas anunciou que vão ser criados mais de 200 postos de trabalho.

«Iremos privilegiar recém-licenciados e licenciados do Instituto Politécnico de Tomar, com espacial enfoque nas áreas de gestão, tecnologias de informação e engenharia», concluiu António Raposo Lima.

Fonte: PCGuia / Sapo

Projecto Edge da Canonical já conseguiu obter 7 milhões de dólares na plataforma Indiegogo

Na semana passada foi noticiado que a Canonical espera obter através da plataforma Indiegogo cerca de 32 milhões de dólares para passar o Edge, um smartphone capaz de funcionar tanto com o sistema operativo Ubuntu como com o Android à fase de produção.



Segundo uma notícia do site Liliputing, a Canonical já conseguiu no espaço de uma semana obter 7 milhões de dólares.

No entanto, caso não seja possível obter os tais 32 milhões de dólares até ao dia 21 de Agosto, o projecto Edge será abandonado.

As especificações do Edge apontam para a presença de um ecrã de 4,5 polegadas com uma resolução de 1280×720 píxeis, de um processador quad-core, 4GB de RAM, 128GB de espaço de armazenamento interno, de uma câmara traseira de 8 Megapixéis e de uma câmara frontal de 2 Megapixéis.

Virá ainda equipado com o Bluetooth 4.0, Wi-Fi, tecnologia NFC e com acesso às redes LTE.

Fonte: PCGuia / Sapo

TEMA DE CAPA

Web Persistente: Local Storage

Web Persistente: Local Storage

Introdução

Neste artigo vamos estudar uma solução para o desenvolvimento de aplicações web mais fluídas recorrendo ao Local Storage. Esta tecnologia possibilita a persistência de dados no browser, o que permite a implementação de aplicações web que respondem visualmente de forma imediata – uma caraterística que sempre foi possível em aplicações nativas, mas que a web apenas recentemente começou a suportar.

No decorrer deste artigo introduzimos a utilização do Local Storage e analisamos em maior detalhe a técnica utilizada para tirar partido da potencialidade da tecnologia. Finalmente, é feita a apresentação dos resultados de desempenho conseguidos, capturados recorrendo à plataforma de nome AnyKB desenvolvida como prova de conceito para esse objectivo.

Buzzword: HTML5

O HTML5, para além de ser uma especificação, é uma buzzword que engloba todas as tecnologias emergentes que estão a revolucionar a Web. Apesar do nome, algumas destas tecnologias não estão relacionadas com a linguagem de marcação, mas tratam-se de APIs (Application Programming Interface) JavaScript que permitem a implementação de funcionalidades inovadoras: este é o caso do Local Storage.

A especificação W3C “Web Storage” [1] define esta API do seguinte modo:

“This specification defines an API for persistent data storage of key-value pair data in Web clients.

Na prática, Local Storage é uma API JavaScript que permite guardar de forma persistente dados no formato chave-valor. Isto é, os dados são mantidos no browser mesmo após o utilizador fechar a página web.

O principio básico do Local Storage é idêntico ao dos cookies, mas de utilização simplificada e com limites de espaço mais generosos. De momento é possível guardar cerca de 5MB por domínio ou subdomínio, estando o acesso aos dados limitado igualmente ao domínio ou subdomínio originador dos mesmos.

Apesar de recente, o suporte a esta tecnologia pelos browsers mais populares é bastante boa. O Local Storage é suportado em todas as actuais versões dos browsers Chrome, Firefox, Internet Explorer, Safari e Opera, e ainda pelas versões móveis iOS Safari, Android Browser e Blackberry Browser [1].

Introdução ao Local Storage

Aceder ao Local Storage através de JavaScript é bastante simples e directo. A especificação define a interface localStorage que contém os métodos setItem e getItem para guardar e obter dados, respectivamente. Vejamos o exemplo abaixo.

```
//Guardar
localStorage.setItem("chave", "valor");
//Obter
var valor = localStorage.getItem("chave");
```

Listagem 1: Exemplo de acesso ao interface localStorage.

Infelizmente, a persistência de objectos não é nativamente suportada, para tal é necessário serialização e desserialização dos objectos. Uma solução muito popular passa pela tradução dos objectos para JSON (JavaScript Object Notation) - uma notação para intercâmbio de dados derivada do JavaScript. Este processo é suportado pelo objecto JSON utilizando as funções stringify e parse para serializar e desserializar, respectivamente. Vejamos o exemplo abaixo, onde aproveitámos para estender os métodos setItem e getItem para suportar a persistência de objectos.

```
//Guardar referência às funções originais
Storage.prototype._setItem =
Storage.prototype.setItem;
Storage.prototype._getItem =
Storage.prototype.getItem;

//Override da função setItem
Storage.prototype.setItem = function (key,
value) {
    //Se for um objecto, serializar
    if (value instanceof Object) {
        value = JSON.stringify(value);
    }
    //Guardar
    this._setItem(key, value);
}

/* Override da função getItem
* O parâmetro isObject é utilizado para
* indicar se o valor a devolver é um
* objecto
*/
Storage.prototype.getItem = function (key,
isObject) {
    //Obter
    var item = this._getItem(key);
    //Se for um objecto, desserializar
    if (isObject) {
        item = JSON.parse(item);
    }
    return item;
}

//Guardar um objecto
var obj = { foo: "bar" };
localStorage.setItem("chave", obj);
```

WEB PERSISTENTE: LOCAL STORAGE

```
//Obter um objecto
var obj =
localStorage.getItem("chave", true);
```

Listagem 2: Persistência de objectos serializados em Local Storage.

Conseguimos assim uma interface de armazenamento persistente capaz de suportar não só tipos de dados primitivos mas também objectos.

De seguida analisamos o contexto em que pretendemos aplicar esta tecnologia.

O problema da Web não-persistente

As aplicações web executadas no browser têm como característica a completa dependência de um servidor que fornece toda a lógica da aplicação. Isto significa que em cada acesso o browser tem de descarregar todos os elementos da aplicação, independentemente de qualquer acesso anterior. Esta redundância de dados que circulam na rede agrava-se ao verificarmos que a componente visual (HTML, CSS & JavaScript) não serve qualquer interesse ao servidor. O ideal seria persistir estes elementos no browser no primeiro acesso, eliminando a necessidade de nova transferência em acessos posteriores.

Os sistemas de cache implementados pelos browsers assistem ao guardar localmente alguns elementos, no entanto apenas reduzem a quantidade de informação trocada, não eliminando por completo a transferência da componente visual da aplicação. Adicionalmente, não existe uma interface programática que permita a gestão deste recurso.

A utilização de AJAX (Asynchronous Javascript and XML) é outra técnica que permite a redução da quantidade de dados transferidos ao possibilitar ligações assíncronas ao servidor. É possível utilizar esta técnica para, por exemplo, descarregar a componente visual da aplicação na abertura da página web, e posteriormente utilizar ligações AJAX para obter apenas o resultado do servidor. No entanto introduz-se um novo problema: cada nova abertura da aplicação torna-se mais lenta e pesada. Para além disso, o problema inicial mantém-se pois continuamos a depender do servidor para fornecer a componente visual da aplicação em cada novo acesso.

O Local Storage vem oferecer uma solução mais completa ao permitir que o programador persista no browser todos os elementos de apresentação da aplicação, eliminando completamente a transferência desta componente a partir do servidor. Exceptuam-se apenas dois casos incontornáveis: quando ainda não foi persistida e quando existem actualizações. Pode ser feita uma analogia a aplicações nativas: o primeiro acesso corresponde à "instalação", existindo "actualizações" posteriores apenas quando necessário. Após "instalada", a aplicação comunica com o servidor utilizando AJAX, consistindo a comunicação apenas da informação relevante ao pedido do utilizador, geralmente estruturada recorrendo a JSON ou XML.

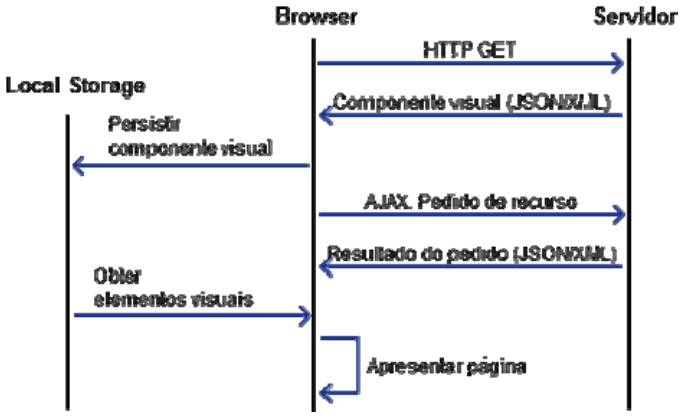


Figura 1: Diagrama da interacção entre browser e servidor utilizando Local Storage para persistência da componente visual da aplicação.

Exemplo de Implementação

Como demonstração concreta deste conceito vamos implementar uma aplicação simples, recorrendo à linguagem PHP para a implementação de um webservice.

O seguinte diagrama auxilia na contextualização dos ficheiros que compõem este exemplo, indicando a posição final dos mesmos após a primeira abertura da plataforma pelo browser.

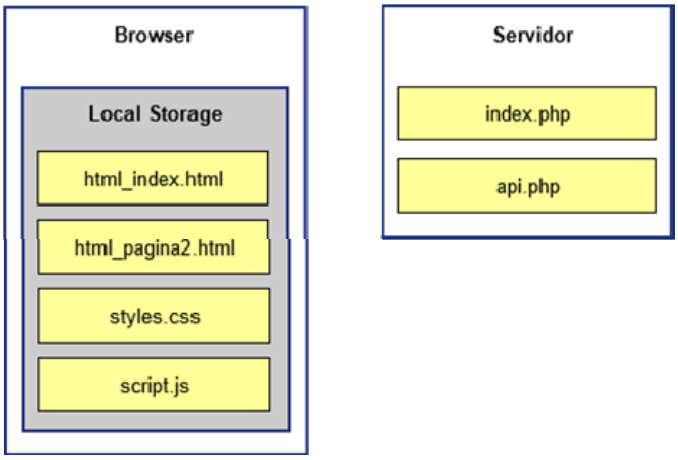


Figura 2: Contextualização dos ficheiros que compõem o exemplo de implementação.

Todo o código de seguida apresentado foi cuidadosamente documentado para melhor compreensão.

Ficheiro index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>Aplicação Exemplo</title>
<meta http-equiv="Content-Type"
                content="text/html; charset=UTF-8">
  <!-- Carregar jQuery -->
<script src="http://code.jquery.com/
                              jquery-2.0.0.min.js">
```

```
</script>
<!-- Carregar script com a lógica de
apresentação da aplicação -->
<script src="script.js"></script>
  <style type="text/css"></style>
 </head>
 <body>
 </body>
</html>
```

Listagem 3: Ficheiro index.html. Ponto de entrada para a aplicação.

Ficheiro html_index.html

```
<div>
 <p>
Página de inicio.
<a href="pagina2">Ir para a Página 2.</a>
 </p>
</div>
```

Listagem 4: Ficheiro html_index.html. Conteúdo da página principal da aplicação.

Ficheiro html_pagina2.html

```
<div>
 <p>
Página 2.
<a href="index">Voltar ao início.</a>
 </p>
 <p id="serverContent"></p>
</div>

<script type="text/javascript">
/**
Exemplo de pedido de recursos ao servidor através
de AJAX.
O resultado é carregado para o parágrafo de ID
"serverContent".
**/
$("#serverContent").load("api.php?getExampl
eText=1");
</script>
```

Listagem 5: Ficheiro html_pagina2.html. Conteúdo da segunda página da aplicação.

Ficheiro styles.css

```
body
{
    background-color: #262626;
}

div
{
    padding: 10pt;
    margin: 0 auto;
    width: 800pt;
    text-align: center;
    background-color: white;
}


p
{
    border: solid 2px orange;
    padding: 5pt;
}
```

Listagem 6: Ficheiro styles.css. Folha de estilos da aplicação

Ficheiro api.php

```
<?php
/**
Pedido de template (componente visual da aplicação)
**/
if($_GET["getTemplate"] == "1")
{
    //Construir um array com os elementos
visuais
    $result = array
    (
        "version" => "0.1",
        //Páginas HTML
        "html_index" =>
              file_get_contents("html_index.html"),
        "html_pagina2" =>
            file_get_contents("html_pagina2.html"),
        //Folhas de estilo
        "css_styles" =>
                    file_get_contents("styles.css")
    );

    //Traduzir o array para JSON e enviar
   //como resposta
    echo json_encode($result);
    //Parar
    die();
}

/**
Exemplo de pedido de um determinado recurso
**/
if($_GET["getExampleText"] == "1")
{
    //Responder com texto demonstrativo
    echo "Este texto é o resultado de um
pedido AJAX ao servidor.";
    //Parar
    die();
}
?>
```

Listagem 7: Ficheiro api.php. Webservice fornecedor do resultado da lógica da aplicação.

Ficheiro script.js

```
//Quando o documento estiver pronto...
$(document).ready(function()
{
    captureLinkClick();
    captureOnPopState();

    //Se o template já existe no browser...
    if(templateExists())
    {
        //... mostrar aplicação
        showApplication();
    }
    else
    {
        //Senão, fazer download do template
        getTemplate(function()
        {
            //Quando concluído, mostrar
aplicação
            showApplication();
        });
    }
});

/**
```

TEMA DA CAPA

WEB PERSISTENTE: LOCAL STORAGE

```
Capturar click em links
Esta acção irá ter um comportamento diferente do
normal @method captureLinkClick
**/
function captureLinkClick()
{
    //Ao clicar num link...
    $(document).on("click", "a",
    function(e)
    {
        //...impedir o comportamento por defeito
        e.preventDefault();
        //Mudar para a página definida no attributo
                                        "href" do link
        changePage($(e.currentTarget).attr("href"),
                                                false);
    });
}

/**
Capturar eventos retroceder e avançar do histórico
do browser@method captureOnPopState
**/
function captureOnPopState()
{
    //Ao retroceder/avançar...
    window.onpopstate = function(e)
    {
        var pageName = "";

        //Obter objecto state que contém o nome da
                                        página destino
        var state = e.state;
        //Caso state seja null...
        if(state == null)
        {
            //...é o primeiro acesso à página.
            //Mostrar página principal (index)
            pageName = "index";
        }
        else
        {
            //Senão, mostrar página indicada no
                                        objecto state
            pageName = state.page;
        }

        //Mudar para a página destino
        changePage(pageName, true);
    }
}

/**
Verificar se o template existe no browser
@method templateExists
**/
function templateExists()
{
    //Neste caso verificamos a existência do
      parâmetro "version" como teste da presença do
                                                template
    return
localStorage.getItem("version") !== null;
    //É possível aproveitar este parâmetro para
//verificar se existe uma nova versão do template
//no servidor
}

/**
Obter template e guardar em Local Storage
@method getTemplate
@param {function} callback Função chamada após ter-
minado o download do template
**/
function getTemplate(callback)
{
    //Fazer pedido ao servidor pelo
template
    $.ajax(
    {
        url: "api.php?getTemplate=1",
        type: "get",
        success: function(data)
        {
            //Resposta JSON
            //Fazer parse do resultado do servidor
            var jsonData = $.parseJSON(data);
            //Iterar os elementos enviados pelo
                                            //servidor
            $.each(jsonData, function(key, value)
            {
                //Guardar em Local Storage
                localStorage.setItem(key, value);
            });

            //Chamar função callback
            callback();
        }
    });
}

/**
Carregar pela primeira vez os elementos visuais
para o ecrã
@method showApplication
**/
function showApplication()
{
    //Carregar folha de estilos do template
    $("style").append(localStorage.getItem
("css_styles"));
    //Mostrar página inicial (index)
    changePage("index", true);
}

/**
Carregar uma determinada página guardada em localS-
torage
@method changePage
@param {String} pageName Nome da página a carregar
@param {boolean} replaceState Indica se deve ser
criado ou reutilizado um estado do histórico
**/
function changePage(pageName, replaceState)
{
    //Carregar para o corpo do documento o conteúdo
presente em Local Storage correspondente à pagina
indicada
    $("body").html(localStorage.getItem
                                    ("html_+pageName));
    //Construir objecto de estado stateObject, que
indica o nome da página destino
    var stateObject = {page: pageName};

    //Se foi indicado que o estado deve ser
                                        //substituido...
    if(replaceState)
    {
        //...substituir o estado do histórico
       //utilizando o objecto de estado stateObject
        history.replaceState(stateObject, null,
pageName);
    }
    else
    {
        //Em caso contário, criar um novo estado do
histórico utilizando o objecto de estado
stateObject
```

```
        history.pushState(stateObject, null, page-
Name);
    }
}
```

Listagem 8: Ficheiro script.js. Lógica de apresentação da aplicação.

Caso prático: Plataforma AnyKB

AnyKB é uma plataforma web colaborativa para a partilha de conhecimento no formato de artigos criados pela comunidade, vocacionada para a área tecnológica. A plataforma surge da necessidade de implementar o conceito descrito neste artigo numa aplicação de utilidade real com o objectivo de gerar resultados num ambiente de utilização o mais intensivo possível.

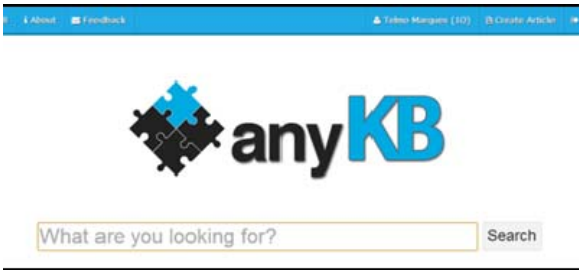


Figura 3: Página inicial da plataforma AnyKB.

Na plataforma AnyKB, qualquer utilizador pode partilhar com a comunidade as suas descobertas, experiências e soluções, ou ainda melhorar o conteúdo já existente. A participação é recompensada na forma de pontos que reflectem a reputação do utilizador dentro da plataforma e a qualidade dos conteúdos que cria. O utilizador enriquece a comunidade com o seu conhecimento e é recompensado com a aprovação desta, num formato quantitativo que destaca o utilizador entre os profissionais da sua área.

Deixo assim o convite para visitar e experimentar o desempenho da plataforma em http://www.anykb.net e contribuir para a comunidade com os seus artigos!

Resultados Estatísticos

Por forma a estudar a escalabilidade da aplicação foram comparadas duas implementações equivalentes do mesmo projecto. A versão "*Proof of Concept*" (PoC) implementa o conceito detalhado neste documento, e a versão "Clássica" funciona como a maioria dos *websites* de hoje: cada página corresponde a um pedido novo (GET) ao servidor.

O desempenho das duas versões foi comparado ao criar um caso de teste que consiste na navegação pelas várias páginas da aplicação, de forma a tirar partido dos benefícios que ambas as abordagens têm ao seu dispor. A métrica utilizada corresponde ao tempo decorrido desde a acção do utilizador (clique num *link*, por exemplo) até ao carregamento completo da página pelo *browser*. Adicionalmente, o caso de teste foi executado em vários cenários de carga do servidor, por forma a analisar como este factor afecta o desempenho de ambas as versões. Os valores de CPU *Load* apresentados correspondem ao formato conhecido como Unix CPU Load [2], tendo sido o servidor levado ao limiar da capacidade de resposta.

O servidor utilizado é uma máquina virtualizada recorrendo ao software para virtualização Xen [4], assente em infra-estrutura escalável de última geração, recorrentemente referenciada como *Cloud*. O servidor apresenta as seguintes características:

CPU: Quad-core Intel® Xeon® CPU E5649 @ 2.53GHz ou equivalente

RAM: 1.5GB DDR3 (1033Mhz)

Disco: 25GB (10K)

Software utilizado:

CentOS Linux 2.6.32 (64-bit)

Apache 2.2.15 (64-bit)

PHP 5.3.3 (64-bit)

MySQL 14.14 (64-bit)

Google Chrome 28 (64-bit)

Para simular, de forma controlada, os vários cenários de carga no servidor foi utilizado um script que executa, sem objectivo, operações pesadas de I/O e CPU.

O gráfico abaixo mostra, em média e mediana, o *speedup* conseguido pela versão PoC relativamente à versão Clássica, ao longo dos vários cenário de carga. É possível verificar que, em condições normais, a versão PoC consegue um desempenho cerca de 5 vezes superior. O desempenho baixa para cerca de 2 a 3 vezes superior à medida que a carga do servidor aumenta. No entanto, a mediana mostra-nos a tendência que a versão PoC tem para um desempenho mais elevado, conseguindo resultados até cerca de 24 vezes superiores.

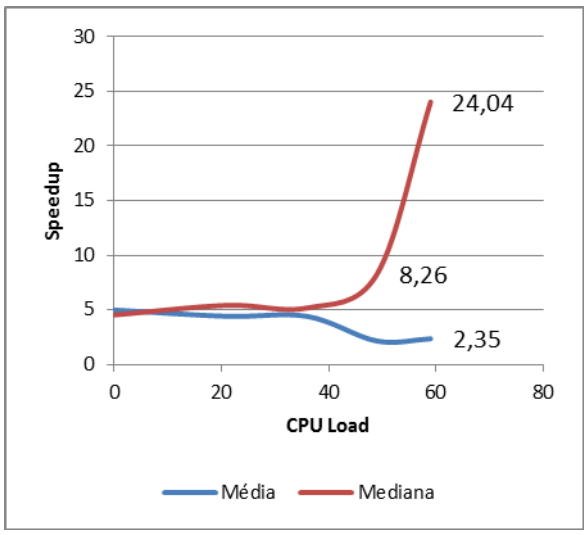


Gráfico 1: *Speedup* médio e mediano da versão PoC, comparativamente à versão Clássica.

WEB PERSISTENTE: LOCAL STORAGE

O seguinte gráfico mostra a evolução do tempo médio e mediano do carregamento das páginas da aplicação, ao longo dos vários cenários de carga. Como esperado, é possível verificar que o valor médio e mediano da versão clássica cresce à medida que a carga do servidor aumenta. O valor médio da versão PoC cresce igualmente com a carga do servidor mas mostra uma curva mais suave, enquanto que a mediana da versão PoC se mantém estável em cerca de 78ms. Isto acontece porque existe uma compensação do tempo acrescido que a informação demora a chegar do servidor pela instantaneidade de reposta do interface, e pela redução substancial de dados enviados pelo servidor.

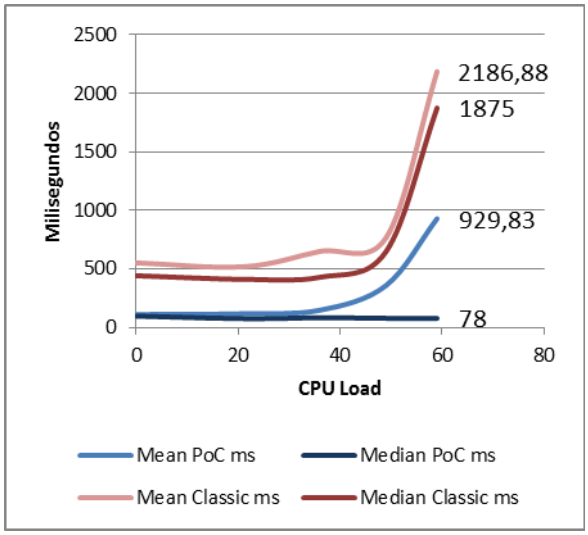


Gráfico 2: Tempo médio e mediano (em milissegundos) do carregamento de várias páginas de ambas as versões.

Conclusão

Concluímos assim que a separação da lógica da apresentação da restante lógica da aplicação, fazendo recurso ao Local Storage, possibilita ganhos de desempenho cerca de 5 vezes superior, comparativamente à mesma aplicação servida de forma clássica. Em casos extremos de carga o valor médio do desempenho baixa para 2 a 3 vezes superior, enquanto que o valor mediano mostra uma capacidade 24 vezes superior de resposta, colocando em evidencia a tendência para a fluidez, e não o contrário.

A implementação deste conceito possibilita assim a criação de aplicações mais rápidas e fluidas, fornecendo uma experiencia mais agradável ao utilizador.

> ***"O Local Storage vem oferecer uma solução mais completa ao permitir que o programador persista no browser todos os elementos de apresentação da aplicação, eliminando completamente a transferência desta componente a partir do servidor."***

Referências

[1] World Wide Web Consortium (W3C), "W3C Proposed Recommendation 9 April 2013," [Online]. Available: http://www.w3.org/TR/webstorage/. [Acedido em Julho 2013].

[2] A. Deveria, "Can I use... Support tables for HTML5, CSS3, etc," [Online]. Available: http://caniuse.com/#feat=namevalue-storage. [Acedido em Junho 2013].

[3] Linux Kernel Organization, Inc., "The Linux Kernel Archives," [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-load.txt. [Acedido em Junho 2013].

[4] Xen Project, "The Xen Project, the powerful open source industry standard for virtualization," [Online]. Available: http://www.xenproject.org/. [Acedido em Julho 2013].

AUTOR

Escrito por Telmo Marques

Estudante licenciado em eng.ª informática com um afecto especial por tecnologias *web*, encontra-se atualmente a concluir o Mestrado em Computação Móvel (MEI-CM) do Instituto Politécnico de Leiria (ESTG-IPLeiria), sob orientação do professor Patrício Domingues, no âmbito do qual se encontra inserido este artigo. Profissionalmente, após uma curta passagem pelo Instituto de Telecomunicações como investigador, é atualmente o responsável pelo desenvolvimento de *software* numa empresa portuguesa de serviços *web.*

A PROGRAMAR

Debug de aplicações em C

Arduino: Accionamento e Controlo de Servos Via Teclado

Introdução ao Web2py

Listas Simplesmente Ligadas e Exemplo de Implementação em C

Pascal – tipo de dados Variant e tipos genéricos

Debug de Aplicações em C

Neste artigo pretende-se dar ao iniciado na programação (e não só) uma ideia de como depurar os seus programas, seja com recurso à própria linguagem, introduzindo no programa código que permita despistar erros, seja com recurso a programas externos, neste caso o gdb.

É minha opinião que todos os iniciados na programação devem experimentar mover a sua aprendizagem para um sistema *nix (no meu caso, utilizo Linux), e assim sendo, este artigo assume que o leitor tem acesso a um sistema semelhante, embora isso não seja de todo um pré-requisito (o gdb também está disponível para Windows). Sendo a linguagem C bastante importante no currículo do programador e ubíqua no mundo da computação (especialmente no dos iniciados), este artigo foca-se na depuração de software escrito nessa linguagem e na utilização do gdb, um *debugger* poderoso e muito utilizado, disponível para imensas plataformas.

Todo o código apresentado neste artigo foi compilado com o gcc versão 4.6.3 e as sessões de debug foram levadas a cabo com o gdb versão 7.5.1 num sistema com Linux instalado (kernel 3.8.13, embora isto não seja muito importante). Os (pequenos) programas apresentados são compilados com as seguintes flags:

-Wall -Wextra -std=c99 -pedantic -O0 -ggdb3

Embora explicar as flags -Wall, -Wextra, -std=c99 e -pedantic fique fora do âmbito deste artigo, é de salientar que a sua utilização permite ao gcc avisar-nos de práticas duvidosas no nosso código, além de nos forçar a seguir o standard C99 (em vez de utilizarmos extensões do gcc). Já as flags -O0 (a letra O maiúscula seguida do número zero) e -ggdb3 merecem uma breve explicação:

- -O0: esta flag indica ao gcc que não deve optimizar o código gerado, o que nos facilita o processo de depuração (impede que o gcc elimine variáveis e reorganize o código, entre outras coisas, o que nos poderia dificultar o processo de depuração).

- -ggdb3: esta flag indica ao gcc para incluir no executável gerado informação para o gdb o mais expressiva possível (ie. não só são incluídos os nomes de variáveis e funções, mas também as macros utilizadas, e outras informações). Podemos, naturalmente, utilizar apenas a flag -g, mas -ggdb3 fornece mais informação ao gdb.

Posto isto, a depuração de um programa pode ser feita de várias formas:

1.Depuração mental do código. Esta é a forma mais comum de depuração, e podemos seguramente dizer que todos os programadores a utilizam. Por esse motivo, não iremos abordá-la (não faz sentido divagarmos sobre como "pensar no código" que escrevemos).

2.Depuração com recurso a printf()s Esta forma de depuração é também bastante utilizada, mas tipicamente recorre-se sempre a printf()s simples. Neste artigo vamos explorar como desenvolver uma função semelhante à printf() exclusivamente para debug.

3.Depuração com recurso ao gdb Esta é a forma mais avançada que iremos abordar no artigo. Com o gdb podemos interromper o programa a meio da sua execução, examinar a sua memória (e alterá-la!), entre outras coisas.

Depuração com printf()

Esta é a modalidade de debug mais conhecida e utilizada (com excepção do debug mental, claro); todos nós, nalguma situação, colocámos meia dúzia de printf() no nosso código para percebermos a evolução do seu estado ao longo da execução.

Imaginemos que estamos a fundo numa função que lida com apontadores (coisa assustadora, muitos dirão), e algo está a funcionar mal. Podemos imprimir o endereço do apontador suspeito para sabermos se é ou não válido:

```
printf("p = %p\n", (void*)p);
```

Para quem não conhece, o especificador %p é utilizado para imprimir endereços de memória (e requer um void* como argumento):

```
// p aponta para um endereço válido
p = 0xbff34a29

// p é NULL
p = (nil)
```

Quem diz apontadores diz qualquer outro fragmento de informação ou valor de uma variável que nos possa ajudar a compreender o estado do nosso programa.

Essencialmente, a utilização de printf() é isto que acabámos de ver, mas seria interessante se pudéssemos obter um pouco mais de informação acerca da origem dos nossos avisos, como por exemplo, em que ficheiro .c (e linha do mesmo) teve origem uma determinada mensagem. Graças às macros __FILE__ e __LINE__ podemos utilizar essa informação no nosso programa. Em C99 podemos até utilizar a macro __func__, que contém o nome da função actual:

```
printf("%s:%d: p = %p\n", __FILE__, __LINE__, d*)
p);
```

o nosso output passa a ser:

```
main.c:24: p = (nil)
```

Bastante melhor! Isto é mais informativo que anteriormente e permite-nos inserir vários printf() no código sem nos preocuparmos em distingui-los uns dos outros, uma vez que cada um estará identificado com o seu ponto de origem.
No entanto, escrever aquela linha enorme sempre que se quer imprimir algo é uma tarefa repetitiva, sujeita a erros. Podemos tentar encapsular o nosso raciocínio numa macro e deixar o pré-processador trabalhar por nós. Uma forma bastante específica do gcc para fazermos isto (um *gccismo*) é declarar a seguinte macro:

```
#define debug(M, ...) printf("%s:%d: " M "\n",
__FILE__, __LINE__, ## __VA_ARGS__)
```

O exemplo acima funciona perfeitamente bem se estivermos a compilar o nosso código sem compatibilidade com o standard C. Em particular, em C89 não existem macros variádicas (que aceitam um número variável de argumentos), e no caso do C99 o problema é pior, pois o operador ## é obrigado pelo standard a não eliminar vírgulas excessivas, como neste exemplo:

```
// o que o programador escreve
debug("Estamos aqui");

// tradução feita pelo pré-processador
printf("%s:%d: Estamos aqui\n", __FILE__,
                                    __LINE__, );
```

Como podemos ver, há uma vírgula adicional antes do parêntesis final. Fora do modo C99, o operador ## elimina essa vírgula e tudo funciona bem. O problema surge quando queremos compilar em modo C99 (com -std=c99) e em modo pedântico (com -pedantic). É-nos apresentado o seguinte aviso:

warning: ISO C99 requires rest arguments to be used

O que quer isto dizer? Bem, quando no modo C99, o operador ## não pode eliminar a vírgula adicional, e o compilador queixa-se que é necessário inserir os argumentos todos (uma vírgula pressupõe um argumento à sua direita). Temos então duas opções: ou ignoramos este aviso ou tentamos alterar o nosso código para ser completamente compatível com C99. Uma vez que ignorar é simples (já está!), vou mostrar como fazer isto de forma compatível com C99. O que precisamos de fazer é, de alguma forma, criar uma função que receba __FILE__ e __LINE__ *antes* da mensagem e seus argumentos opcionais. Desta forma, tanto a mensagem msg como os argumentos opcionais args serão colectivamente referidos como __VA_ARGS__ e nunca haverá uma vírgula em excesso porque virá sempre msg a seguir a ela (e é garantido que msg exista). Vamos então utilizar funções variádicas (que aceitam um número variável de argumentos), suportadas em C através do header <stdarg.h>:

```
void dbg_printf(char *f, int l, char *msg, ...)
{
    va_list args;

    printf("%s:%d: ");  // imprimir o ficheiro e
                                        //linha
    va_start(args, msg);
    vprintf(msg, args); // invocar printf com msg
                                        //e args
    va_end(args);
    putchar('\n');// inserir \n automaticamente
```

Porreiro! Agora já podemos invocar a nossa função da seguinte forma:

```
dbg_printf(__FILE__, __LINE__, "p: %p", (void*)p);
```

O que significa que podemos criar a macro debug da seguinte forma:

```
#define debug(...) dbg_printf(__FILE__, __LINE__,
                                    __VA_ARGS__)
```

E tudo funcionará. __VA_ARGS__ contém sempre *pelo menos* a mensagem a ser escrita no output, por isso a vírgula à direita de __LINE__ tem sempre pelo menos um argumento a seguir. O aviso anterior desaparece.

Mas será que ficou mesmo tudo bem? O que acontece quando quisermos lançar a versão final do nosso programa? Temos que passar todos os ficheiros a pente fino para determinar em quais foi utilizada a macro debug e eliminar manualmente as linhas correspondentes antes de compilarmos a versão final. Que carga de trabalhos. Deve haver uma maneira mais simples! A resposta encontra-se novamente no pré-processador da linguagem C.

Façamos um pequeno desvio. Se calhar o leitor já utilizou nalgum programa a macro assert (disponível em <assert.h>). O que esta macro faz, para quem não a conhece, é receber como argumento uma expressão booleana e terminar o programa naquele ponto caso a expressão seja falsa (== 0, portanto). Um breve exemplo:

```
#include <assert.h>

...

// Assegurar que p não é NULL
assert(p != NULL);
```

Num programa em que p seja de facto NULL, obteremos o seguinte output:

```
$ ./assert-test
assert-test: main.c:42: func: Assertion `p !=
                                ((void *)0)' failed.
Aborted
```

E o programa termina. A expressão ((void *)0) é a forma como a macro NULL está definida no meu sistema (sim, NULL é uma macro). Este tipo de verificações são terríveis para um ambiente de produção, uma vez que não há qualquer tentativa de recuperação do "erro" -- o programa simplesmente termina -- mas são bastante úteis para pequenos programas nos quais queremos que certas condições sejam sempre verdade: a assert encarrega-se de terminar o programa por nós caso não sejam.

E agora o motivo pelo qual fizemos este desvio: o que acontece quando queremos compilar a versão final sem asserts?

DEBUG DE APLICAÇÕES EM C

Segundo man assert, se compilarmos o programa com a macro NDEBUG definida, nenhuma das invocações da macro assert irá gerar código: efectivamente, o programa fica sem asserts. Para isso basta passarmos o símbolo NDEBUG ao gcc no parâmetro -D, assim:

$ **gcc -DNDEBUG -o assert-test assert-test.c**

Temos agora uma ideia para a nossa macro debug: e se ela funcionar como a macro assert? Em particular, e se debug for ligada/desligada pela mesma macro que a assert? Desta forma, o programador pode utilizar tanto debug como assert e ligar ou desligar ambas com um simples parâmetro passado ao compilador.

Vejamos então criar a solução para o nosso problema:

```
#ifndef NDEBUG
# define debug(...) dbg_printf(__FILE__, __LINE__ ,
__VA_ARGS__)
void dbg_printf(char *f, int l, char *msg, ...) {
...
}
#else

# define debug(...)

#endif
```

Sim, é tão simples quanto isto! Quando compilarmos o nosso programa com a flag -DNDEBUG, a macro debug será expandida para espaços em branco, e o próprio código da função dbg_printf() será eliminado pelo pré-processador antes do compilador iniciar a sua função.

O programador cauteloso inserirá algumas instruções debug() no seu programa profilacticamente em pontos chave do mesmo, seguro de que serão removidas com a adição de -DNDEBUG às flags do gcc. Desta forma, os problemas poderão ser encontrados sem necessidade de recompilação após uma execução.

debug() e stderr

Outro dos problemas de utilizarmos a nossa debug() é que o output verdadeiro e o output da debug() aparecerão misturados:

```
$ ./ex02
Idade? 26
main.c:7: idade = 26
Você tem 26 anos de idade
```

Isto nem sempre é desejável. Imaginemos o que acontece num programa com uma execução longa: examinar o output da debug() torna-se algo complicado. O ideal até seria obrigar debug() a escrever o seu output num ficheiro separado. Será que temos de ajustar a nossa dbg_printf() para escrever num ficheiro separado? Sim. E que ficheiro será esse? Bem, esse ficheiro será o stderr.

Por defeito, quando utilizamos a função printf(...), isso é o mesmo que utilizarmos fprintf(stdout, ...). Se utilizarmos manualmente fprintf(stderr, ...) na nossa dbg_printf(), então estaremos a separar o output de debug() do output do programa:

```
void dbg_printf(char *f, int l, char *msg, ...) {
    va_list args;

    fprintf(stderr, "%s:%d: ");
    va_start(args, msg);
    vfprintf(stderr, msg, args);
    va_end(args);
    fputc('\n', stderr);
}
```

O nosso output está agora separado. Mas quando executamos ex03 continuamos a ver tudo junto. Isso acontece porque, por defeito, tanto stdout como stderr estão redireccionados para o mesmo output (habitualmente, esse output é feito no terminal). No entanto, podemos fazer o seguinte:

$ **./ex03 2> debug**
$
$ **./ex03**
Idade? **42**
Você tem 42 anos
$
$ **cat debug**
main.c:7: idade = 42

Não podemos alongar-nos sobre este tipo de redirecção que as shells nos permitem, pelo que o leitor é aconselhado a fazer man bash para saber mais sobre o assunto. Ainda assim, posso adiantar que 2> significa "redireccionar o *file descriptor* 2 (que corresponde a stderr, sendo stdin o 0 e stdout o 1) para o ficheiro debug".

Debug de programas com o gdb

Existem, no entanto, situações em que simplesmente encher o programa de debug() não chega, ou não é prático (ler o output de todos os debug() pode ser entediante e sujeito a erros). Existem situações nas quais vamos querer, em vez de ler o output das debug(), acompanhar a execução do programa em tempo real, examinando a sua memória (e, em certos casos, alterando-a!).

Para estes casos, dispomos do gdb, um depurador bastante conhecido e fácil de utilizar.

Para começar, a compilação de um programa tem que ser ligeiramente alterada de modo a incluir no executável alguma informação que ajuda o gdb: informação sobre os nomes de variáveis, funções, linhas de código correspondentes:

```
$ cat -n ex01.c
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4    for (int i = 0; i < argc; ++i)
5      printf("#%d: '%s'\n", i, argv[i]);
6
7    return 0;
8  }
$ gcc ... -ggdb3 -O0 -o ex01 ex01.c
```

Saliento que ... na linha de comandos acima representa os

parâmetros colectivamente conhecidos como CFLAGS que o programador queira passar ao gcc (ex.: -Wall, -Wextra, ...).

Agora, o executável ex01 será ligeiramente maior que antigamente, e passará a conter informação adicional utilizada pelo gdb.

Vamos então à nossa primeira experiência com o gdb. Começámos por iniciá-lo através da linha de comandos:

*$ **gdb -q***
(gdb)

A flag -q utilizada é uma questão de preferência pessoal, pois omite alguma informação que o gdb emite sempre que é iniciado (versão, licença de utilização, etc). Com a flag -q, tudo isso é omitido e passamos directamente para a linha de comandos do próprio gdb, identificada pelo texto (gdb).

Começamos por dizer ao gdb que vamos querer depurar o nosso ex01. Tendo em conta a função deste nosso programa de exemplo (imprimir o conteúdo de argv), vamos também querer passar-lhe alguns argumentos antes de o iniciarmos com o comando run:

*(gdb) **file ex01***
Reading symbols from /home/pap/e42/ex01...done.
*(gdb) **set args um dois tres***
*(gdb) **run***
Starting program: /home/pap/e42/ex01 um dois tres
#0: '/home/pap/e42/ex01'
#1: 'um'
#2: 'dois'
#3: 'tres'
[Inferior 1 (process 4413) exited normally]

Que podemos ver no output acima? Bem, podemos ver que ex01 foi chamado com os argumentos um dois tres e temos também o seu output, que corresponde ao que esperávamos. No final temos também uma indicação que o processo 4413 (este número será diferente no computador do leitor) terminou normalmente: isso significa que devolveu ao sistema operativo o *exit code* 0 (na linha 7 do código).

Esta experiência não foi, no entanto, muito interactiva; o programa simplesmente correu de uma ponta à outra sem podermos fazer tudo aquilo que mencionei anteriormente: interromper a execução, examinar a memória, etc. Vamos fazer tudo isso *agora mesmo*. Para começar, vamos aprender a pausar a execução do nosso programa através da utilização de *breakpoints* (literalmente, pontos de paragem). Podemos fazê-lo com o comando break X, sendo X o nome de uma função, uma linha do ficheiro .c, um endereço, entre outros. Além do comando break, o gdb possui também o comando tbreak, em tudo semelhante ao break, com a excepção de que o breakpoint apenas funciona uma vez, sendo eliminado mal seja atingido.

Por uma questão de simplicidade, vamos interromper o programa mal entre na função main:

*(gdb) **break main***
Breakpoint 1 at 0x804840d: file ex01.c, line 4.

O gdb rapidamente informa que foi definido o breakpoint onde pedimos. Se quisermos ver uma lista dos breakpoints definidos, podemos fazê-lo via info breakpoints. É-nos apresentada uma lista com os breakpoints (identificados por um número), a sua localização, e se estão ou não activos (campo Enb). Para activar ou desactivar um breakpoint, utilizamos os comandos enable breakpoint ou disable breakpoint, seguidos do número do breakpoint que queremos alterar. Podemos também eliminá-los com `delete breakpoint. Vejamos:

*(gdb) **info breakpoints***
Num Type Disp Enb Address What
1 breakpoint keep y 0x0804840d in main at ex01.c:4
*(gdb) **tbreak main***
Note: breakpoint 1 also set at pc 0x804840d.
Temporary breakpoint 2 at 0x804840d: file ex01.c, line 4.
*(gdb) **info breakpoints***
Num Type Disp Enb Address What
1 breakpoint keep y 0x0804840d in main at ex01.c:4
2 breakpoint del y 0x0804840d in main at ex01.c:4
*(gdb) **delete breakpoint 2***
*(gdb) **info breakpoints***
Num Type Disp Enb Address What
1 breakpoint keep y 0x0804840d in main at ex01.c:4

Como podemos ver, definir pontos de paragem no nosso programa é uma coisa fácil. Mais à frente veremos como criar breakpoints *condicionais* que são accionados, tal como o nome indica, apenas quando se verifica uma determinada condição especificada por nós.

Se agora tentarmos executar o programa, a execução será interrompida mal entremos na main:

*(gdb) **run***
Starting program: /home/pap/e42/ex01 um dois tres

Breakpoint 1, main (argc=4, argv=0xbffff3a4) at ex01.c:4
4 for (int i = 0; i < argc; ++i)

Agora sim começámos a parte engraçada da utilização do gdb. Como podemos ver, é-nos apresentado o breakpoint no qual nos encontramos (é o 1), em que função estamos (main), quais os argumentos passados a essa função (argc=4, argv=0xbffff3a4) e a próxima linha a ser executada (a linha 4, com o ciclo for). Neste ponto, podemos fazer o que quisermos: examinar memória, continuar a execução, o que quisermos. Vamos avançar a execução com a instrução next (ou n). Entraremos no ciclo, e depois executamos novamente next (basta-nos pressionar a tecla ENTER para o gdb repetir o comando anterior) para executar o primeiro printf:

*(gdb) **next***
5 printf("#%d: '%s'\n", i, argv[i]);
*(gdb) **<ENTER>***

#0: '/home/pap/e42/ex01'
4 for (int i = 0; i < argc; ++i)

Existem mais comandos que nos permitem avançar no nosso programa: step e stepi. O comando step funciona igual ao next, mas tem a diferença de que entra nas funções invocadas (ou seja, no nosso exemplo, a utilização de step tentaria entrar na função printf(), se o código da mesma estivesse disponível). O comando stepi funciona a um nível mais baixo: avança o programa de instrução em instrução (estamos a falar de instruções em Assembly).

Como já sabemos que podemos dizer ao gdb onde deverá parar a execução do programa, resta-nos saber como podemos dizer-lhe que queremos que o programa continue a sua execução:

(gdb) ***continue***
Continuing.
#1: 'um'
#2: 'dois'
#3: 'tres'
[Inferir 1 (process 5578) exited normally]

Munidos da capacidade de interromper e retomar a execução dos nossos programas, vamos agora tentar examinar o que se passa durante a sua execução. Recorrendo ao nosso programa de exemplo, vamos tentar inspeccionar argc e argv antes do ciclo principal do programa.

O gdb é um programa complexo e bastante poderoso, o que nos impede de falarmos aqui de todos os comandos que coloca ao nosso dispor. Ainda assim, vamos falar dos mais importantes.

Uma das tarefas mais importantes (e comuns) é a impressão do valor de uma determinada variável (ou endereço de memória, etc). Podemos conseguí-lo através do comando print (ou p):

(gdb) ***tbreak main***
Temporary breakpoint 2 at 0x804840d: file ex01.c, line 4.
(gdb) ***run***
Starting program: /home/pap/ex01

Temporary breakpoint 1, main (argc=4, argv=0xbfffee64) at ex01.c:4
4 for (int i = 0; i < argc; ++i)
(gdb) ***print i***
$1 = -1208229900

Como podemos ver, i tem um valor aleatório, uma vez que ainda não foi inicializada (o que apenas acontece quando entrarmos no ciclo). Tal como outros comandos do gdb, o comando print permite-nos especificar em que formato o valor deverá ser mostrado através da sintaxe print /fmt var, onde fmt é um especificador de formato (o comando help x permite explorar os diferentes formatos):

(gdb) ***print /x i***
$2 = 0xb7fbdff4
(gdb) ***print /t i***
$3 = 10110111111110111101111111110100

Como o leitor terá adivinhado, os especificadores x e t permitem mostrar valores em hexadecimal e binário (t de "two"), respectivamente.

O comando display (disp) é praticamente igual ao print, mas dirige-se a uma necessidade diferente. Utilizamos display quando queremos imprimir um determinado valor sempre que o programa é interrompido (útil para inspeccionarmos variáveis ao longo de um ciclo):

(gdb) ***display argv[i]***
1: argv[i] = <error: Cannot access memory at address 0x9fef6e14>

Uma vez que a variável i ainda não foi correctamente inicializada, o comando display emite o erro acima, pois estamos literalmente a tentar ver o conteúdo de argv[-1208229900]. No entanto, se avançarmos o programa:

(gdb) ***next***
5 printf("#%d: '%s'\n", i, argv[i]);
1: argv[i] = 0xbffff024 "/home/pap/e42/ex01"

Podemos ver que além da próxima linha a ser executada (linha 5), o gdb imprimiu também o valor de argv[i], tal como tínhamos pedido. Uma vez que o gdb conhece os tipos de dados que estamos a utilizar, imprime também o valor da string para onde argv[i] aponta (o que poderia ser feito manualmente com disp /s argv[i], uma vez que tal como print, display pode ser utilizado com especificadores de formato de apresentação).

Imaginemos que não queremos que o nosso programa execute todo o ciclo. Mais especificamente, queremos que apenas mostre dois dos três argumentos passados. Podemos conseguir o que queremos de duas formas:

1. Podemos alterar argc durante a execução para que o seu valor seja 3 e não 4 e deixar o programa seguir o seu rumo.
2. Podemos interromper o programa quando i for modificado para 4 e aí utilizar o comando jump (essencialmente um goto) para fora do ciclo.

Pessoalmente, prefiro a primeira abordagem. Ainda assim, exploremos ambas.

(gdb) ***tbreak main***
Temporary breakpoint 3 at 0x804840d: file ex01.c, line 4.
(gdb) ***run***
Starting program: /home/pap/e42/ex01 um dois tres

Temporary breakpoint 3, main (argc=4, argv=0xbfffee44) at ex01.c:4
4 for (int i = 0; i < argc; ++i)
(gdb) ***set variable argc = 3***

(gdb) ***continue***
Continuing.
#0: '/home/pap/e42/ex01'
#1: 'um'
#2: 'dois'
[Inferior 1 (process 67934) exited normally]

Como podemos ver, alterar o valor de uma variável (ou do conteúdo de qualquer endereço de memória, de facto) é extremamente simples: basta utilizarmos set variable (ou set var) seguido de uma expressão válida em C. Se estivermos a utilizar o gdb com um programa escrito noutra linguagem, adaptamos a expressão à sintaxe dessa mesma linguagem, caso seja suportada pelo gdb. Vejamos como seria num programa compilado com Free Pascal:

...
(gdb) ***set variable i := 3***
…

Vamos agora à nossa segunda hipótese. Queremos interromper o programa antes de imprimir o 3º argumento e saltar para a linha que contém return 0. Para isso, vamos utilizar um breakpoint temporário (tbreak) *condicional*, que só será activado se a condição que lhe fornecermos se verificar:

(gdb) ***tbreak 5 if i == 3***
Temporary breakpoint 4 at 0x8048417: file ex01.c, line 5.

Essencialmente, estamos a dizer para o gdb interromper o programa na linha 5 quando a variável i for 3. Por que motivo não definimos o breakpoint para a linha 4? Nem sempre existe uma correspondência exacta entre as linhas de código e as instruções exactas nas quais o programa pode ser interrompido (devido à visibilidade das variáveis, por exemplo), pelo que é boa ideia utilizar a primeira linha do bloco do ciclo como ponto de paragem, na qual sabemos que o valor de i está bem presente.

(gdb) ***run***
Starting program: /home/pap/e42/ex01 um dois tres
#0: '/home/pap/e42/ex01'
#1: 'um'
#2: 'dois'
Temporary breakpoint 1, main (argc=4, argv=0xbfffee44) at ex01.c:5
5 printf("#%d: '%s'\n", i, argv[i]);
(gdb) ***list***
1 #include <stdio.h>
2
*3 int main(int argc, char *argv[]) {*
4 for (int i = 0; i < argc; ++i)
5 printf("#%d: '%s'\n", i, argv[i]);
6
7 return 0;
8 }

Com o comando list podemos ver que o nosso destino neste ponto é a linha 7, que devolve 0 e termina o programa. Basta-nos dizer ao gdb para *saltar* para esse ponto:

(gdb) ***jump 7***
Continuing at 0x804844a.
[Inferior 1 (process 68571) exited normally]

Como podemos ver, alterámos a execução natural do nosso programa com o gdb. Podemos, em vez de simplesmente 7, especificar ex01.c:7, ou então o endereço no qual queremos continuar (0x804844a, neste caso). E assim terminamos esta nossa breve introdução ao mundo da depuração. O gdb possui alguns comandos que não falei (senão o artigo não teria fim) que nos permitem, entre outras coisas examinar o conteúdo de um endereço de memória (através do comando x, que também aceita especificadores de formato de apresentação, tal como print e display, mas mais avançados que estes), e até nos permite interromper o programa sempre que uma determinada variável (ou endereço) é alterada, através dos comandos watch, awatch e rwatch.

Eventualmente, e quando o leitor estiver mais ambientado naquilo que é a organização de um programa em execução na memória, deverá explorar os comandos relacionados com *stack frames, que são de um valor inestimável quando estamos a explorar a execução de um programa complexo. Estes comandos permitem-nos, entre outras coisas, saber que funções foram chamadas até ao ponto actual de execução.*

Convido portanto o leitor a explorar este fantástico programa. Para começar, basta utilizar um dos comandos menos utilizados, apesar de imensamente útil: o comando help.

Conclusão

Neste artigo falámos de várias coisas, desde a criação de uma "função" printf()-like para as nossas necessidades de registo da evolução do estado do programa até à utilização (ainda que superficial) do gdb. As técnicas aqui ensinadas são úteis para qualquer programador (não apenas os que programam em C, embora esses sejam claramente o público-alvo), e têm como objectivo encorajar práticas de programação emparelhadas com uma boa depuração ao longo de toda a evolução do código.

Espero sinceramente que tenha valido a pena, e que tenha algum impacto no código que escrevam no futuro!

AUTOR

Escrito por António Pedro Cunha (pwseo)
Médico natural de Guimarães, formado na Universidade do Minho.
Programador autodidacta em parte dos tempos livres, inscrito no fórum desde 2006.
Website: http://pwseo.aloj.net

Arduino: Accionamento e Controlo de Servos Via Teclado

Introdução:

Nesta edição irei evoluir um pouco mais as funcionalidades demonstradas no artigo publicado na edição 39 sobre o "Accionamento de Servos Usando Arduino" apresentando agora um artigo sobre o Accionamento e Controlo de Servos usando o teclado do computador como input das acções pretendidas.

Desta forma será possível controlar a acção dos servos usando a interface das teclas do teclado para a definição das acções.

Para a implementação e controlo dos servos usando o teclado não é necessário efectuar qualquer alteração ao esquema eléctrico apresentado no artigo da edição 39, bastando apenas usar a tecnologia "Processing" como interface de ligação entre o utilizador e o micro-controlador Arduino.

Programação:

A linguagem de programação Processing foi a linguagem de programação utilizada para a criação da interface que permitiu o controlo do micro-controlador. Esta linguagem usa uma sintaxe muito semelhante à da linguagem de programação C e permite efectuar com alguma facilidade interfaces gráficas.

No entanto também disponibiliza algumas funcionalidades que permitem recolher o resultado do input de um dispositivo de entrada como o teclado ou o rato e usar essa mesma informação para despoletar uma acção.

Neste caso o Arduino estará conectado via porta de série, que na verdade é emulada pelo cabo USB. Usando a biblioteca "**processing.serial**" iremos recolher o input do teclado e envia-lo para o micro-controlador Arduino em tempo real para que seja despoletada a acção pretendida.

O exemplo demonstrado irá ter apenas 5 acções em que as teclas "UP", "DOWN", "RIGHT", "LEFT" e "S" que despoletam a acção de andar para a frente, para trás, para a direita, para a esquerda e para parar, respectivamente.

Arduino: Código-Fonte Utilizado

Apesar de as acções serem dadas a partir do teclado e recolhidas pela pequena aplicação programada usando a linguagem de programação Processing é necessário que o micro-controlador esteja programado de modo a saber reagir às acções despoletadas pelo utilizador.

Para isso usamos o seguinte código-fonte, que não é mais do que os movimentos pré-definidos em quem a recepção do código ASCII pertencente a determinada tecla recolhida pela função "n = Serial.read();" se traduz numa acção real dos servos.

Este será o código que estará armazenado pelo micro-controlador Arduino, e é em muito semelhante ao código apresentado no artigo publicado na edição 39, tempo apenas definidas as acções que serão realizadas pelos servos mediante o input recebido.

Processing: Interface Utilizador / Arduino

No exemplo em questão é o utilizador que irá despoletar as acções dos servos mediante o toque em determinadas teclas do teclado do computador, sendo neste caso as teclas "UP", "DOWN", "RIGHT", "LEFT" e "S" despoletam a acção de andar para a frente, para trás, para a direita, para a esquerda e para parar, respectivamente.

Para que tal aconteça temos de criar uma "interface" entre o utilizador e o micro-controlador Arduino que irá então despoletar as acções sobre os servos. Para isso iremos utilizar uma pequena aplicação recorrendo a linguagem de programação Processing que irá servir de ponte entre a recolha das acções dadas pelo utilizador e a sua transmissão para o micro-controlador Arduino para que este as possa interpretar e processar.

```
//Definição de Variáveis
int n;
int motorPin1 = 5;
int motorPin2 = 6;
int motorPin3 = 10;
int motorPin4 = 11;
void setup()
{
Serial.begin(9600);
pinMode(motorPin1, OUTPUT);
pinMode(motorPin2, OUTPUT);
pinMode(motorPin3, OUTPUT);
pinMode(motorPin4, OUTPUT);
}
void loop()
{
//Verificar se a Porta de Série está disponível
if (Serial.available() > 0)
{
n = Serial.read();
}
//Definição das Acções andar para a frente, para
trás, para a direita, para a esquerda e para parar
if (n == 119)
{
//Avançar
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH);
digitalWrite(motorPin3, HIGH);
digitalWrite(motorPin4, LOW);
}
if ( n == 115)
{
```

```
//Recuar
digitalWrite(motorPin1, HIGH);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, HIGH);
}
if ( n == 100)
{
//Direita
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, HIGH);
}
if ( n == 97)
{
//Esquerda
digitalWrite(motorPin1, HIGH);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, HIGH);
digitalWrite(motorPin4, LOW);
}
if (n == 113)
{
//Parar
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, LOW);
}
}
```

Este será o código que estará armazenado pelo micro-controlador Arduino, e é em muito semelhante ao código apresentado no artigo publicado na edição 39, tempo apenas definidas as acções que serão realizadas pelos servos mediante o input recebido.

Processing: Interface Utilizador / Arduino

No exemplo em questão é o utilizador que irá despoletar as acções dos servos mediante o toque em determinadas teclas do teclado do computador, sendo neste caso as teclas “UP”, “DOWN”, “RIGHT”, “LEFT” e “S” despoletam a acção de andar para a frente, para trás, para a direita, para a esquerda e para parar, respectivamente.

Para que tal aconteça temos de criar uma “interface” entre o utilizador e o micro-controlador Arduino que irá então despoletar as acções sobre os servos. Para isso iremos utilizar uma pequena aplicação recorrendo a linguagem de programação Processing que irá servir de ponte entre a recolha das acções dadas pelo utilizador e a sua transmissão para o micro-controlador Arduino para que este as possa interpretar e processar.

```
//Definição da biblioteca utilizada
import processing.serial.*;

Serial myPort;

void setup()
{
  size(200,200);

  noStroke();
  background(0);

  // Alteração "COM35para a Porta COM pretendida
  myPort = new Serial(this, "COM3", 9600);
}

void draw()
{
  //Seta avançar
  triangle(100 , 25, 75, 75, 125, 75);
  //Seta Esquerda
   triangle(75 , 125, 75, 75, 25, 100);
  //Seta Recuar
  triangle(75 , 125, 100, 175, 125, 125);
  //Seta Direita
  triangle(175 , 100, 125, 75, 125, 125);
}

void keyPressed()
{
  if (key == CODED)
  {
    if (keyCode == UP)
   {
      myPort.write(119);
      fill(153);
      //Seta avançar
      triangle(100 , 25, 75, 75, 125, 75);
      fill(255);
      //Seta Esquerda
      triangle(75 , 125, 75, 75, 25, 100);
      //Seta Recuar
      triangle(75 , 125, 100, 175, 125, 125);
      //Seta Direita
      triangle(175 , 100, 125, 75, 125, 125);
   }
    else if (keyCode == DOWN)
    {
      myPort.write(115);
      fill(153);
      //Seta Recuar
      triangle(75 , 125, 100, 175, 125, 125);
      fill(255);
      //Seta avançar
      triangle(100 , 25, 75, 75, 125, 75);
      //Seta Esquerda
      triangle(75 , 125, 75, 75, 25, 100);
      //Seta Recuar
      triangle(75 , 125, 100, 175, 125, 125);
      //Seta Direita
      triangle(175 , 100, 125, 75, 125, 125);
    }
    else if (keyCode == LEFT)
   {
      myPort.write(100);
      fill(153);
      //Seta Esquerda
      triangle(75 , 125, 75, 75, 25, 100);
      fill(255);
      //Seta Recuar
      triangle(75 , 125, 100, 175, 125, 125);
      //Seta avançar
      triangle(100 , 25, 75, 75, 125, 75);
      //Seta Esquerda
      triangle(75 , 125, 75, 75, 25, 100);
      //Seta Direita
      triangle(175 , 100, 125, 75, 125, 125);
   }
    else if (keyCode == RIGHT)
   {
      myPort.write(97);
      fill(153);
      //Seta Direita
      triangle(175 , 100, 125, 75, 125, 125);
      fill(255);
       //Seta Esquerda
      triangle(75 , 125, 75, 75, 25, 100);
```

```
      //Seta Recuar
      triangle(75 , 125, 100, 175, 125, 125);
      //Seta avançar
      triangle(100 , 25, 75, 75, 125, 75);
    }
  }

    else if (key == &apos;s&apos; || key ==
                                    &apos;S&apos;)
  {
    myPort.write(113);
  }

}
```

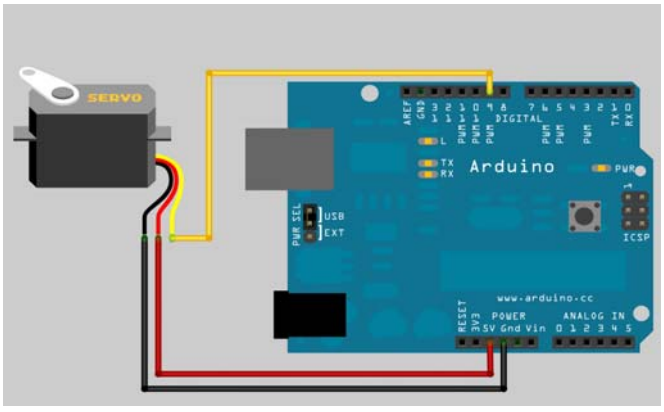
Conclusão:

O artigo desta edição mostra apenas uma das inúmeras formas do utilizador controlar directamente e em tempo real as acções dos servos usando como "processador/interpretador" dos dados o micro-controlador Arduino.

Como se diz na gíria popular a criatividade de cada um é o limite, sendo que é possível aplicar este exemplo aos mais variados projectos recorrendo à agilidade e facilidade de programação do micro-controlador Arduino.

"A linguagem de programação Processing foi a linguagem de programação utilizada para a criação da interface que permitiu o controlo do micro-controlador."

"Apesar de as acções serem dadas a partir do teclado e recolhidas pela pequena aplicação programada usando a linguagem de programação Processing é necessário que o micro-controlador esteja programado de modo a saber reagir às acções despoletadas pelo utilizador."



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/

Introdução ao Web2py

Introdução:

A linguagem Python foi inicialmente introduzida em 1989 como uma linguagem de propósito geral (*general-purpose)* de alto-nível, bastante legível (há quem a considere *pseudo-código executável)*. À linguagem foi sendo adicionado suporte para diversos paradigmas de programação, nomeadamente o paradigma de imperativo, o funcional e mais tarde o paradigma de programação orientada a objectos. Foi ainda complementada com a introdução de ferramentas de terceiros que a tornaram numa linguagem ainda mais robusta, mas isto já sai fora daquilo que vos trago neste artigo.

Voltando ao tema do artigo, o web2py é uma framework web, livre (open-source), destinada ao desenvolvimento de aplicações segundo a metodologia AGILE. Tem como principal objectivo o desenvolvimento de *a*plicações web seguras e "orientadas" pelas bases de dados. Foi escrita em Python e é programável em Python. É uma framework (*full-stack*), ou seja, contém todos os componentes necessários para construirmos aplicações web totalmente funcionais, e foi projectado para orientar o web-developer de forma a que acompanhe as boas práticas de engenharia de software, como a utilização do modelo MVC (Model View Controller), separa a representação de dados (o *model*) da apresentação dos mesmos (a *view*) e estes dois da lógica e aplicação (o *controller*).

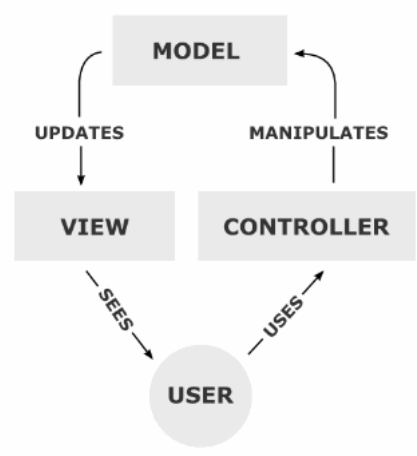
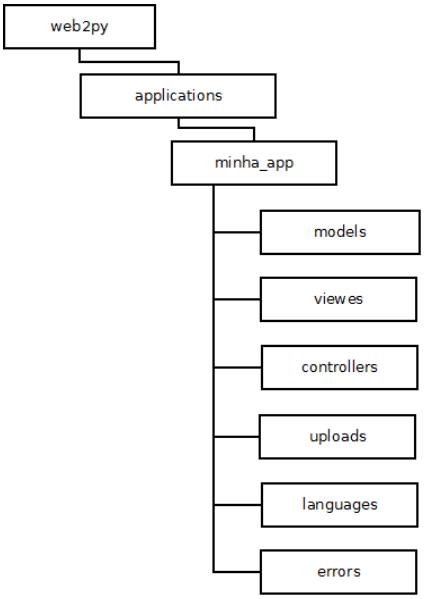


Diagrama de funcionamento de uma aplicação MVC em web2py.

A estrutura de directórios de uma aplicação em web2py é "auto-explicativa"; ainda assim, e para uma melhor compreensão da mesma e de todas as potencialidades da framework, coloco o diagrama resumido da estrutura de directórios onde se podem observar alguns directórios adicionais (além dos habituais models, views e controllers): o directório "languages" (o web2py já inclui suporte multi-lingue), o directório "uploads", para onde vão todos os uploads a menos que redireccionados explicitamente no código, o directório "errors" onde são armazenados todos os tickets de erro, e respectiva stack-trace, para posterior consulta, com vista à correcção do erro.



A framework web2py fornece também bibliotecas para ajudar o programador projectar, implementar e testar cada uma dessas três partes separadamente e fá-las trabalhar juntas. É construída tendo como ponto forte a segurança, isto é, aborda automaticamente muitas das questões que podem levar a vulnerabilidades, seguindo práticas bem estabelecidas. Por exemplo, valida todas as entradas (para evitar injections), executa escape de todo o output (para evitar cross-site scripting, XSS), renomeia os arquivos enviados (directory traversal attacks), aborda e trata as questões de segurança mais comuns, para que os programadores se possam concentrar em produzir a aplicação sem estar constantemente preocupados com a possibilidade de introduzirem de forma não-intencional vulnerabilidades no código da aplicação.

O web2py inclui uma Camada de Abstração de Dados (DAL, Database Abstraction Layer), que nos permite interagir com bases de dados como se fizessem parte da linguagem Python; a DAL "transforma" os nossos objectos Python em queries de acordo com o SGDB que estivermos a utilizar, permitindo-nos escrever o nosso código de uma forma idiomática e independente de sistemas específicos. De facto, a DAL sabe como gerar código SQL para os SGBD's SQLite, MySQL, PostgreSQL, MSSQL (inclui versão 2013), Firebird, Oracle, IBM DB2, Informix, Ingres, e MongoDB, SAPDB, Sybase, entre outros, bastando que estejam instalados os

respectivos drivers. Outra das funcionalidades da DAL é poder invocar as funções de armazenamento de dados do Google, se estiver em execução Google App Engine (GAE).

Uma vez criadas uma ou mais tabelas na base de dados, o web2py gera automaticamente uma interface de administração web-based completamente funcional para aceder à base de dados e suas tabelas, bem como realizar outras tarefas que, em PHP, seriam comummente executadas com recurso ao PHPMyAdmin. A framework web2py distingue-se de outras frameworks web na medida em que é a única framework que abraça plenamente o paradigma da Web 2.0, onde a web é o computador.

Outra vantagem do web2py é o facto de não requerer instalação ou configuração.

Além disso pode ser executado em qualquer arquitectura onde seja executado o interpretador de Python (Windows, Windows CE, Mac OS X, iOS, Unix / Linux, Android) e para o desenvolvimento, implantação e manutenção de fases para as aplicações, pode ser feito através de uma interface web local ou remota.

A framework web2py é executada com CPython (a implementação feita em C) e PyPy (Python escrito em Python), em versões do Python 2.5, 2.6 e 2.7. A web2py fornece um sistema de emissão de tickets para eventos de erro que permitem informar o programador, o administrador da plataforma e o utilizador de que ocorreu um erro na execução da aplicação, com a respectiva "stack-trace". Além disso a web2py é open-source e disponível sob a licença LGPL versão 3.

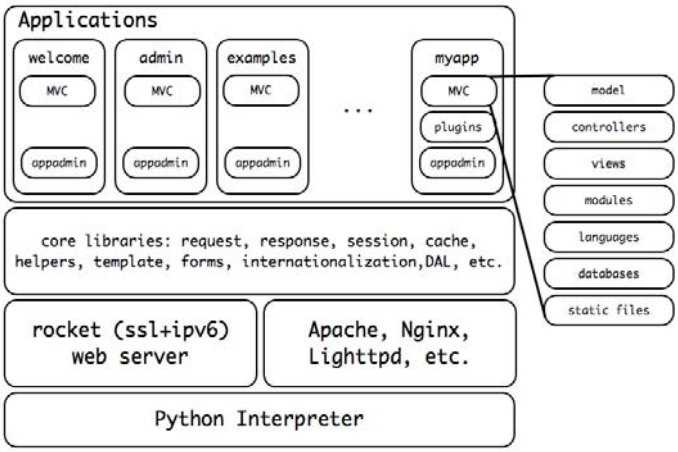


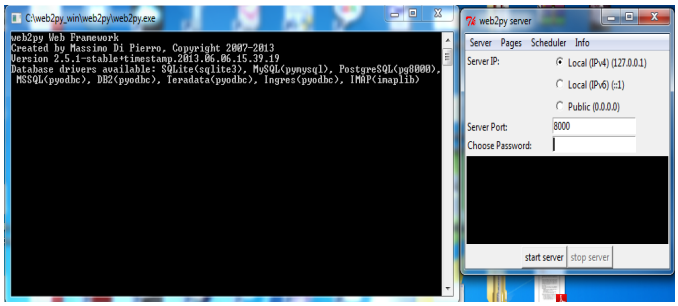
Diagrama de estrutura da framework web2py

Vamos à preparação do ambiente de execução do web2py:

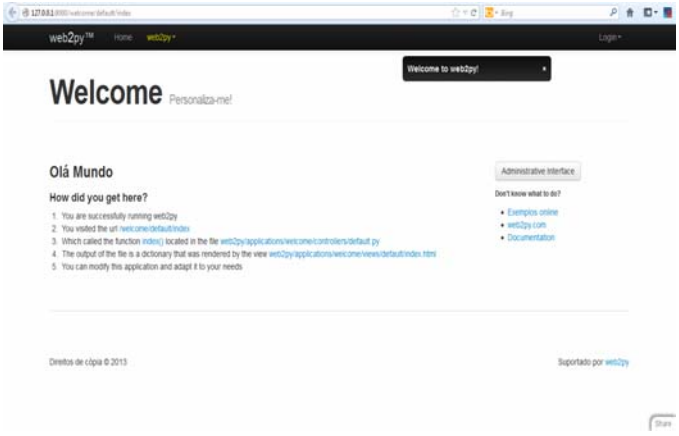
A primeira coisa a fazer para se começar a usar o web2py é descarregar do web site da framework todos os componentes necessários, tanto o Python como a framework web2py propriamente dita. Isto varia consoante o sistema operativo utilizado, pelo que ao longo do artigo irei focar-me apenas na preparação do ambiente nos sistemas operativos Microsoft Windows e GNU/Linux, neste caso a distribuição Ubuntu (baseada na distribuição Debian), pois são os mais comummente utilizados.

Windows:

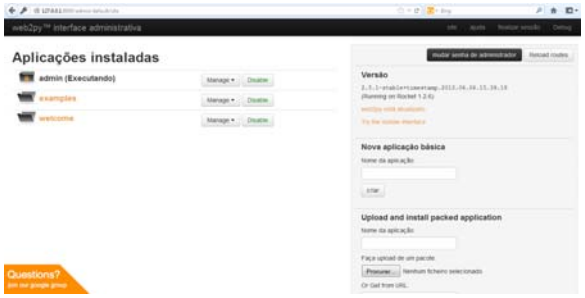
- Fazer o download do ficheiro web2py_win.zip e copiá-lo para a directoria C:\
- Descomprimir o ficheiro descarregado
- Executar o ficheiro web2py.exe que se encontra em C:\web2py_win\web2py\web2py.exe
- Vai surgir uma janela onde pede para se inserir uma password.
- Nesta janela deve ser inserida uma password que depois será utilizada para administração.



- Uma vez inserida a password, basta clicar em "Start Server" e no browser será aberta a página de administração do web2py.



- Neste momento podemos ir à interface administrativa, bastando clicar em Administrative Interface, onde será pedida a password anteriormente definida e estaremos na interface de administrador do web2py.



- Nesta interface podem ser iniciadas e terminadas as aplicações em execução no ambiente web2py, instaladas e carregadas novas aplicações e até desenvolvidas novas aplicações.

GNU/Linux Ubuntu:

- Na shell basta digitar a seguinte sequência de comandos:
  - # wget http://web2py.googlecode.com/hg/scripts/setup-web2py-ubuntu.sh
  - # chmod +x setup-web2py-ubuntu.sh
  - # sudo ./setup-web2py-ubuntu.sh
  - #sudo -u www-data python -c "from gluon.main import save_password; save_password (raw_input('admin password: '),443)"
  - Mover para o directório do web2py e executar ./scripts/web2py.ubuntu.sh

Existem muitas outras configurações que são possíveis de ser feitas e muitos módulos que podem ser acrescentados, mas não irei aprofundar essas questões num artigo que se pretende ser apenas e só uma *introdução* ao web2py. Futuramente, poderei vir a aprofundar esses aspetos noutros artigos, mas de momento, vamos concentrar-nos no essencial para começar a utilizar o web2py.

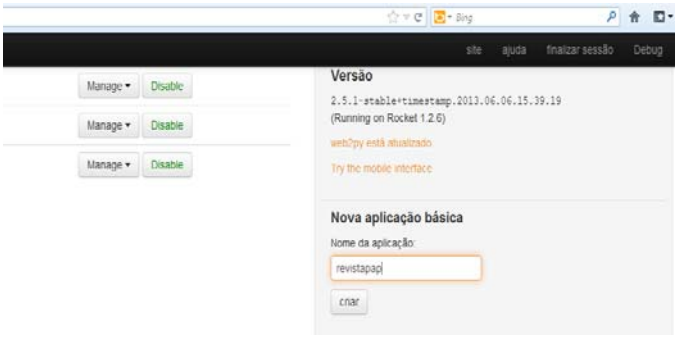
Criar a primeira aplicação:

Não vou entrar em detalhes sobre IDEs de desenvolvimento, porque muito honestamente não sou fã de nenhuma em particular. Em ambiente Windows uso o Notepad++ em ambientes *nix uso o GNU nano, por isso não vou aprofundar um tema sobre o qual existem diversas opiniões e todos os programadores têm um IDE de "preferido". No entanto, deixo a ressalva de que já uma grande parte dos IDE's existentes dispõem de plugins para Python. Exemplos disso são o Eclipse, o Visual Studio, entre outros.

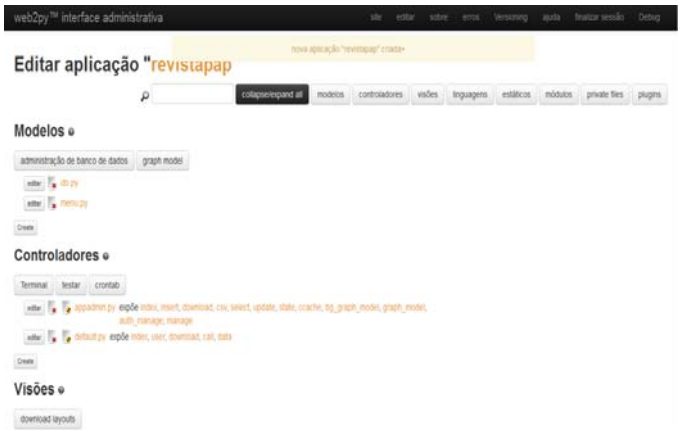
Agora vamos à nossa primeira aplicação, que se pretende ser uma aplicação simples, baseada no modelo MVC, com uma pequena base de dados e que apenas pretende demonstrar as potencialidades do web2py.

Para começar o desenvolvimento desta nossa primeira aplicação, iniciamos o web2py normalmente como descrito anteriormente e entramos na área administrativa.

Do lado esquerdo, como na imagem abaixo, vamos ter uma caixa de texto para escrever o nome da nossa nova aplicação. Neste caso e para o devido efeito, vamos chamar-lhe "*revistapap*", digitando o nome na caixa de texto e de seguida clicando no botão criar.



Feito isto, deve aparecer no browser algo semelhante ao que se encontra na imagem abaixo.



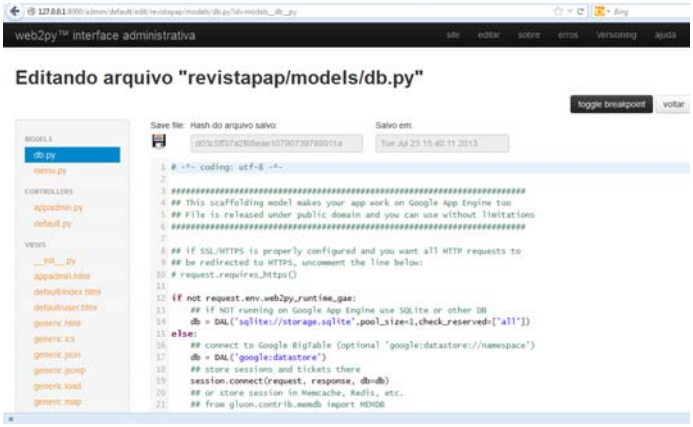
É nesta altura que começamos a construção da base de dados. Para isso clicamos em "Editar" em frente do ficheiro db.py e aparecerá automaticamente no browser um editor pré-definido que corre no próprio browser. Podemos editar o código directamente aqui, editar localmente e colar aqui ou simplesmente fazer o download do ficheiro, editar localmente e fazer o upload posteriormente; fica ao critério de cada um fazer como entender melhor. Neste caso vou editar no editor web, mas pessoalmente prefiro editar num editor de texto normal e fazer o upload para o respetivo directório de seguida.

É neste ficheiro que criamos a tabela "revistas" que vai suportar esta nossa aplicação. A tabela revistas contém os campos edicao, descricao, editor, tema_capa e pub_data. Cada campo tem logo definido o seu comprimento, se é ou não requerido, se tem valores pré-definidos e o seu tipo de dados, conforme podemos observar no código abaixo.

db.py

```
db.define_table('revistas',
Field('edicao', length=200, required=True,
                        requires=IS_NOT_EMPTY()),
Field('descricao', length=2000, required=True,
                        requires=IS_NOT_EMPTY()),
Field('editor', db.auth_user, required=True,
                        requires=IS_NOT_EMPTY()),
Field('tema_capa', length=20,requires=IS_IN_SET
(('Python', 'Java', 'GNU/Linux', 'Frameworks em

PHP'))),
```

```
Field('pub_data', 'datetime', default=request.now,
                                        writable=False)
)
```



Aprofundando um pouco mais sobre a DAL, vemos que se encontra organizada em diversas classes, cada uma com os seus métodos e funcionalidades, como poderemos ver nos exemplos seguintes:

Para fazer uma ligação a uma base de dados executamos o seguinte objecto da DAL:

db = DAL('sqlite://revistapap.db')

A classe *Table* representa uma tabela da base de dados. Em vez de intanciamos a tabela directamente (criando um objecto da classe Table manualmente) usamos o metodo *define_table* da DAL para fazer isso por nós. Por exemplo:

db.define_table('revistas', field('edicao'))

Os metodos da classe *Table* da DAL mais utilizados são *insert*, *truncate*, *drop*, e *import_from_csv_file* sendo este ultimo destinado a importar dados vindos de ficheiros *.csv* (que contêm dados separados por virgulas).

O metodo *field* é utilizado para instanciar campos de uma tabela, e pode ser passado como um argumento para o método *define_table* da DAL.

Quando executamos um pedido à base de dados recebemos aquilo que chamariamos de "resultset", neste caso em forma de objectos do tipo Row, como podemos ver no seguinte exemplo:

rows = db(db.revista.edicoes!=None).select()

Como *rows* contém vários objectos do tipo Row, podemos utilizar por um ciclo for para os apresentar. Neste caso ficaria algo como:

```
for row in rows:
print row.edicoes
```

Vamos usar ainda com bastante frequencia o objecto *Query, que de forma resumida é uma* query sql com uma clausula *where,* por exemplo:

```
myquery = (db.revistas.edicoes != None) |
(db.revistas.edicoes > '30')
```

Set é outro objecto que se utiliza de forma comum e que representa um conjunto de registos. Tem como métodos mais importantes os seguntes: count, select, update, e delete, como podemos ver no exemplo abaixo:

myset = db(myquery)

rows = myset.select()

myset.update(myfield='valor')

myset.delete()

Neste momento o leitor já deve estar farto de ler sobre a DAL. Faltam apenas mais dois exemplos que pretendo apresentar neste artigo, por considerar importantes e pernitentes; são eles as expressões de ordenação e agrupamento, e os métodos para se ligar a uma base de dados.

As expressões como *orderby e groupby,* podem ser escritas em python tal como tudo o que foi feito até aqui e encontram-se na classe Expression, como pode ser observado no exemplo que se segue:

```
ordenacao = db.revistas.edicoes.upper() |
                                    db.revistas.edicoes
db().select(db.revistas.ALL, orderby=edicoes)
```

E por fim vamos às ligações à base de dados, também conhecidas por "connection strings":

Uma ligação à base de dados é estabelecida instanciando um objecto DAL, como podemos ver no exemplo abaixo onde se apresenta uma ligação a uma base de dados MySQL:

```
db = DAL('mysql://username:password@localhost/
                                revista', pool_size=0)
```

Note-se que *db* não é nenhuma instrução ou palavra reservada, é uma variável local que é declarada e armazena um objecto do tipo DAL.

Apenas por curiosidade deixo também o exemplo de conecção a uma base de dados Google/NoSQL:

db = DAL('google:datastore')

Sei que existe muito mais a falar sobre a DAL, pelo que aconselho a leitura da documentação oficial da framework web2py para aprofundar esta tão importante e diferenciadora funcionalidade.

Voltando agora à nossa aplicação, uma vez criado o *Model*, precisamos de criar o respectivo *Controller*. Para tal, procedemos de forma idêntica à anterior, mas desta vez vamos editar o ficheiro default.py, que irá conter as funções para lidar com os pedidos do utilizador.

Primeiramente escrevemos o código para lidar com as operações CRUD (Creat, Read, Update, Delete)

```
def index():
""" Executa a query à base de dados para selecionar
""" todas as revistas, ordena-as por data de
""" publicação, e retorna um 'dicionário' dict ao
```

```
""" template, com todas as edições """
response.flash = "Bem vindo à View index!"
notes = db(db.revistas).select
(orderby=db.revista.pub_date)
return dict(revista=revista)
```

De seguida escrevemos o código das restantes funções:

Função create:

```
def create():
""" Gera um form correspondente ao model e executa
""" o seu 'render', se o formulário enviar alguns
""" dados, a função valida os dados e grava-os na
""" base de dados."""
response.flash = "Esta é a página onde são
                          inseridas novas edições"
form=SQLFORM(db.revistas)
if form.process().accepted:
response.flash = 'query sql aceite'
elif form.errors:
response.flash = 'query sql contem erros'
else:
response.flash = 'Por favor preencha o formulário'
return dict(form=form)
```

Função edit:

```
def edit():
""" Esta função pré-preenche os dados da instância
""" da edição que foi pedida para ser editada e faz
""" o seu render. Assim que o utilizador envie
da""" dos, ela grava-os na base de dados."""
note = db.revista(request.args(0)) or redirect(URL
                                          ('erro'))
form=SQLFORM(db.revistas, edicao, deletable = True)
if form.validate():
if form.deleted:
db(db.notes.id==note.id).delete()
redirect(URL('index'))
else:
note.update_record(**dict(form.vars))
response.flash = 'registo alterado'
else:
response.flash = 'Ocorreu algum erro!'
return dict(form=form)
```

Ambas as funções acima devem ser colocadas no ficheiro default.py, conforme a listagem abaixo, do ficheiro completo:

default.py:

```
def index():
""" Executa a query à base de dados para selecionar
""" todas as revistas, ordena-as por data de
""" publicação, e retorna um 'dicionário' dict ao
""" template, com todas as edições """
response.flash = "Bem vindo à View index!"
notes = db(db.revistas).select
(orderby=db.revista.pub_date)
return dict(revista=revista)

def create():
""" Gera um form correspondente ao model e executa
""" o seu 'render', se o formulário enviar alguns
""" dados, a função valida os dados e grava-os na
""" base de dados."""
response.flash = "Esta é a pagina onde são
""" inseridas novas edições"
form=SQLFORM(db.revistas)
if form.process().accepted:
response.flash = 'query sql aceite'
elif form.errors:
response.flash = 'query sql contem erros'
else:
response.flash = 'Por favor preencha o formulário'
return dict(form=form)

def edit():
""" Esta função pré-preenche os dados da instância
""" da edição que foi pedida para ser editada e faz
""" o seu render, assim que o utilizador envie
da""" dos, ela grava-os na base de dados."""
note = db.revista(request.args(0)) or redirect(URL
('erro'))
form=SQLFORM(db.revistas, edicao, deletable = True)
if form.validate():
if form.deleted:
db(db.notes.id==note.id).delete()
redirect(URL('index'))
else:
note.update_record(**dict(form.vars))
response.flash = 'registo alterado'
else:
response.flash = 'Ocorreu algum erro!'
return dict(form=form)
```

Agora só nos falta criar uma *View* para podermos interagir com a nossa aplicação.

Para tal, regressamos ao painel "Editar aplicação", clicamos no botão "Create" ou "Criar", escrevemos /default/create.html (neste caso vamos criar a view para criar edições) e uma outra view para mostrar todas as edições (index.html). Como poderão ver as views são pouco mais do que código HTML, mas pode ser também utilizado código javascript, podem ser utilizadas bibliotecas de jQuery, etc... Neste caso e para simplificar ainda mais, vamos utilizar templates que já vêm incluídos no web2py.

create.html:

```
{{extend 'layout.html'}}
<h1>This is the default/create.html template</h1>
{{=form}}
```

index.html:

```
<h1>Olá Mundo e Leitores, Bem vindos à nossa pri-
meira aplicação!!</h1>
<p>Nesta aplicação podem realizar as seguintes
                                        operações:</p>
<h1><a href= "{{=URL(r=request,
f='create')}}">Criar edições da nossa revista
              favorita "Revista PROGRAMAR"</a></h1>

{{ for edicao in revistas: }}
<h3>{{=edicao.titulo}}</h3>
<h4>{{=edicao.descricao}}</h4>
<ul>
<li><a href= "{{=URL(r=request, f='edit',
args=edicao.id)}}">Editar edições</a></li>
</ul>
{{pass}}
```

Posto isto, podemos aceder à nossa aplicação através do browser, bastando para isso digitar o endereço http://127.0.0.1:8000/revistapap/default/index no campo do endereço do browser.

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

> ***Tem como principal objectivo o desenvolvimento de aplicações web seguras e "orientadas" pelas bases de dados.***

Bibliografia:

Beginning Python (Using Python 2.6 and 3.1); James Payne

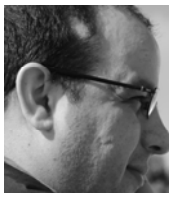
web2py (3rd Edition); Massimo Di Pierro

Conclusão:

Como podemos ver, o web2py fez a maior parte do trabalho duro por nós. Não houve qualquer preocupação de escrita de código para validar inputs, nem sequer preocupação em saber qual o SGBD que está a correr, muito menos estar preocupado em criar a tabela no SGBD. Tudo é feito em código Python de forma simples, deixando apenas as views escritas em HTML.

Novamente neste exemplo não houve preocupação alguma em usar HTML5 nem em indicar a codificação (charset), nem nada de semelhante, pois trata-se de um artigo introdutório à framework web2py e não de um artigo que pretenda explicar de forma exaustiva a framework web2py, pois se tal fosse a pretensão, o artigo tornar-se-ia demasiado extenso e aborrecido para os leitores. Com esta introdução pretendi apenas apresentar a framework, mostrar um pouco das suas potencialidades e deixar ao critério de cada leitor prosseguir a sua aprendizagem pesquisando e lendo mais sobre esta fantástica framework.

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.

.

Listas Simplesmente Ligadas e Exemplo de Implementação em C

O que será isto das listas? Uma lista é uma estrutura de dados sequencial que resolve o típico problema dos arrays. E qual é o problema do arrays (vectores) poderá pensar o leitor? Se pensarmos mais concretamente na linguagem C, o que precisamos para definir um array? O tipo de dados e o número de elementos que iremos ter! E no início da implementação de um programa, definir o número de elementos que o array poderá ter pode ser problemático. Ou porque não sabemos em definitivo quantos elementos vai ter o array ou porque depois poderemos querer adicionar ou remover elementos.

Mantendo sempre o nosso pensamento na linguagem C, temos uma forma de aumentar ou diminuir o número de elementos de um array usando a função realloc disponível na biblioteca stdlib.h, mas o que esta função faz na prática é realocar todos os elementos novamente, ou seja, se pensarmos no caso em que só queremos adicionar um elemento, e se o nosso array tiver vários elementos, realocar toda a memória novamente é um desperdício de recursos computacionais, recursos estes que poderíamos estar a utilizar noutra operação do nosso programa.

Para além do problema da realocação de memória ainda existe o facto de, na linguagem C, os elementos dos arrays serem alocados de forma sequencial, ou seja, se o array for grande só o conseguimos alocar havendo quantidade suficiente de memória contígua livre. Contudo, no caso das listas os elementos são alocados dinamicamente, não sendo preciso existir grandes quantidades de memória contígua.

Na figura seguinte o leitor pode verificar como é que cinco elementos de uma lista podem ser alocados em memória.

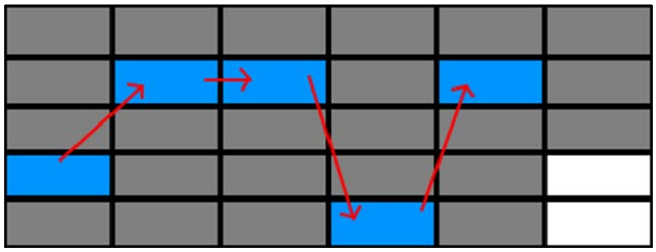


Figura 1 – Exemplo visual da alocação em memória de uma lista com 5 elementos

Na figura 1, os rectângulos cinzentos são espaços de memória não disponível, os rectângulos brancos correspondem às zonas de memória que estão disponíveis e a azul estão representados os elementos que fazem parte da lista. Para os leitores que têm menos experiência em implementação de listas, e que ainda não sabem como uma lista funciona, é natural que se perguntem o que fazem as setas que estão visíveis na figura. Como já foi referido em cima, as listas são estruturas de dados sequenciais, ou seja, as setas representam apontadores de um elemento da lista para o seguinte, formando assim uma sequência. De uma forma simples, cada elemento da lista tem a indicação de qual é o elemento que se segue na lista, assim, quando queremos aceder ao próximo elemento de uma lista, basta vermos para que parte da memória está a apontar o apontador.

Agora que já temos uma ideia geral do que é uma lista simplesmente ligada vamos falar sobre alguns aspetos mais técnicos antes de passarmos à prática.

Por norma, a cada elemento de uma determinada lista é chamado de nó. E nada melhor para termos uma ideia do que é um nó do que utilizarmos um exemplo prático. Um nó vai ser uma estrutura de dados que obrigatoriamente tem de ter o "tal" apontador, apontador esse que irá fazer a ligação ao próximo elemento da lista, para além do apontador podemos ter outros dados, tantos quanto acharmos necessários.

Gostaria de dar destaque a dois casos especiais de nós das listas, neste caso, o primeiro e o ultimo nó, designados por cabeça e cauda da lista respetivamente.

Sobre a "cabeça da lista" há uma regra dourada que devemos ter sempre em mente quando usamos esta estrutura: Qualquer função que altere a estrutura da lista deve sempre retornar um apontador para a sua cabeça. Chamo a atenção do leitor pois esta regra é importante para nunca perdermos a cabeça da lista pois em certos casos poderíamos não conseguir aceder a alguns elementos pois perdendo a cabeça da lista, não saberíamos qual o nó de início da mesma, e não poderíamos voltar a percorrer toda a lista, acedendo assim a toda a informação que guardamos na mesma.

Sobre a "cauda da lista" apenas há um aspeto que temos de ter em atenção, que é como este é o ultimo elemento, não havendo portanto nenhum elemento seguinte, devemos deixar o apontador para o próximo nó a NULL, ou seja inicializado mas não utilizado. Assim podemos utilizar este apontador como critério de paragem caso queiramos percorrer toda a lista. Quando chegarmos ao apontador a NULL teremos a certeza de que estamos no fim da nossa lista.

A próxima imagem representa graficamente uma lista com N elementos.

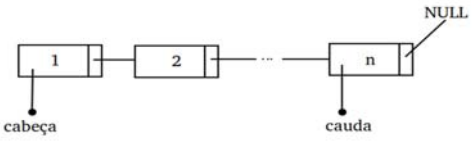


Figura 2 – Exemplo visual da estrutura de uma lista

LISTAS SIMPLESMENTE LIGADAS E EXEMPLO DE IMPLEMENTAÇÃO EM C

Nada melhor para compreendermos as listas do que passarmos à ação. Vamos definir um problema para resolver: Vamos supor que uma escola precisa de um sistema básico para registar os seus alunos. O nosso programa de exemplo deverá ter três opções, uma para adicionar alunos, outra para remover alunos e outra ainda para consultar a lista dos alunos por ordem numérica. Para simplificar a implementação o programa, por questões de gestão de memória, deve utilizar listas simplesmente ligadas e a ordenação dos alunos na lista deve ser feita a cada novo registo, ou seja o número de aluno é dado sequencialmente conforme os alunos vão sendo registados no sistema.

Passemos então à implementação do código em Linguagem C:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define TAM 100

typedef struct no{
    int numero; // para guardar o número de aluno
    char nome[TAM];
    struct no *nseg;
} No; //cada nó representa um elemento da lista

//Função para inserir um novo elemento na lista
No* insereNaLista(No *lista, No *novoNo){
      No *cabeca=lista;
    No *auxiliar;
      //caso a lista esteja vazia ainda (neste caso
//tem o apontador a null como foi explicado no
//texto do artigo)
      if(lista==NULL) return novoNo;
      //caso o primeiro elemento seja maior
      //que o que vamos inserir
      if(lista->numero > novoNo->numero){
            novoNo->nseg=lista;
            return novoNo;
            }
    while(lista!=NULL){
    auxiliar=lista;
      lista=lista->nseg;
        //inserir no fim da lista
        if(auxiliar->nseg==NULL){
            auxiliar->nseg=novoNo;
            return cabeca; //retorna a cabeça
        }
        //inserir no meio da lista
      if(lista->numero >= novoNo->numero){
        auxiliar->nseg=novoNo;
        novoNo->nseg=lista;
        return cabeca;
      }
    }
      return cabeca; //devemos sempre retornar o
//apontador para a cabeça da lista de forma a não
//perdermos o acesso aos dados
}


//Função para Remover um elemento da Lista - A
//função recebe como parâmetros o numero do aluno a
//remover e um apontador para a cabeça da lista de
//forma a podermos percorrer toda a lista
No* removeDaLista(int numero, No *lista){
    No *cabeca=lista;
    No *auxiliar;

    //lista vazia
    if(lista==NULL) return lista;
    //caso seja o primeiro elemento da lista a
    //remover
    if(lista->numero==numero){
        cabeca=lista->nseg;
        free(lista);
        return cabeca;
    }

    while(lista!=NULL){
    auxiliar=lista;
    lista=lista->nseg;
    //caso em que se remove o ultimo elemento
    if(auxiliar->nseg==NULL){
            free(auxiliar);
            return cabeca; //retorna a cabeça
        }

        //remove elemento no meio da lista
      if(lista->numero == numero){
        auxiliar->nseg=lista->nseg;
        free(lista);
        return cabeca;
      }
    }
    return cabeca;
}

//Função para alocar a memória para criar um novo
//nó da lista - recebe como paramêtros os dados do
//aluno em questão
No* criaNo(char *nome,int numero,No *lista){
    No *novoNo;
    novoNo =(No *) malloc(sizeof(No));
    novoNo->nseg=NULL;
    novoNo->numero=numero;
    strcpy(novoNo->nome,nome);
    lista=insereNaLista(lista, novoNo); //chama a
//função insereNaLista para inserir o novo elemento
    return lista;
}


//Função para mostrar no ecrã os dados da lista ao
//utilizador
void imprimeLista(No *lista){
      while(lista!=NULL){
            printf("Numero: %d",lista->numero);
            printf("\tNome: ");
            puts(lista->nome);
          lista=lista->nseg;
      }
}

//Programa Principal
int main(){
    int menu=1;
    int numeroAluno;
    char nome[TAM];
    No *lista=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);

        if(menu==1){
            printf("Nome: ");
            scanf("%s",nome);
            printf("Numero: ");
            scanf("%d",&numeroAluno);
            printf("\n");
```

```
            lista=criaNo(nome,numeroAluno,lista);
        }

        if(menu==2){
           printf("Numero do aluno a remover: \n");
            scanf("%d",&numeroAluno);
            lista=removeDaLista(numeroAluno,lista);
        }

        if(menu==3) imprimeLista(lista);

        if(menu==4) menu=0;

    }
    return 0;
}
```

Como o leitor pode verificar, as funções para adicionar e remover alunos na lista são um pouco complexas, isto porque precisamos tratar as quatro situações que nos pode ocorrer neste problema, isto é: O caso da lista estar completamente vazia, o caso de querermos inserir ou remover um elemento que seja a cabeça da lista, o caso em que queremos remover a cauda da lista, e ainda o caso em que queremos inserir ou remover um novo elemento no meio da lista.

No nosso programa de exemplo, o utilizador do sistema, poderia remover qualquer elemento que desejasse. É importante que todos os casos que possam existir sejam tratados para que possamos assegurar o bom funcionamento do nosso programa.

Neste problema de exemplo, tornar-se-ia mais simples e como o leitor pode verificar, as funções para adicionar e remover alunos na lista são um pouco complexas, isto porque precisamos tratar as quatro situações que nos pode ocorrer neste problema, isto é: O caso da lista estar completamente vazia, o caso de querermos inserir ou remover um elemento que seja a cabeça da lista, o caso em que queremos remover a cauda da lista, e ainda o caso em que queremos inserir ou remover um novo elemento no meio da lista.

No nosso programa de exemplo, o utilizador do sistema, poderia remover qualquer elemento que desejasse. É importante que todos os casos que possam existir sejam tratados para que possamos assegurar o bom funcionamento do nosso programa.

Neste problema de exemplo, tornar-se-ia mais simples e vantajoso utilizar as listas para guardar informação, uma vez que uma das funções do programa elimina a informação desnecessária dos alunos. Caso usássemos arrays, teríamos que eliminar a estrutura da posição correspondente e teríamos que realocar o array, além de termos que mover os restantes elementos do array para a posição interior (de forma a não ficarmos com espaços em branco no array). Usando as listas, apenas precisamos de alterar a posição da memória para qual o apontador do elemento anterior está a apontar. Depois disso, usamos apenas a função free para libertar o espaço de memória que tínhamos alocado previamente para o elemento eliminado.

Espero ter agradado os leitores e gostaria também de vos informar que futuramente irei escrever o próximo artigo sobre estruturas de dados, desta vez com listas duplamente ligadas.

> ***"cada elemento da lista tem a indicação de qual é o elemento que se segue na lista, quando queremos aceder ao próximo elemento de uma lista, basta vermos para que parte da memória está a apontar o apontador."***

Bibliografia

Websites:

http://www.di.ubi.pt/~hugomcp/estruturas/ (12/07/2013)

http://pt.wikipedia.org/wiki/Lista_simplesmente_ligada (12/07/2013)

AUTOR

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

www.cdgramos.com

Pascal – tipo de dados Variant e tipos genéricos

São várias as linguagens cujas variáveis são dinâmicas pelo facto de não terem um tipo de dados bem definido na hora da compilação, sendo antes verificados em *runtime*. Em várias ocasiões, é útil ter uma variável cujo tipo de dados varie, e nas linguagens estaticamente tipadas isso pode tornar-se um pouco trabalhoso ou mesmo impossível.

Neste campo, Pascal tem vindo a evoluir, chegando a um pequeno conjunto de soluções relativamente simples e muito versáteis: o **tipo de dados *Variant***, que é uma forma de tipagem dinâmica, e os **tipos de dados genéricos**, que permitem abstrair certos conceitos que poderão mais tarde ser especializados – falaremos deste aspecto mais a fundo na continuação do artigo. Note-se que os tipos genéricos são, mais especificamente, **classes genéricas**.

Todo o artigo centra-se no *Free Pascal* (versão 2.6.0) (FPC), podendo haver diferenças para o Delphi e/ou outros compiladores.

Tipo de dados *Variant*

O tipo de dados *Variant* é isso mesmo: um tipo de dados que varia. É suportado nativamente pela *unit system* desde a versão 1.1 do FPC, mas para tirar o máximo proveito deste tipo de dados recomenda-se o uso da *unit* ***variants***. O verdadeiro tipo de dados que assume só é definido em *runtime* e pode ir mudando ao longo do programa.

```
uses variants;
var V : variant;
```

Nem todos os tipos de dados podem ser atribuídos a um *variant*. Por exemplo, um *record* não poderá ser atribuído a uma variável do tipo *variant*. O mesmo se aplica a *sets*, *arrays*, *files*, *objects* e classes. De resto, qualquer tipo de dados simples poderá ser directamente atribuído. Portanto, o seguinte programa é totalmente válido pois todas as atribuições feitas também o são:

```
program exemplo;
uses variants;
var V : variant;
    R : real;
    I : integer;
    W : word;
    I64 : Int64;
    S : string;

begin
  V := R;
  V := I;
  V := W;
  V := I64;
  V := S;
end.
```

Contudo, há uma excepção para o caso das *arrays*. Apesar de não ser possível atribuir directamente uma *array* a uma variável do tipo *variant*, esta poderá conter uma *array* desde que seja criada pela função **VarArrayCreate**.

A *unit system* contém um pequeno conjunto de procedimentos e funções que permitem que uma variável *variant* se torne numa *array* e que esta seja manipulada. Analisemos a situação através do seguinte código:

```
var i : integer;
    v : variant;
begin
  v:=VarArrayCreate([1,10], varInteger);
  for i:=1 to 10 do
    v[i]:=i;
  VarArrayRedim(v,20);
  for i:=11 to 20 do
    v[i]:=i;
  for i:=1 to 20 do
    write(v[i], ',');
end.
```

VarArrayCreate é uma função que permite a criação de uma *array* numa variável *variant*, onde lhe é fornecida por argumento a dimensão (neste caso seria uma **array[1..10]**) e qual o tipo de dados armazenado na *array* (**varInteger** é uma constante da *System* que indica que a *array* irá armazenar *integer*'s).

De seguida, podemos redimensionar a *array* com o procedimento **VarArrayRedim**, onde passamos a variável *variant* que representa, neste momento, uma *array*, seguida da nova dimensão:

neste caso ficaremos com uma **array[1..20]**.

Como é possível observar, a partir do momento em que se cria a *array* na variável *variant* com este método, podemos aceder aos seus elementos através dos seus índices como se de uma *array* tradicional se tratasse.

Uma nota importante acerca das variáveis *variant* que assumem *strings*: ao contrário de uma variável *string*, as *variant*'s não permitem o acesso aos caracteres da *string* através dos seus índices. Se o fizermos, será levantada uma excepção do tipo **EVariantInvalidArgError**.

Um pormenor interessante que pode levantar dúvidas prende-se com a atribuição do valor contido numa variável *variant* a outra variável de tipo estático. Isto é possível e traz, claro, alguns pormenores relacionados com a compatibilidade dos tipos de dados e a **conversão implícita** que daí poderá surgir. Vejamos o seguinte exemplo:

```
var V : variant;
    I : integer;
begin
  V := '42';
  I := V;
end.
```

Estamos a atribuir a uma variável do tipo *integer* o valor de

uma variável *variant* que representa, no momento, uma *string*. Esta atribuição é **válida** porque a *string* "42" representa um número inteiro válido, pelo que ocorre a **conversão implícita** para *integer*. Contudo, se a *string* fosse, por exemplo, "42A", a conversão seria impossível, ocorrendo então uma excepção do tipo **EConvertError**.

Este tipo de dados não é propriamente eficiente, como é lógico. O programa, em *runtime*, deverá avaliar constantemente os valores que são atribuídos e manipulados nas variáveis *variant* e isto implica processamento extra. Portanto, havendo tipos estáticos que são mais eficientes, o tipo *variant* deverá ser utilizado apenas quando estritamente necessário.

Tipos genéricos

Já é do conhecimento geral que a família de linguagens Pascal tem pleno suporte ao paradigma OOP com as suas variantes Object Pascal e Delphi. Contudo, o que poucos saberão é que as classes podem ser **genéricas**.

Regra geral, uma classe terá em si uma variável, ou conjunto de variáveis, onde os seus dados estão armazenados e os quais são manipulados com os métodos da classe. Se pretendemos, por exemplo, guardar informações acerca de carros teríamos uma lista do tipo TCarro, se pretendermos guardar informações acerca de pessoas teremos uma lista do tipo TPessoa. Podemos já constatar que ambos estes exemplos têm algo em comum: são listas com o objectivo de registar e manipular dados relativos a objectos. Ora, será natural pensar que os métodos que realizam essas operações são não só equivalentes como muito provavelmente iguais – a única diferença está no tipo dos objectos.

Deparamo-nos, portanto, com o seguinte dilema: pretende-se criar uma classe que possa lidar com um conjunto de objectos, mas estes objectos podem ser de qualquer tipo. O primeiro pensamento do programador será "preciso de implementar uma classe para cada tipo de dados".

E aqui é onde o programador estará equivocado.

Uma **classe genérica** é uma classe que está preparada para trabalhar com dados de qualquer tipo, tipo este que é informado à classe por um processo designado **especialização**. Portanto, há duas fases:

1. **Implementação** da classe genérica;
2. **Especialização** da classe genérica com a criação de um novo tipo de dados.

Ou seja, criamos a classe para que ela trabalhe com um tipo de dados genérico **T**, e mais tarde o programador que precisar da classe **especializa-a**, indicando o que é que este tipo **T** deverá ser: poderá ser um *Integer*, um *Real*, uma *String*, uma *Array*… ou até mesmo uma classe!

A sintaxe da declaração de uma classe genérica em *Free Pascal* é a seguinte:

```
type generic TClasse<T> = class
```

A *keyword* **generic** indica que a classe é genérica, e à frente do nome da classe, entre os sinais de maior e menor, damos o nome ao **tipo genérico**, neste caso **T** (que é o nome dado mais comum).

Quando o programador necessitar da classe, deverá indicar um novo tipo de dados onde a classe genérica é **especializada**, sendo a sintaxe a seguinte:

```
type TClasseEspecializada = specialize
                                    TClasse<Tipo>;
```

Tipo não poderá ser uma classe genérica, mas poderá ser uma classe especializada. Ou seja, se especializarmos uma classe genérica num novo tipo de dados, este tipo poderá ser utilizado para especializar outra classe genérica. Contudo, uma classe genérica não poderá ser utilizada *directamente* para a especialização de outra classe genérica.

Colocada a teoria acerca de tipos genéricos e variantes, vamos aplicar estes conceitos na prática.

Implementação de uma *stack* com recurso a tipos genéricos

Para quem programa noutras linguagens há uma pergunta que poderá surgir: faz sentido implementar uma *stack* recorrendo a uma classe genérica? Não há forma de dizer que não. Aliás, é mesmo uma estratégia com bastantes vantagens. A utilização de métodos mais tradicionais implicaria, de uma forma geral, a implementação de uma *stack* para cada tipo de dados necessário. A utilização de uma *stack* com dados *variant* não seria eficiente nem desejável – se pretendemos ter uma *stack* cujos dados sejam do tipo *Integer*, esta solução não seria, de todo, a melhor.

Portanto, as classes genéricas oferecem a possibilidade de implementar a *stack* independentemente do tipo de dados com que vá lidar, havendo posteriormente a sua especialização. Isto significa que, em vez de haver a implementação repetitiva de *stacks* para diferentes tipos, há uma implementação genérica de uma classe que poderá ser especializada nos tipos que forem necessários.

Para começar, a classe terá, então, a seguinte definição:

```
type generic TStack<T> = class(TObject)
```

Esta será uma classe que irá herdar de *TObject* e terá por base uma *array* dinâmica para armazenar os valores.

A *stack* deverá aumentar de tamanho à medida que lhe são adicionados elementos. Para tornar esta expansão eficiente, sempre que se tentar adicionar um elemento e a *array* já não tiver capacidade, duplicamos-lhe o tamanho. Isto é eficiente não só por causa do funcionamento em memória do procedimento **SetLength** que iremos utilizar para redimensionar a

array como também leva a um muito menor número de redimensionamentos (não há nova alocação de memória sempre que se acrescenta um elemento à *stack*).

Necessitaremos também de uma variável privada que nos indique quantos elementos tem a *stack* no momento (ou seja, quantos elementos tem a *array*).

O construtor da classe oferecerá duas opções ao programador: ele poderá indicar ou não quantos elementos deverá ter a *stack* inicialmente. Caso ele não o defina, um tamanho será atribuído por defeito, tamanho este representado pela constante *INITSIZE*.

Por fim, necessitamos dos métodos típicos de uma *stack*: ***pop*** e ***push***. Adicionalmente criaremos uma função que informa se a *stack* está vazia: ***IsEmpty***.

```
type generic TStack<T> = class(TObject)
 private
  const INITSIZE = 32;
  var data : array of T;
      top  : word;
 public
  function Pop : T;
  procedure Push(value : T);
  function IsEmpty : boolean;
  constructor Create;
  constructor Create(limit:word); overload;
end;
```

Resta-nos implementar estes métodos. Comecemos pelos construtores:

```
constructor TStack.Create();
begin
  self.Create(self.INITSIZE);
end;

constructor TStack.Create(limit:word);
overload;
begin
  if limit < 1 then
    limit := 1;
  SetLength(self.data, limit);
  self.top := 0;
end;
```

Como vimos antes, o programador pode definir um tamanho inicial para a *stack* ou não, e no caso de não o definir, esta terá um tamanho por defeito, indicado com a constante privada *INITSIZE*.

Podemos também já definir a função **IsEmpty** pela sua simplicidade: simplesmente verificamos se ***top*** é igual a zero, o que indica que não há elementos na *stack*.

```
function TStack.IsEmpty:boolean;
Begin

  IsEmpty := (self.top = 0);

end;
```

De seguida vamos implementar o método ***push*** que vai permitir adicionar um elemento à *stack*. Como foi referido, se a *array* não tiver tamanho suficiente, a sua dimensão será duplicada:

```
procedure TStack.Push(value : T);
begin
  inc(self.top);
  if self.top > length(self.data) then
    SetLength(self.data, length(self.data)*2);
  self.data[self.top-1] := value;
end;
```

Ou seja, aumentamos ***top*** em uma unidade, verificamos o tamanho da *array* e atribuímos o valor, fornecido no argumento ***value***.

Por fim, só nos resta implementar a função ***pop***. Esta função vai retornar o último valor da *stack*, decrementando o indicador ***top***. No caso de o programador realizar um ***pop*** indevido (quando a *stack* está vazia), ele deverá lidar com o erro que daí surgir – há um método *IsEmpty* que o auxilia, pelo que não está no âmbito do artigo a gestão deste erro. Traduzindo em código:

```
function TStack.Pop : T;
begin
  Pop := self.data[self.top-1];
  dec(self.top);
end;
```

Temos, portanto, a nossa stack implementada. Só nos resta testá-la.

```
{$mode objfpc}
program Teste_Stack;
// Código da classe aqui

type TIntegerStack =
     specialize TStack<Integer>;
var s : TIntegerStack;
    i : integer;

begin
  s := TIntegerStack.Create(5);
  for i in [1..10] do
    s.Push(i);
  while not s.IsEmpty do
    write(s.Pop, ', ');
  writeln('FIM');
  s.Free;
  readln;
end.
```

Especializámos a *stack* para lidar com dados do tipo ***Integer***, e indicámos que estamos a prever que esta tenha 5 elementos, o que, neste caso, não se concretiza – vamos testar a capacidade de autoexpansão da *stack*. Fizemos, portanto, *push* de números de 1 a 10 por ordem, e de seguida ordenámos o esvaziamento completo por *pop* encaixado numa estrutura *while* que termina quando a *stack* estiver vazia (verificado pela função *IsEmpty*), escrevendo os valores no *output*. Eis o resultado deste programa:

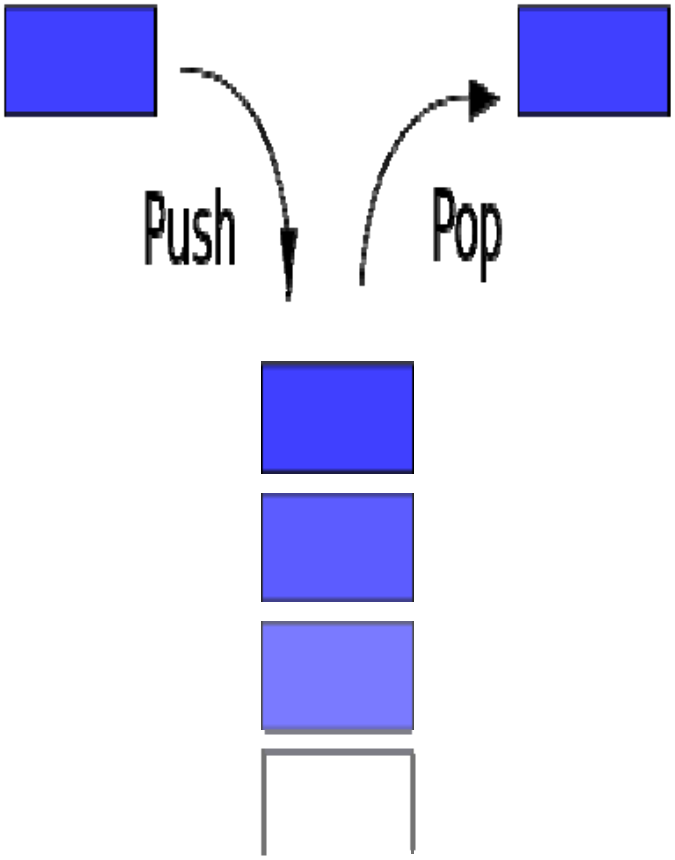
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIM

> ***(...)tipo de dados Variant, que é uma forma de tipagem dinâmica, e os tipos de dados genéricos, que permitem abstrair certos conceitos que poderão mais tarde ser especializados (...)***

Como podemos constatar, a *stack* está a cumprir o seu propósito LIFO (*Last In First Out*) – o último elemento a entrar é o primeiro a sair –, bem como foi capaz de se redimensionar de forma a acomodar todos os valores.

> ***as classes genéricas oferecem a possibilidade de implementar a stack independentemente do tipo de dados com que vá lidar***

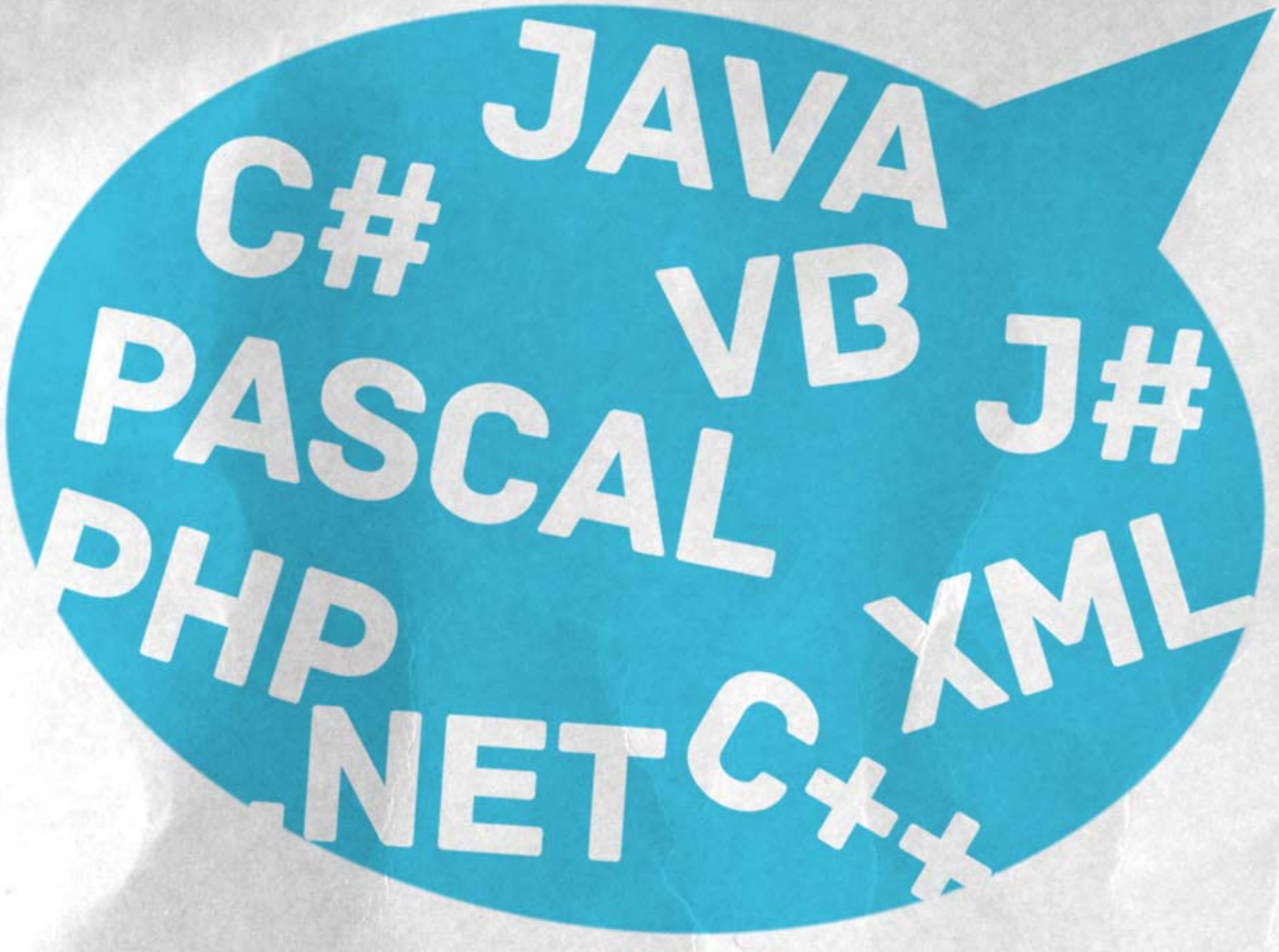
Por curiosidade, podíamos ter especializado a classe com o tipo ***variant*** – ou seja, a *stack* podia literalmente conter elementos de qualquer tipo de dados simples. Isto revela, em última instância, a versatilidade dos tipos variantes e genéricos não só em separado como em conjunto.



AUTOR

Escrito por Igor Nunes

Curioso na área das tecnologias e em especial da programação, tem um grande interesse pelas tecnologias de ensino da Texas Instruments™ e pelo TI-Basic, além de ter uma saudável relação com o Pascal. É conhecedor de outras linguagens de programação, como VB.NET. No P@P, é membro da Wiki Team e Moderador Local dos quadros de Pascal e Delphi/Lazarus. Escreveu o novo Tutorial de Pascal, recentemente disponibilizado em PDF.



ENTÃO, SÓ FALAS EM CÓDIGO?

TEMOS O REMÉDIO CERTO PARA TI!

portugal-a-programar.pt

A MAIOR COMUNIDADE PORTUGUESA DE PROGRAMAÇÃO, APARECE!

COLUNAS

Visual (Not) Basic - Eventos e handlers

EVENTOS E HANDLERS

Eventos, interrupts, triggers, handlers, listeners, consumers… existem muitos nomes e muitos conceitos mas redundam todos na mesma filosofia.
O conceito de evento é tão essencial, elementar e intrínseco que tão depressa os encontramos desde o nível de comunicação do hardware até ao software mais complexo do mundo, como também encontramos quem os use diariamente e nunca tenha ouvido falar deles.

Se pensarmos numa aplicação sem interface gráfico, apenas consola, e se assumirmos que é uma aplicação "standalone", todas as instruções que são escritas são interpretadas na sua sequência original. Avança-se para a próxima instrução porque a anterior terminou, e assim sucessivamente. Se for necessária introdução de dados do utilizador, por exemplo, espera-se que termine e só depois se avança.

GUIs. O melhor exemplo "event-driven"

Conseguem imaginar este "pára-arranca" num interface gráfico?
Esperar que o utilizador carregue no OK? E se ele quiser carregar no Cancel? Ouvir um MP3 até ao fim porque não o posso parar enquanto não terminar? Não dá!
Fazemos então a coisa ao contrário: "AVISA-ME TU quando fores carregado", "AVISA-ME TU quando terminares". Fazer com que surjam notificações de algo que aconteceu, no momento em que aconteceu, e reagir-lhes imediatamente.

Um interface gráfico é um excelente exemplo disto: existe um vasto leque de potenciais situações para cada um dos componentes.

Um botão, por exemplo,. Vejamos a que tipo de eventos, numa análise preliminar, está um botão sujeito:

- Selecção
- Clique
- Entrada de cursor
- Saída de cursor
- Tecla pressionada
- Alvo de drag&drop

E isto para um botão apenas.
A melhor solução é estar com atenção, escutar, quando um destes eventos ocorrer, para fazer o que se tem de fazer. Será necessária uma forma de indicar que ações queremos tomar na eventualidade de um dos eventos ter sido disparado, ter acontecido.

À escuta de um evento, lidar com um evento

Em Visual Basic, existem duas formas de registar um evento. Uma forma é delegar diretamente a um método, com a keyword Handles:

```
Private Sub Button1_Click(sender As Object,
                          e As EventArgs) _
                        Handles Button1.Click
    'botão carregado. Fazer coisas!
End Sub
```

Este é o método mais comum, e que é automaticamente implementado quando no IDE se faz duplo clique sobre um botão, por exemplo.
Está basicamente a assinalar aquele método como sendo o método que vai tratar aquele evento.

Seguindo o mesmo raciocínio, e imaginando que o Button1, Button2 e Button3 fazem rigorosamente o mesmo, podemos delegar o mesmo handler para um único evento. Dessa forma, quando o evento acontecer em qualquer um dos 3 botões, o método vai ser chamado:

```
Private Sub Button1_Click(sender As Object,
                          e As EventArgs) _
                        Handles Button1.Click,
                        Button2.Click,
                        Button3.Click
    If sender Is Button1 Then MessageBox.Show
                                        ("botão1")
    If sender Is Button2 Then MessageBox.Show
                                        ("botão2")
    If sender Is Button3 Then MessageBox.Show
                                        ("botão3")
End Sub
```

Também é possível registar diferentes tipos de eventos para o mesmo handler.

A outra forma é através de registo explícito:

```
AddHandler Button1.Click, AddressOf
                                    Button1_Click
```

O múltiplo registo, como em cima, é possível se registarmos mais handlers:

```
AddHandler Button1.Click, AddressOf Button1_Click
AddHandler Button2.Click, AddressOf Button1_Click
AddHandler Button2.Click, AddressOf Button1_Click
```

O handler para estes registos tomaria a seguinte forma:

```
Private Sub Button1_Click(sender As Object,
                                    e As EventArgs)
        'botão carregado. Fazer coisas!
End Sub
```

Atenção: **Button1_Click** é apenas um nome. O handler de um evento não qualquer tipo de vínculo com o nome do mé-

todo.

Poderia chamar ZeManel(sender As Object, e As EventArgs) se necessário.
O único vínculo incide na assinatura do método.

Esta forma permite também outros caminhos para definir o Delegate, como por exemplo, um método anónimo:

```
AddHandler Button1.Click, Sub(s As Object, ev As
                                        EventArgs)
'botão carregado. Fazer coisas!
End Sub
```

Eventos, os meus eventos, e como os erguer

Não demoraria muito tempo a "brincar" com handlers até fazer uma pergunta a si mesmo: Eu posso criar os meus eventos?

A resposta é um sonante sim. Lembremo-nos que todos esses objectos bonitos, como o Button, que já damos como garantidos em todas as versões do IDE, não passam de implementações padrão.
Se nada o impede de criar um botão inteiramente seu, então nada o vai impedir de criar um leque de eventos para o mesmo.

Vamos considerar a seguinte classe que trata de fazer download do conteúdo de uma página e devolver o número de caracteres:

```
Public Class DownloadX

    Public Shared Sub ContarCaracteres(Pagina As
                                            Uri)
        Dim N As New Net.WebClient
        AddHandler N.DownloadStringCompleted, _
            Sub(sender As Object, r As
             Net.DownloadStringCompletedEventArgs)
                Dim Resposta As String =
                                    r.Result.Trim
            End Sub
        N.DownloadStringAsync(Pagina)
    End Sub

End Class
```

Como se pode observar, logo no exemplo estamos a utilizar um registo de handler com um método anónimo como delegate.
Se chamarmos o método **DownloadX.ContarCaracteres**, não vai acontecer nada... que seja visível.
O pedido está de facto a ser feito e a resposta a chegar, mas não estamos a notificar nada nem ninguém do que lhe fazer quando chega.

Para todos os que estão a este ponto a pensar que bastaria um **Return Resposta**, pensem novamente.
O pedido é assíncrono, ou seja, nem vai bloquear a thread do interface do utilizador, nem se pode prever quando termine. Pode ser instantâneo ou demorar dezenas de segundos, que não queremos esperar.
Isto significa que ao chamar **DownloadX.ContarCaracteres**, a aplicação não vai ficar congelada à espera. Poderia, pelo contrário, fazer mil destas chamadas que o interface não sofreria, mas as coisas estão a acontecer.

Como fazer então para que consiga notificar no meu form (por exemplo) quantos caracteres existem afinal na resposta?
A resposta é óbvia. Vamos colocar um evento. Vamos colocar dois, aliás. Ora observem:

```
Public Class DownloadX

    Public Shared Event ContagemPronta(Contagem As
                                            Integer)
    Public Shared Event Erro(Excepcao As
                                        Exception)

    Public Shared Sub ContarCaracteres(Pagina As
                                            Uri)
        Dim N As New Net.WebClient
        AddHandler N.DownloadStringCompleted, _
            Sub(sender As Object, r As
             Net.DownloadStringCompletedEventArgs)
                If r.Error Is Nothing Then
                    Dim Resposta As String =
                                    r.Result.Trim
                    RaiseEvent ContagemPronta
                                (Resposta.Count)
                Else
                    RaiseEvent Erro(r.Error)
                End If
            End Sub
        N.DownloadStringAsync(Pagina)
    End Sub

End Class
```

Existem pelo menos dois novos elementos importantes:

A keyword Event, em cima, serve para especificar o tipo de eventos que esta classe pode levantar. Deve-se especificar a assinatura a levantar. Neste caso específico, uso a keyword Shared porque o levantamento origina de um método Shared, mas é rigorosamente igual para instâncias.

```
Public Shared Event ContagemPronta(Contagem As
                                        Integer)
```

O parâmetro Contagem vai transportar o valor no momento do disparo, assim como:

```
Public Shared Event Erro(Excepcao As Exception)
```

O parâmetro Excepcao vai transportar o erro oriundo do WebClient.

A partir deste momento, a classe DownloadX já possuí dois eventos que podem ser registados, mas de pouco servem se não forem disparados nos momentos chave.

```
RaiseEvent ContagemPronta(Resposta.Count)
```

O statement RaiseEvent faz disparar um evento do mesmo contexto da classe.
Indica-se não só o evento a levantar (ContagemPronta) co-

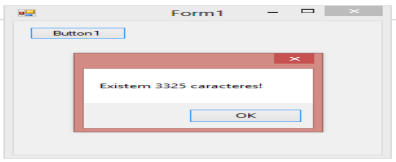
EVENTOS E HANDLERS

mo se envia o valor que se quer comunicar que é o número de caracteres da resposta.

Da mesma forma, quando se detecta que o WebClient vem em erro, dispara-se o evento de erro.

Nos locais onde estão registados, os RaiseEvent vão causar a notificação e fazer com que os handlers executem com os parâmetros levantados.

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) _
        Handles Button1.Click
        AddHandler DownloadX.ContagemPronta, _
            Sub(Quantos As Integer)
                MessageBox.Show("Existem " & Quantos & " caracteres!")
            End Sub

        AddHandler DownloadX.Erro, _
            Sub(Err As Exception)
                MessageBox.Show("ERRO!" & vbNewLine & Err.Message)
            End Sub

        DownloadX.ContarCaracteres(New Uri(
                                   "http://www.sergioribeiro.com"))
    End Sub
End Class
```



E com isto apenas, foi efectuada uma comunicação de dados entre dois objectos distintos através de eventos. Dois quaisquer objectos. Poderiam até ser duas forms que precisam de ser separadas mas com constante comunicação. À medida que se vão entendendo os eventos, começa-se a perceber que estão e são usados em muitos mais sítios do que se imagina.

Entender isto faz também levantar uma dúvida pertinente… e se o método não for Shared e existirem 1000 instâncias do objecto? Como é que os handlers vão distinguir os objectos? Se voltarmos acima e fizermos a mesma pergunta com o handler do botão, vamos reparar que existem dois parâmetros no handler. Isto diz-nos que o evento Click é levantado com dois parâmetros: um **sender** do tipo **Object** e um **e** do tipo **EventArgs**.
O EventArgs serve para comunicar uma série de estados do objecto, e vamos ignorar.
O segredo da distinção está no **sender**. Internamente, o RaiseEvent do button deverá ser algo como:

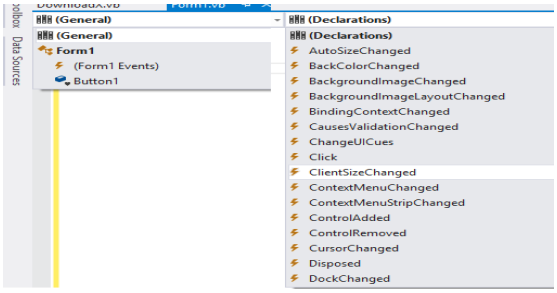
```
RaiseEvent(Me,_eventargs)
```

Onde o **Me** se refere ao próprio objecto. Se o Raise é feito a partir do próprio objecto, então o parâmetro **sender** representa o objecto origem do disparo. É aqui que o handler os distingue.

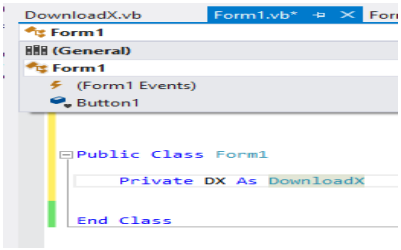
```
If sender Is Button1 Then MessageBox.Show
                                    ("botão1")
If sender Is Button2 Then MessageBox.Show
                                    ("botão2")
If sender Is Button3 Then MessageBox.Show
                                    ("botão3")
```

O que é que o meu IDE pode fazer para facilitar?

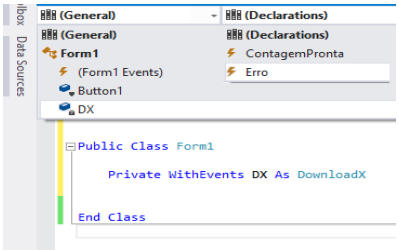
Voltando a pegar nos botões, quando os adicionamos a um form por exemplo, o IDE dá uma ajuda listando essa instância na dropdown de objectos, e os seus respectivos eventos e métodos na dropdown dos membros:



No entanto, se eu criar o botão ou outro qualquer objecto em código, o IDE não me permite ter acesso aos membros da mesma forma:



É possível dar uma pequena dica ao IDE e indicar que queremos ver os membros deste objecto, simplesmente por acrescentar a keyword **WithEvents**:



(Os eventos são representados no IDE por um relâmpago amarelo.)

Ao clicar, por exemplo no evento ContagemPronta, o IDE gera o handler como gera para o botão:

```
Private Sub DX_ContagemPronta(Contagem As Integer)
                            Handles DX.ContagemPronta
End Sub
```

AUTOR

Escrito por Sérgio Ribeiro

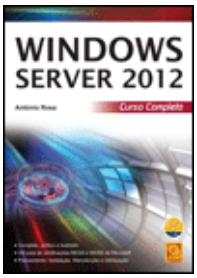
Curioso e autodidata com uma enorme paixão por tecnologias de informação e uma saudável relação com a .NET Framework. Moderador global na comunidade Portugal@Programar desde Setembro de 2009. Alguns frutos do seu trabalho podem ser encontrados em http://www.sergioribeiro.com

Media Partners da Revista PROGRAMAR

Análises

Windows Server 2012–Curso Completo

Windows Server 2012–Curso Completo



Título: Windows Server 2012
Curso Completo

Autor: António Rosa

Editora: FCA

Páginas: 688

ISBN: 978-972-722-753-2

Embora esteja ligado à programação, nunca aprofundei muito o tema dos servidores. Estava esperançoso quanto a este livro que superou bastante as minhas espectativas.

Como é habitual, quando temos um livro, começamos por analisar a sua capa e contracapa e logo comecei a tomar interesse pelo mesmo. Por fora, aparenta ser um pouco "maçudo" para quem não está habituado a ler, mas essa ideologia acaba por desvanecer.

Começamos a "folhear" e deparamo-nos com um extenso índice de 14 páginas que só nos faz pensar "afinal é mesmo um curso completo". E é mesmo!

O leitor pode ficar um pouco receptivo por não ter bases de servidores, nunca ter operado com um Windows Server, mas não é verdade. Este livro destina-se a vários níveis académicos e de curiosos, pois mesmo "não percebendo da arte", é tudo explicado de forma bastante clara e directa, com a ajuda de ilustrações para nos "localizarmos" caso estejamos a fazer a instalação ou configuração do servidor propriamente dito, lado a lado com a leitura do livro.

É uma leitura bastante cativante e útil desde para quem quer fazer uma certificação, administradores de redes Windows Server 2012, estudantes dos mais diversos níveis de ensino, cursos de formação e meros interessados em instalar e gerir uma rede destas.

Outro ponto forte do autor António Rosa, e há que dar mérito por isso, é a capacidade escrita que cativa o leitor, principalmente "nas piadas para quebrar o gelo" que nos fazem rir a olhar para um livro que não tem nada a ver com comédia, e continuar a leitura com mais ânimo.

É um livro dividido por temas, cada um com o seu peso no produto final, cada tema com descrições sucintas de todo o seu conteúdo, utilizando os termos Ingleses juntamente com o termo em Português, bastante ilustrado em todos os passos importantes.

Após 651 páginas de leitura, contemplamo-nos com o glossário de termos em Inglês, Português e Português do Brasil, aumentando assim o seu público alvo, seguido dum índice remissivo.

No geral, e como nota final, é um livro que aconselho a todas as pessoas que queiram ou precisam de saber mais sobre este grande Windows Server 2012, pois está muito bem construído, nota-se que o planeamento foi cuidadoso, uma linguagem bastante simples mas precisa e focalizada, que refletiu um resultado geral fantástico!

Boas leituras!

> ***"(...) a capacidade escrita que cativa o leitor, principalmente "nas piadas para quebrar o gelo" que nos fazem rir a olhar para um livro(...)"***

AUTOR

Escrito por Nelson Belo Lima

Programador Especialista de Sistemas de Informação (Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Viana do Castelo), Técnico de Mecatrónica Automóvel e Mecânico Perito Citroen. Apaixonado pela tecnologia, conciliando o lazer com a vertente profissional, fui melhorando o meu conhecimento e aumentando experiência em vários pontos de interesse.

bit.ly/nelsonbelolima

Redes de Computadores - Curso Completo

Título: Redes de Computadores – Curso Completo (10.ª Edição Atualizada e Aumentada)

Autor: José Gouveia / Alberto Magalhães
Editora: FCA

Páginas: 368

ISBN: 978-972-722-781-5

Para mim que já li uma quantidade considerável de livros sobre redes de computadores, este foi de certa forma uma surpresa, pela forma como aborda os temas, de forma simples e ainda assim aprofundada o suficiente para os autodidactas, atingirem aquilo que é proposto no livro.

De inicio, para quem está mais interessado em “hands-on”, pode parecer um pouco massivo, apesar dos conceitos serem de extrema importância. No entanto, com o decorrer da leitura o livro, acaba por se entusiasmar, com a pratica.

A partir do capitulo 4, começa-se uma verdadeira “saga” hands-on em que o leitor pode seguir todos os exemplos passo a passo, utilizando uma versão trial do sistema operativo que é utilizado nos exemplos.

Foi agradável ler a parte sobre Active Directory (LDAP), ainda que na minha opinião pudesse ser apresentado o OpenLDAP, para sistemas *nix, de forma a não deixar o leitor com a sensação de que esteve a ler um livro inteiro sobre redes Microsoft. Da mesma forma que me agradou ver a apresentação sobre DHCP, ainda que de igual modo, muito focado no sistema operativo Windows.

O oitavo capitulo por si só dava quase um livro! Foi bastante agradável de ler e parece-me útil para qualquer pessoa que pense em configurar uma rede, por mais pequena que seja, pois grande parte dos “segredos” da configuração de uma rede que se ligue à internet, está na configuração correcta dos routers.

Ao ler o sétimo capítulo, confesso que estava muito expectante. Não sabia bem o que iria encontrar, não me tinha acautelado para o que iria encontrar e estava ansioso por ler o que lá estava.

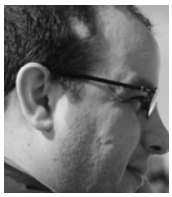
Ainda que de forma superficial pois este é mais um capitulo que por si só daria um excelente livro, aborda de uma forma objectiva os principais temas relacionados com a redundância a falhas em redes, e os aspectos básicos da segurança de uma rede. Não se limita a focar os aspectos dos dispositivos de rede, mas foca-se também na salvaguarda dos dados e na tolerância a falhas de hardware, o que me pareceu de elevada importância e interesse.

Os oitavo e novo capítulos quase davam para fazer um só, de qualquer das formas são uma leitura bastante agradável e simples, ainda que para os utilizadores mais experientes possam parecer um pouco “aborrecidos”, para os menos experientes, é uma leitura quase “obrigatória”.

Por fim no ultimo capitulo, eu estava já com algumas espectativas talvez exageradas. O tema da segurança tem sido de certa forma tratado como “tabu”, pelo que estava à espera de encontrar algo extremamente simplista, sobre testes de segurança defensiva, e não scan de segurança com uma ferramenta tão “poderosa” como o Nessus. Não são efectivamente abordados todos os aspectos de um scan com o Nessus, nem com outras ferramentas de scan de segurança que podem ser uteis, na hora de efectuar um teste a uma rede, antes de finalmente começar a colocar em utilização com utilizadores, dados reais, etc… No entanto pareceu-me bastante interessante a apresentação feita. Simples, objectiva, concisa, e sem fugir do tema principal do livro. Este seria mais um dos temas que daria para se escrever um livro inteiro sobre ele, e talvez não chega-se dada a amplidão do tema.

Em conclusão, é um livro que recomendo aos estudantes dos cursos de informática e aos autodidactas, que não se contentam em ler o manual do software ou do equipamento e pertentem uma leitura mais aprofundada do tema.

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.
.

COMUNIDADES

Comunidade NetPonto – Implementando publicidade usando Nokia NAX em aplicações Windows Phone

IMPLEMENTANDO PUBLICIDADE USANDO NOKIA NAX EM APLICAÇÕES WINDOWS PHONE

Este artigo explica como implementar publicidade baseada em NAX ad (Banner) em aplicações Windows Phone 8.

Introdução

Uma nova forma de ter publicidade em aplicações de Windows Phone é o uso de Nokia Ad Exchange (NAX), fornecida por Inneractive

Se o teu país não tem disponível o Microsoft Ad Exchange, facilmente podes usar o NAX para ganhar algum dinheiro com as tuas aplicações!

NAX é uma "mobile in-app advertising exchange" que oferece acesso às redes de anúncios no topo do mundo. Como uma API e um parceiro, irás ter acesso a mais de 120 agências e redes, e apenas precisas de uma conta de Paypal.

As principais características do NAX são:

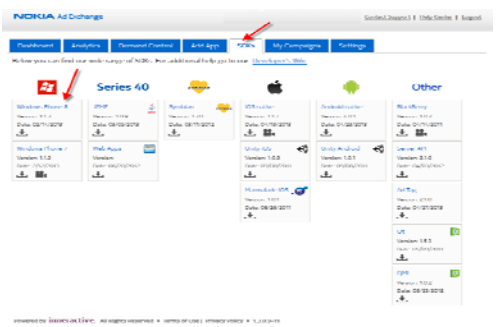
- Otimização através de mais de 120 redes de publicidade
- Pagamento habilitado em mais de 200 países
- Gerenciar suas próprias campanhas publicitárias para promover a sua aplicação
- Poderoso painel de desempenho anúncio
- NAX é gratuito para programadores

Este artigo mostra como implementar publicidade baseada em NAX ad (Banner) na tua aplicação Windows Phone 8. As instruções complementares podem ser consultadas em Windows Phone SDK guidelines (*) no Inneractive wiki.

Passo 1 – Registar e obter o SDK

Primeiro o que é preciso fazer é registar no https://nax.nokia.com.

Depois é preciso fazer o download do SDK para Windows Phone 8, que está disponível na página de SDKs. Atualmente a versão é 1.1.3 (14 February 2013).

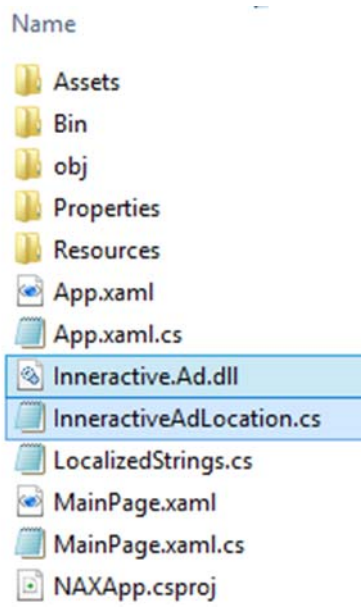


Proponho uma vista de olhos no documento "Ad Placement Strategy.html" que se encontra na pasta da documentação, uma vez que contém sugestões de onde colocar a publicidade (a localização selecionada depende muito de quando depois se irá ganhar por cada publicidade).

Passo 2 – Adicionar ficheiros no projecto

Da pasta **InneractiveAdSDK** que foi extraida do SDK, copiar os ficheiros para a raíz do projecto do Visual Studio (ou colocar numa pasta separada):

- Inneractive.Ad.dll
- InneractiveAdLocation.cs (usar este ficheiro em caso da aplicação estiver localizada)

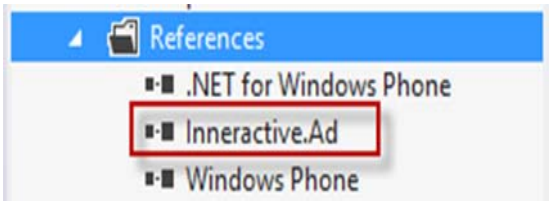


Ficheiros da pasta InneractiveAdSDK

Passo 3 – Adicionar a dll da Nokia NAX

É preciso adicionar a dll Inneractive.Ad.dll', para isso faça References/Add Reference, e clique em Browse para procurar a pasta onde está a dll, para poder adicionar ao projeto esta referecia.

Depois de adicionar obtemos algo do género:



Inneractive.Ad.dll adicionada ao projeto

Note: Caso ao adicionar a referência ocorra o erro "A reference to a higher version or incompatible assembly cannot be added to the project", é preciso desbloquear a dll Para desbloquear a dll clique com o botão do lado direito do rato na **Inneractive.Ad.dll *e selecione*** *Properties* no final do separador, em seguraça, clique em **Unblock** e depois lique em **Apply**.

Passo 4 – Definição dos requisitos

Para podermos usar publicidades NAX é preciso definir/ativar no ficheiro manifest as capabilities que são requisito para esta funcionalidade:

Nas Properties/WMAppManifest.xaml selecionar, na secção de Capabilities:

- ID_CAP_LOCATION
- ID_CAP_NETWORKING
- ID_CAP_WEBBROWSERCOMPONENT
- ID_CAP_PHONEDIALER
- ID_CAP_IDENTITY_DEVICE

Passo 5 – Mostrar publicidade NAX em XAML

Primeiro, é necessário adicionar o controlo em XAML onde a publicidade irá ser apresentada. Iremos ter um nax_control que é um StackPanel que está contido na Grid com o nome ContentPanel:

```
<Grid x:Name="ContentPanel" Grid.Row="1"
                            Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="53"/>
    </Grid.RowDefinitions>
     <ListBox Grid.Row="0">
    </ListBox>
     <StackPanel Height="53" Name="nax_control"
Grid.Row="1">
    </StackPanel>
</Grid>
```

A lista dos tamanhos suportados:

- 300 x 50
- 320 x 53
- 300 x 250 (Rectangulo)
- 320 x 480 (Full Screen)

Iremos usar o tamanho 320 x 53 no fundo da página.

O resultado final é:

Passo 6 – Código C# para adicionar a pubilidade Nokia NAX

Depois de adicionar o control em XAML, iremos criar o código em C# para adicionar o controlo de publicidade. Primeiro iremos adicionar dois namespace na página onde iremos criar o control, neste caso na **MainPage.xaml.cs** :

```
using Inneractive.Nokia.Ad;
using InneractiveAdLocation;
```

é recomendado incluir o namespace:

```
using Microsoft.Phone.Net.NetworkInformation;
```

NetworkInformation é a class que iremos usar para verificar se a existe coneção à internet ou não, usando o método DeviceNetworkInformation.IsNetworkAvailable.

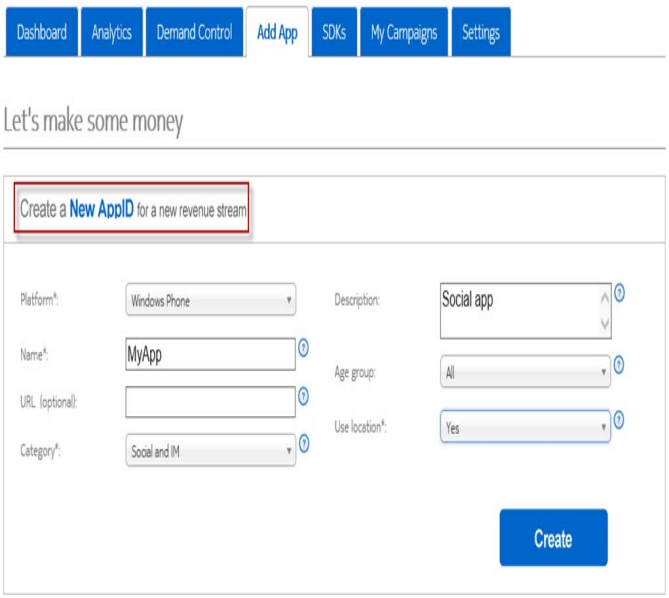
De seguida, define-se então os optionalParams e toda a lógica dos métodos MainPage_Loaded(): e iaLocation_Done ()

```
public partial class MainPage : PhoneApplication-
Page
{
    Dictionary<InneractiveAd.IaOptionalParams,
                            string> optionalParams;

    // Constructor
    public MainPage()
    {
        InitializeComponent();

      // Sample code to localize the ApplicationBar
        //BuildLocalizedApplicationBar();
        this.Loaded += new RoutedEventHandler
                                    (MainPage_Loaded);
    }

    private void MainPage_Loaded(object sender,
                                    RoutedEventArgs e)
    {
        if
(DeviceNetworkInformation.IsNetworkAvailable)
        {
            // Watch location
            IaLocationClass iaLocation = new
                                    IaLocationClass();
            iaLocation.Done += new
           System.EventHandler<IaLocationEventArgs>
                                    (iaLocation_Done);
            iaLocation.StartWatchLocation();

            optionalParams = new
         Dictionary<InneractiveAd.IaOptionalParams,
                                        string>();
            optionalParams.Add
                   (InneractiveAd.IaOptionalParams.
            Key_OptionalAdWidth, "320"); //ad width
            optionalParams.Add
                   (InneractiveAd.IaOptionalParams.
          Key_OptionalAdHeight, "53"); //add height
        }

        //Show Add Banner. Remarks: pay attention
                    to use Application Id from NAX
        if (optionalParams != null)
        {
            InneractiveAd iaBanner = new
                     InneractiveAd("ApplicationId",
            InneractiveAd.IaAdType.IaAdType_Banner,
                              30, optionalParams);
            nax_control.Children.Add(iaBanner);
```

```
        }
    }

    private void iaLocation_Done(object sender,
                                 IaLocationEventArgs e)
    {
        try
        {
            // Add location, if received
            if (e != null && e.location != null)
                optionalParams.Add
                 (InneractiveAd.IaOptionalParams.
                Key_Gps_Coordinates, e.location);
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine
("Error: " + ex.ToString());
        }
    }
}
```

Geração do AppID

Nota: ApplicationId_NAX é gereado do *nax.nokia.com*, na seção Add App.

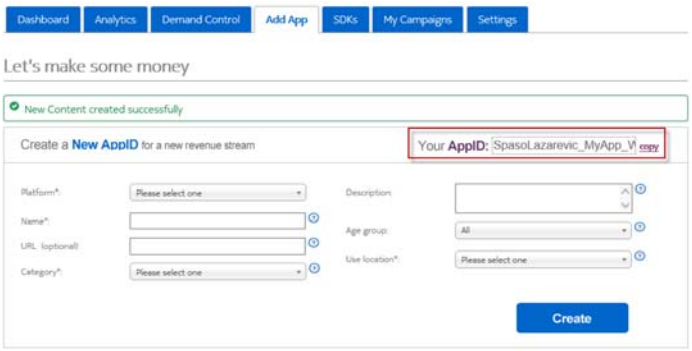


Seção Add App para criar o novo AppID

É preciso incluir a seguinte informação:

- Platforma
- Nome da Plataforma
- Categoria
- Does your app use location

Depois disto foi assim obtido o id da aplicação, isto é crucial para poder seguir todas as impressões no banner que irá aparecer na aplicação do Windows Phone, resultado esse que poderá ser consultado no Nokia NAX dashboard.



Id da Aplicação gerado (AppId), que será usado na aplicação (chamado de ApplicationId_NAX):

Novo AppID

É isto! Eu espero que consiga ganhar algum dinheiro com o Nokia NAX ad.

Referências

- Implementing advert in Windows Phone app using Nokia NAX with C# (Spaso Lazarevic Blog - original source) (*)
- Windows Phone SDK guidelines (Inneractive wiki) (*)
- Nokia Ad Exchange (*)

(*) Artigos disponíveis apenas em inglês

Em conclusão, como podemos verificar é muito simples implementar publicidade Nokia NAX em aplicações Windows Phone, sendo apenas preciso conta Paypal o que facilita todo o processo.

Este artigo é tradução do artigo: Implementing advert in Windows Phone app using Nokia NAX with CSharp, escrito por Spaso Lazarevic, como contributo à comunidade portuguesa para a Nokia Developer Wiki.

AUTOR

Escrito Por Sara Silva

É licenciada em Matemática – Especialidade em Computação, pela Universidade de Coimbra e é Microsoft Certified Profissional 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

Navicat Premium 11

Dispositivo Android: Ser ou Ser reconhecido pela Google

Navicat Premium 11

Chegou a nova versão do Navicat Premium, software produzido pela PremiumSoft CyberTech Ltd., capaz de gerir as bases de dados locais e remotas em MySQL, Oracle, SQLite, PostfreSQL e MS-SQL Server num ambiente GUI (Graphical User Interface) de fácil compreensão que permite criar, organizar, aceder e compartilhar informação de forma fácil e segura, disponível para Windows, Linux e Mac OS X.

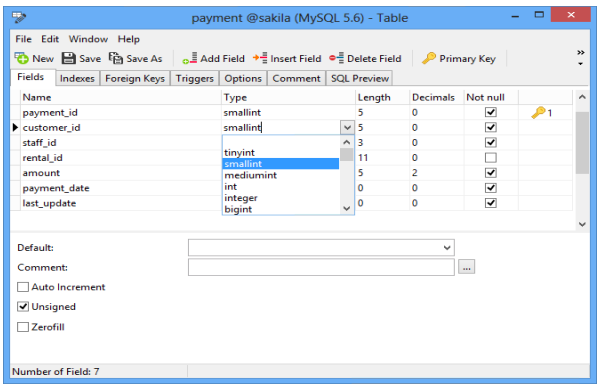
Vamos então conhecer as vantagens da utilização deste software bastante atrativo.

Conexão Segura

Estabelecendo uma sessão segura SSH através de SSH Tunneling, desfrutando de uma autenticação forte e comunicação segura criptografada entre dois hosts. O método de autenticação pode usar uma senha ou um par de chaves pública / privada. Também inclui HTTP Tunneling enquanto seus ISPs não permitem conexões diretas para os seus servidores de banco de dados, mas permite estabelecer conexões HTTP. HTTP Tunneling é um método para se conectar a um servidor que usa o mesmo protocolo (http://) e a mesma porta (porta 80), assim como um servidor web.

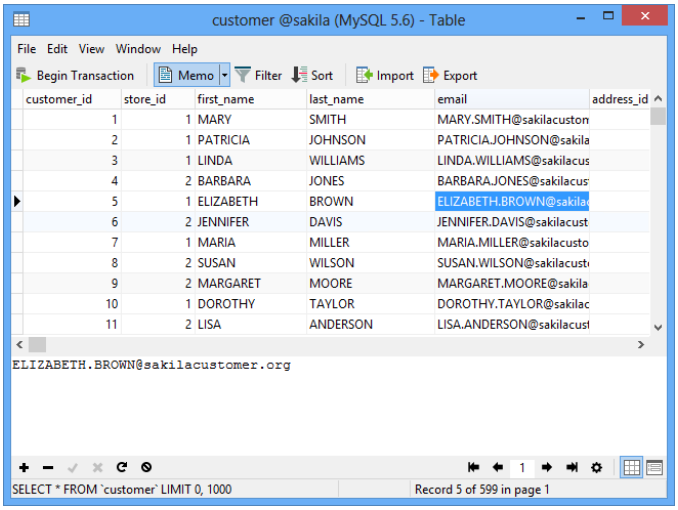
Desenho de Objetos

Podemos criar, modificar e criar objetos na base de dados. Esta função está disponível para todos os objetos da base de dados como tabelas, exibições, funções/procedimentos, índices, triggers e sequências, facilitando e tornando muito mais eficiente a sua criação pois não é necessário escrever o SQL completo para criar/editar os objetos , sabendo exatamente quais as opções que estamos a trabalhar.

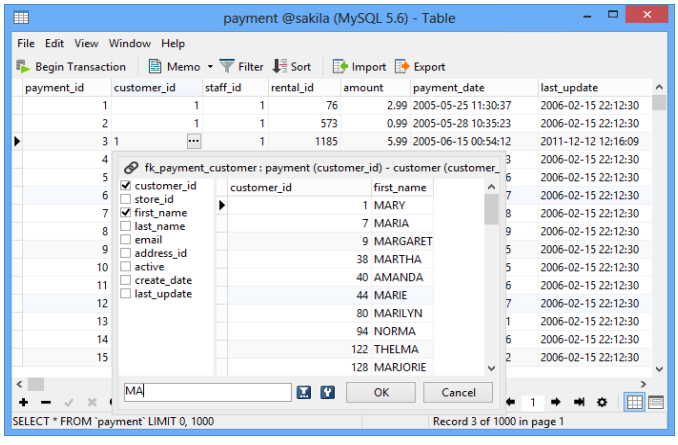


Vista de Tabela

Vista de grelha/formulário - Podemos adicionar, modificar e excluir registros usando exibição de grelha semelhante à folha de cálculo, em que cada coluna representa um campo e cada linha representa um registo, tornando a navegação fácil quando se tem milhares de registos usando as funcionalidades mostrar/esconder colunas na vista, ou utilizando os editores assistentes “memo”, “hex”, “image” e muitos mais. Além disso, é possível manipular o registro atual com vista de formulário. Ainda desfrutamos de uma visualização clara do nome do campo e seu valor com foco no registro atual.

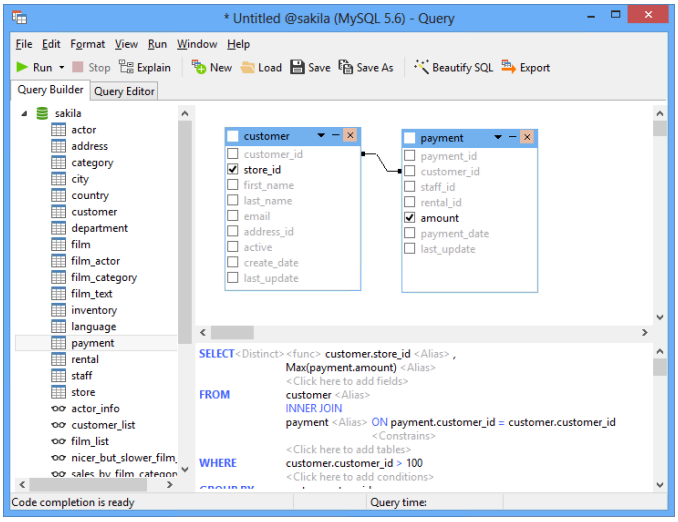


Seleção de dados com chave estrangeira - O Navicat usa as relações de chave estrangeira na sua base de dados para ligar automaticamente tabelas de pesquisa referenciadas por chaves estrangeiras e criar as listas pendentes em seu nome. Assim, podemos localizar os valores da chave estrangeira da tabela de referência facilmente. Assim não temos de mudar da vista da tabela principal para ver os valores disponíveis.

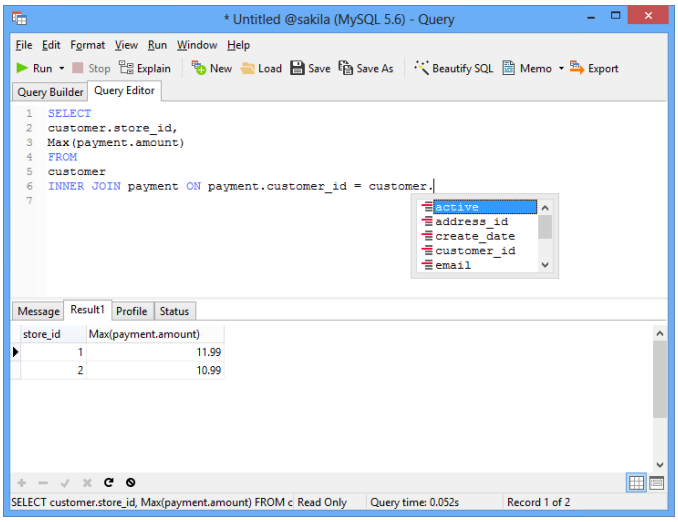


SQL Builder/Editor

SQL Builder - Permite a criação e edição de Queris/Views visualmente, sem ser necessário aprender a sintaxe e o uso adequado dos comandos. Está dividido em duas partes: na parte superior temos a visualização gráfica e na parte inferior a visualização da sintaxe, deixando-nos em aberto a forma como construímos tabelas, escrevendo as instruções SQL automaticamente. Podemos também escolher os parâmetros da query, adicionando valores as variáveis cada vez que corremos a instrução.

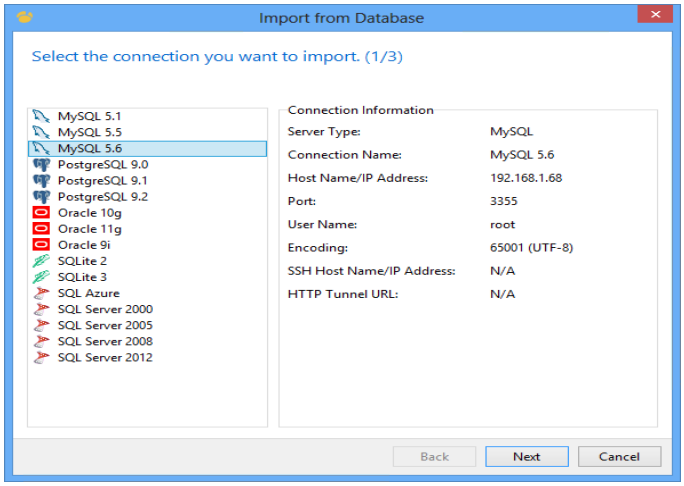


Conclusão de código - É uma maneira rápida da construção de instruções SQL no editor de SQL com recurso a conclusão do código. Temos também a opção de selecionar as propriedades disponíveis dos objetos da base de dados ou palavras-chaves SQL a partir da lista apresentada.

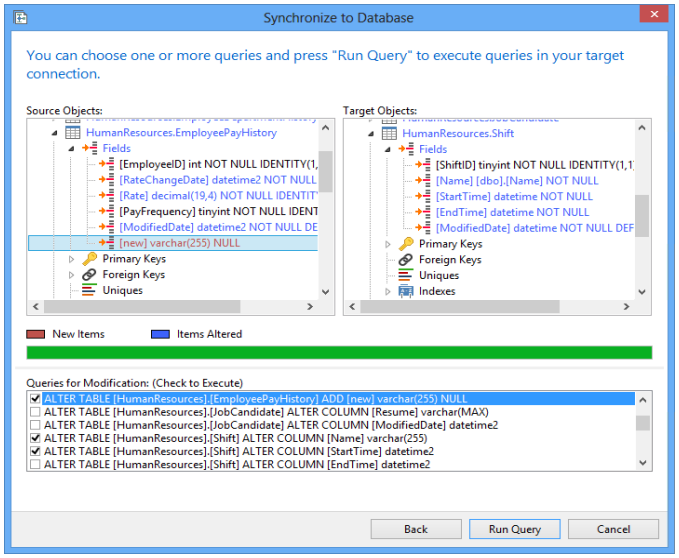


Design da Base de Dados

Engenharia Reversa - Permite criar um modelo de base de dados a partir dum existente, visualizando e editando graficamente a estrutura da base de dados. Também é possível criar o modelo graficamente e gerar uma base de dados que suporta o modelo criado.



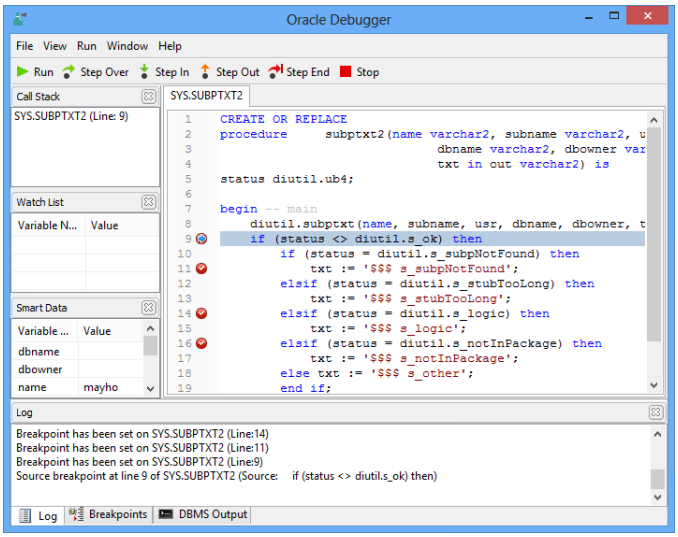
Forward engineering e criação de script - A sincronização da base de dados oferece uma visão completa de todas as diferenças das bases de dados. Uma vez que os dados são comparados, é possível ver as diferenças e gerar o script de sincronização necessário para atualizar a base de dados de destino e torná-lo idêntico ao seu modelo. Configurações de comparação e sincronização flexível permitirá a configuração de uma chave de comparação personalizada para comparação e para a sincronização. Além disso, o Export SQL dá-lhe a oportunidade de obter um controle total sobre o script SQL final, criando partes individuais do modelo, criação de regras de integridade referencial, comentários, conjunto de caracteres, entre outros, poupando várias horas de trabalho.



PL/SQL Debugger

PL / SQL Debugger oferece todos os tipos de recursos, tais como definir pontos de interrupção, percorrer o programa, visualizar e modificar os valores das variáveis e examinar a call stack. Podemos depurar o código PL / SQL, tais como

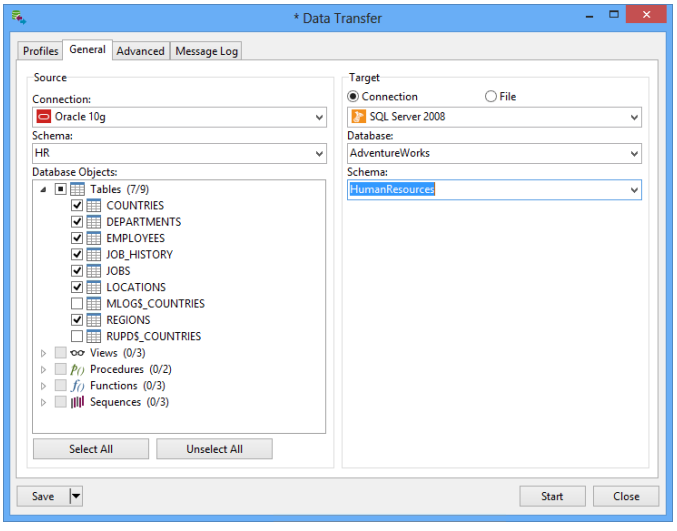
procedimentos e funções, métodos de objetos e triggers.



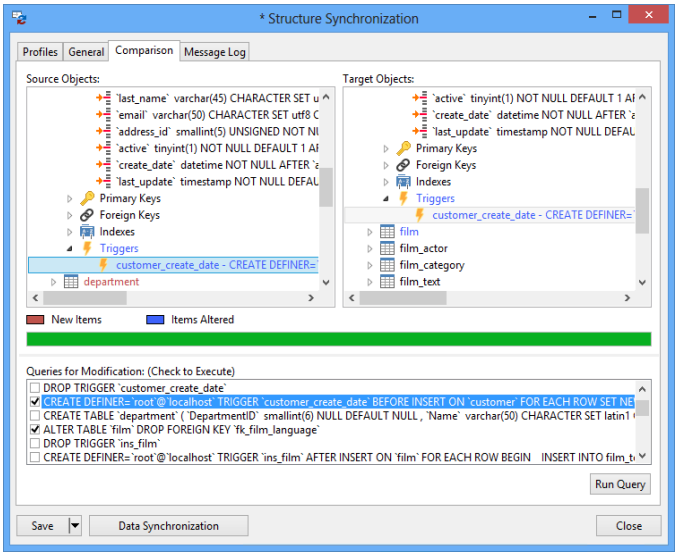
Ferramentas de Manipulação de Dados

Assistente de Importação/Exportação - Permite importar dados em uma tabela a partir de diversos formatos, tais como Access, Excel, XML, TXT, CSV, JSON e muito mais. Pode importar dados de ODBC após a criação de uma conexão de fonte de dados. Todas as tabelas disponíveis serão incluídas se a conexão for bem sucedida. Escolhemos as tabelas que desejamos importar ou especificar uma consulta usando o botão Add Query. Da mesma forma, podemos exportar dados em vários formatos como Excel, TXT, CSV, DBF, XML, etc.

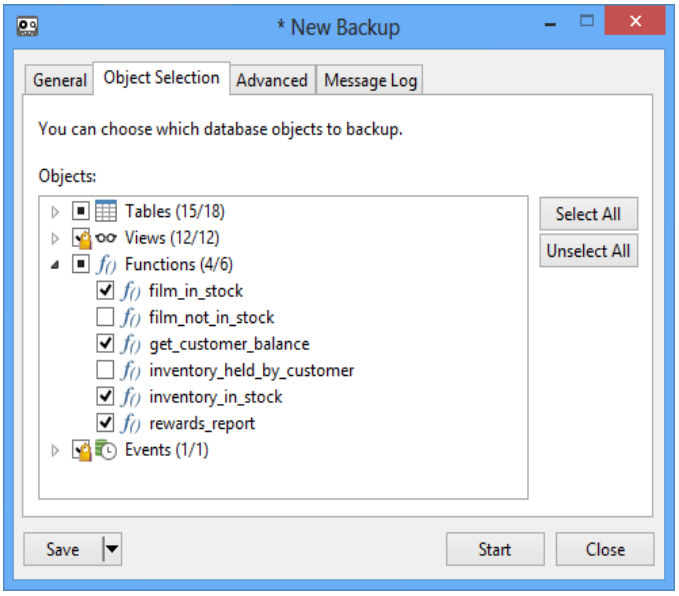
Transferência de Dados - Podemos transferir facilmente os dados entre várias base de dados: MySQL, SQL Server, Oracle, PostgreSQL e SQLite, simplificando bastante a migração de dados. Ainda temos a possibilidade de exportar os dados num ficheiro SQL com o formato e codificação desejada.



Sincronização de Dados/Estrutura - Temos a possibilidade de transferir dados a partir de uma base de dados para outra com o processo analítico detalhado. Da mesma forma, que podemos comparar e modificar as estruturas de tabelas. Para ambos, Dados e Estrutura, a base de dados de destino pode ser sincronizada no mesmo servidor ou noutro.

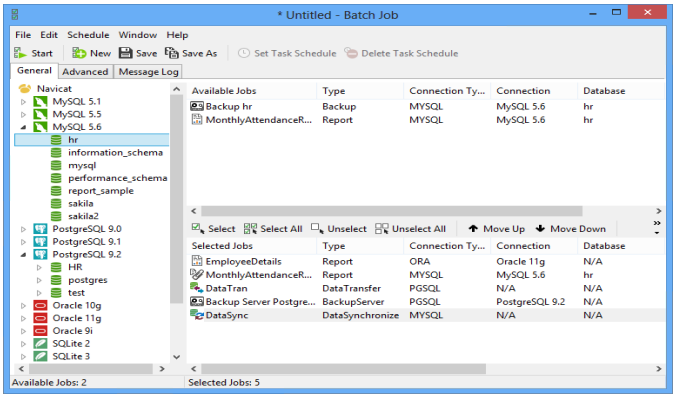


Cópia de Segurançça/Restauro - O Backup regular das bases de dados é um ponto importante. É possível fazer o backup ou restauro de todas as tabelas, registos e views nas bases de dados.



Tarefas Agendadas - Perfis de diferentes tipos de bases de dados podem ser criados num único trabalho de grupo, de modo a poder definir um cronograma a ser executado num momento específico. Lotes podem ser criados para o Relatório de Impressão, Backup, consulta, transferência de dados, sincronização de dados, Importação e Exportação. É possível gerar notificações por e-mails para destinatários específicos para garantir uma conclusão da tarefa com sucesso. Além disso, o arquivo exportado ou relatório impresso pode ser adicionado ao lote como anexo de email

para partilhar os dados facilmente.



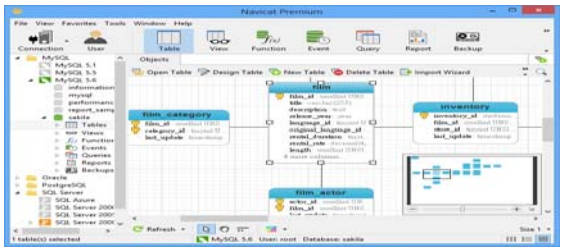
Construtor e Visualizador de Relatórios
(Disponível apenas para Windows)

Cria diferentes tipos de relatórios: Relatórios de projeto para Nota Fiscal, Estatística, etiquetas de endereço, entre outros em vários formatos de saída, como por exemplo, ficheiro de texto, PDF, Lotus, Excel, Gráfico, HTML e muitos mais.

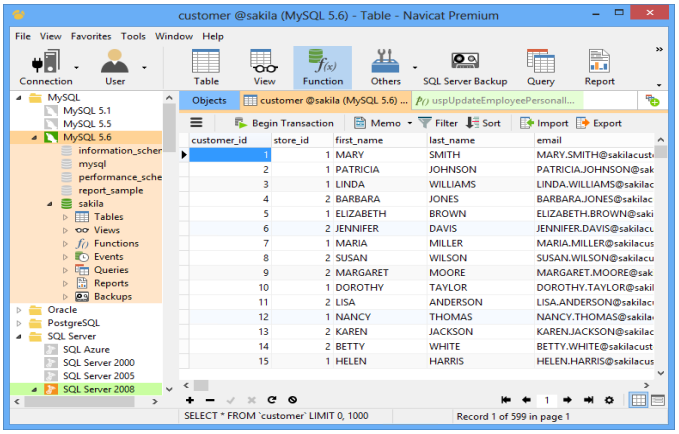
O Visualizador de Relatórios permite a navegação pelos relatórios gerados, permitindo a partilha dos mesmos com quem não tem o Navicat a correr no computador, mas tem o Visualizador de Relatórios Navicat instalado.

Ferramentas Uteis

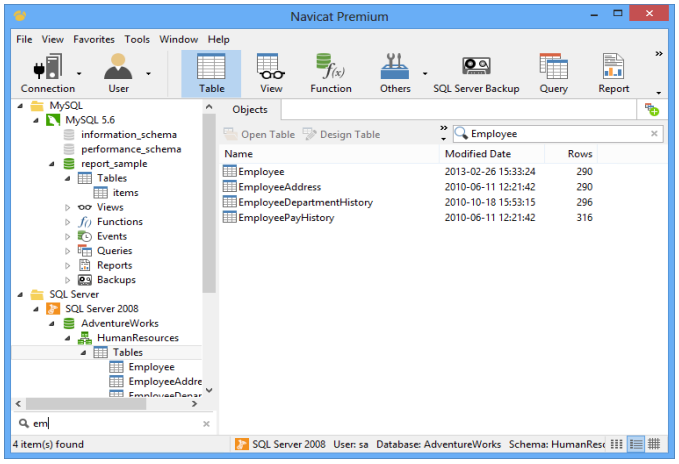
Diagrama ER - Auxilia a melhor compreensão do esquema da Base de Dados, mostrando a sua estrutura gráfica, sendo visíveis as relações entre as tabelas.



Coloração da Conexão/Agrupamento Virtual—Com recurso à coloração da conexão, sabemos imediatamente qual o servidor que está conectado. Inclui um indicador de faixa colorida mostrado na subjanela da barra de ferramentas para identificar conexões e seus objetos da base de dados. Podemos ter uma melhor organização sobre conexão e objetos por categorização em grupos com o recurso ao Agrupamento Virtual.



Filtro de objeto/arvore - É possível restringir a lista de objetos da base de dados exibidos, escrevendo uma sequência de caracteres e apenas mostra os objetos que começam com essa sequência.



Favoritos - Podemos voltar a um objeto da base de dados que visitamos frequentemente com fácil acesso, adicionando caminhos à lista de favoritos, tornando-os à distância de um clique, em vez de ter que navegar na conexão.

Resumo:
2,5 milhões de descargas que acumularam mais de 100 mil consumidores satisfeitos por mais de 138 países, produto premiado inúmeras vezes, utilizado por várias empresas de renome mundial, como é o caso da Apple Inc., fácil de usar e com uma interface bastante apelativa, melhorias continuas, vasto suporte, entre outros, são mais que motivos para darmos uma chance a este fantástico produto que nos facilita bastante a vida, no que refere a gestão de bases de dados e operações associadas.

AUTOR

Escrito por Nelson Belo Lima
Programador Especialista de Sistemas de Informação (Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Viana do Castelo), Técnico de Mecatrónica Automóvel e Mecânico Perito Citroen. Apaixonado pela tecnologia, conciliando o lazer com a vertente profissional, fui melhorando o meu conhecimento e aumentando experiência em vários pontos de interesse.

bit.ly/nelsonbelolima

Dispositivo Android: Ser ou Ser reconhecido pela Google

Nos dias que correm, em que as novas tecnologias praticamente dominam a nossa atenção, há um sistema operativo que ganhou bastante destaque nos últimos anos. Neste artigo vamos falar do Sistema Android.

Para os leitores mais distraídos, este sistema é baseado em Linux, tem como mascote o verdinho Bugdroid e o nome Android faz ainda este ano, 10 anos de existência. Em Outubro de 2003, era fundada a Android Inc, em Palo Alto.

Esta empresa tinha como meta desenvolver um sistema operativo para as máquinas fotográficas digitais, contudo, deixaram de lado a ideia inicial para desenvolverem um sistema operativo baseado em linux para a plataforma móvel para fazer frente, por exemplo, ao sistema Symbian.

Mais tarde em 2005, a Android Inc foi adquirida pela Google. Foi neste ano que se ouviram as primeiras especulações acerca do facto da Google estar a pensar lançar-se no mercado dos dispositivos móveis.

A primeira versão oficial da plataforma Android foi dada a conhecer ao mundo em 2007, sendo nessa altura também fundada a Open Handset Alliance (OHA), um grupo de empresas no qual estavam incluídas a Google, a Samsung, a Intel, a Motorola, a Texas Instruments, a LG, entre outras. Tinham em comum o mesmo objectivo, desenvolver software open source para dispositivos móveis. Desta parceria, cinco anos depois, o nome Android "renascia das cinzas", em Outubro de 2008, era disponibilizado ao público em geral, o primeiro dispositivo com o sistema Android, o HTC Dream.

A primeira versão (empresarial) foi divulgada em Novembro de 2007, sendo que em Outubro de 2008, este sistema foi disponibilizado ao público em geral.

Depois deste primeiro passo, que foi talvez o mais difícil, foi rápida a ascensão do sistema Android. Presente na maior parte dos smartphones e tablets que estão espalhados pelo mundo, são várias as versões que este sistema já teve. Um dado curioso é que todas as versões têm nomes de bolos ou sobremesas, seguindo sempre uma ordem alfabética. Sendo as últimas versões: Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean e ainda com estreia prevista este ano a versão Key Lime Pie.

Um outro facto curioso com que me deparei quando fiz a pesquisa para este artigo é que em todas as versões do sistema Android foram escondidos pequenos "Easter Eggs" (Ovo da Páscoa se traduzirmos à letra) para os utilizadores do Android. Ou seja, todos os dispositivos cujo sistema é Android têm uma pequena imagem escondida… Caso o leitor tenha um dispositivo Android e tenha curiosidade em saber qual a sua, basta ir a "Definições – Acerca do Telefone ", depois disto deve procurar a "Versão Android" e carregar sistematicamente na linha dessa informação. Após alguma insistência aparecerá então a imagem correspondente à sua versão Android.

Mas agora que já estamos familiarizados com a história deste sistema, vamos voltar ao propósito principal do nosso artigo.

Não sei se o leitor se irá identificar comigo, mas quando penso em sistema Android, tenho sempre o pensamento de o interligar com o Google, mais propriamente com o Google Play. Aquela aplicação maravilhosa que nos permite artilhar o

nosso dispositivo Android com tudo e mais alguma coisa. Podemos nem sequer precisar, mas o importante é experimentar e termos. Quantos de nós não estiveram já num café entre amigos em que surge o nome de uma aplicação ou de um jogo e temos sempre a tendência a fazer o

download da mesma para experimentar?

Com o Google Play tudo isto se torna muito simples. Mas então e o que acontece quando não temos o Google Play?

E porque não teríamos o Google Play? Podem perguntar alguns dos leitores mais distraídos… Ora isto pode muito bem acontecer caso o nosso dispositivo Android não seja reconhecido pelo Google.

Vamos regressar um pouco atrás… Como foi dito neste artigo anteriormente, o sistema Android é open source desde 2008, a Google disponibilizou todo o código sob a licença Apache. Ou seja, qualquer pessoa pode usar o sistema gratuitamente. Contudo, para que o dispositivo tenha acesso à Play Store da Google, é necessário uma licença da própria Google.

Para que o dispositivo possa ter um certificado válido, o mesmo tem que passar em alguns testes de software e de hardware. Ou seja, se quando comprarmos um dispositivo Android este não for reconhecido como um dispositivo oficial Android, o dispositivo não poderá ter pelas "vias mais simples" acesso à Play Store da Google.

Caso o leitor tenha curiosidade em ver quais os dispositivos reconhecidos, pode encontrar a listagem oficial aqui: https://support.google.com/googleplay/answer/1727131?hl=pt-BR

Para os dispositivos que não são reconhecidos pela Google há outras opções para conseguirmos fazer o download de algumas aplicações para o nosso dispositivo. Sendo as mais conhecidas as seguintes:

GetJar http://www.getjar.com/

AppBrain http://www.appbrain.com/

SlideME http://slideme.org/

Appsfire http://appsfire.com

Contudo, devo recordar ao leitor que apesar de haver imensas escolhas disponíveis nestes últimos exemplos, há sempre a possibilidade de que haja alguma aplicação de que possa apenas estar disponível no Google Play.

Cabe ao leitor, aquando da compra do seu dispositivo Android, pesar os prós e os contras de adquirir ou não um dispositivo reconhecido pela Google. No fundo o dispositivo em si, será sempre Android e funcionará na perfeição na mesma!

> ***"Com o Google Play tudo se torna muito simples. Mas então e o que acontece quando não temos o Google Play?"***

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/ProgramarED42_V

Veja também as edições anteriores da Revista PROGRAMAR

41ª Edição - Fevereiro 2013

40ª Edição - Fevereiro 2013

39ª Edição - Dezembro 2012

38ª Edição - Outubro 2012

37ª Edição - Agosto 2012

36ª Edição - Agosto 2012

e muito mais em ...

www.revista-programar.info

DUVIDAS?

IDEIAS?

AJUDAS?

PROJECTOS?

portugal-a-programar
.org