PROGRAMAR

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

EDIÇÃO #24 - JUNHO 2010 ISSN 1647-0710

INTRODUÇÃO A BASES DE DADOS PARA OBJECTOS

LUA 4ª PARTE : LINGUAGEM DE PROGRAMAÇÃO

PADRÕES DESENHO CORPORATIVOS

INTRODUÇÃO WINDOWS COMMUNICATION FOUNDATION

MODELO 3 ENTREVISTA A CELSO PINTO

índice

equipa PROGRAMAR

coordenadores
Fernando Martins
Joel Ramos
Pedro Abreu

editor
António Silva

capa
Sérgio Alves

redacção
Augusto Manzano
João Brandão
João Dias
Sérgio Lopes

equipa de revisão
Bruno Oliveira
Fernando Martins
José Oliveira
Liliana Baptista
Miguel Rentes
Sérgio Lopes

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

Os meus três anos...

É com imensa pena que me despeço de todos os leitores desta revista. Esta foi a última edição na qual participei como coordenador da Revista PROGRAMAR.

Foram 3 anos de experiências. Desde o convite inicial do Sérgio Santos - à altura, o coordenador - para editor, à passagem ara coordenador adjunto e depois a coordenação conjunta com o Pedro Abreu.

Foram 3 anos de experimentação. Lembro-me de experimentar vários modelos de coordenação. Desde o "cada um faz o que pode", passando pelo "é melhor alguém, quem tiver tempo, rever este artigo" até chegar ao actual modelo que temos uma equipa de revisão fixa e com um responsável definido e um artista convidado em cada edição, além dos autores que se mantém uma equipa dinâmica e de uma equipa de divulgação. Lembro-me ainda da experimentação de vários programas e das pequenas alterações que o design da revista foi sofrendo - embora nunca uma grande reestruturação.

Foram 3 anos de contactos. Desde o Sérgio Santos, o Miguel Pais e o Pedro Abreu (os dois últimos, tive o prazer de finalmente os conhecer pessoalmente o ano passado) à actual equipa formada pelo António Silva e pelo Fernando Martins (mantendo-se ainda o Pedro Abreu também), passando pelos vários designers das nossas capas, como o Daniel Correia, o José Fontainhas e o Sérgio Alves, entre outros, e ainda das dezenas de autores com os quais tive o prazer de contactar por email. Não esquecendo ainda os emails que recebemos quase semanalmente a elogiar a revista.

Foram ainda 3 anos de dores de cabeça. Desde que a revista saía com um dia de atraso até ao dia em que uma edição saiu quase com um mês de diferença até ao momento em que fomos obrigados a mudar a periodicidade. Ainda pelos autores que andei atrás para concluírem os seus artigos a tempo pois, ao contrário do que possa parecer, no fim de escritos os artigos ainda há muito a fazer na revista.

«Foram os meus 3 anos de Revista PROGRAMAR, nos quais atingi tanto o topo como o fundo.»

Foram 3 anos de discussões. "pode fazer-se isto, aquilo; isto pode ser melhorado; aquilo está mal"... Um infinidade de discussões, gravadas na minha memória e algures entre o Portugal-a-Programar, o Messenger, o GMail e ainda num local que não sei onde, mas nos meus backups que incluam logs do canal de IRC.

Foram ainda 3 anos de edições lançadas após a meia-noite. Um facto curioso, que é não me lembrar de momento de uma edição que tenha sido lançada antes dessa hora - parece que não mas há tanto que fazer nessa altura...

Foram os meus 3 anos de Revista PROGRAMAR, nos quais atingi tanto o topo como o fundo. Foram 3 anos com a magnifica equipa que cá continua e que se vai certificar que daqui a 3 anos eu ainda leia editoriais - de preferência melhor escritos que os meus - e que vai levar ainda mais longe e torná-la cada vez melhor.

Foram 3 anos. Acabam aqui oficialmente, embora tencione manter-me na retaguarda a ajudar quando puder, e quando a minha vida pessoal assim o permitir.

Foi um prazer.

Joel Ramos

Google tem novo sistema de indexação

http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html

A Google lançou um novo sistema de indexação, denominado Caffeine, e que segundo a empresa consegue uma eficácia 50% superior ao sistema anterior:

"Com o Caffeine, podemos analisar a Internet por partes e procurar ou actualizar o nosso índice de modo contínuo e global. À medida que encontramos novas páginas ou nova informação em páginas existentes, podemos adicioná-las imediatamente ao índice. Isso significa que quando procurar, vai estar a fazê-lo o mais próximo possível da versão mais recente da informação que pretende - independentemente de quando e onde foi publicada."

FCCN muda regras de atribuição de dominios

http://windows.microsoft.com/pt-PT/windows7/products/home

Entre as alterações destacam-se a descida de 50% dos nos custos anuais que as entidades especilaizadas no registo de endereços têm, e também a possiblidade de indivíduos e empresas venderem endereços que estão na sua posse.. No entanto ainda não foi desta que os endereços .pt foram liberalizados. Ao não liberalizar o registo de endereços, as autoridades portuguesas optam por uma posição preventiva, que evita os litígios causados pela usurpação de endereços por pessoas ou entidades que não são as legítimas proprietárias das marcas ou denominações comerciais, todavia na UE apenas Portugal e Malta ainda não liberarizaram o registo de endereços.

IE6 abaixo dos 5% de quota de mercado nos EUA

O browser, que até a própria Microsoft tem vindo a tentar "matar", baixou para 4,7% em Maio, acompanhado pelo irmão mais velho Internet Explorer 7 que também perdeu expressividade, para os 16,6%. Do conjunto de browsers da Microsoft, apenas o IE8 ganhou quota, passando para os 30,5%.

Ao mesmo tempo que o IE6 está a caminho da sepultura (um browser, recorde-se, de 2001 e cheio de falhas de segurança), o Google Chrome continua a crescer.
O Firefox perdeu quota de mercado, e está, agora, com 29,3%, enquanto que o Safari se manteve praticamente com os mesmo números, nos 9,1%.

Apesar de estas tendências se verificaram também a nível mundial, a verdade é que o envelhecido Internet Explorer 6 continua a ter muita expressividade em algumas zonas do globo, especialmente nos países em vias de desenvolvimento.

IOI 2010 - International Olympiad in Informatics

http://www.ioi2010.org/index.shtml
http://www.dcc.fc.up.pt/oni/2010/

A ser realizado este ano no Canadá, irá também receber a participação portuguesa, composta pelos 4 primeiros classificados na ONI (Olimpíadas Nacionais de Informática), e são eles: Rafael Schimassek, Rodrigo Gomes, David Ferreira e Francisco Huhn. Após a suas boas prestações na final nacional, temos a certeza que tudo farão para representar Portugal com orgulho, com brio e com vontade de trazer resultados de sucesso nas Olimpíadas Internacionais.

Não te esqueças, esta página pode ser tua!
http://www.revista-programar.info/front/yourpage

Introdução a Bases de Dados para Objectos

Quando falamos de bases de dados, os sistemas de Gestão de Bases de Dados Relacionais (MySQL, SQLite, Postegres, DB2, etc) são sem dúvida os sistemas que todos os programadores conhecem. Desde o seu aparecimento nos anos 70, e com o desenvolvimento da linguagem Structured Query Language (SQL), os sistemas relacionais conquistaram uma posição dominante no mundo da gestão de dados.

Criados em torno do principio simples de que os dados podem ser representados em pequenas entidades tabulares de 2 dimensões, compostas por linhas e colunas, que são depois relacionadas entre si através de chaves identificadoras, os sistemas relacionais mostraram, vezes sem conta, o seu valor. Este é um facto indiscutível.

Programação Orientada a Objectos (POO) é também, e por mérito próprio, um sistema largamente adoptado e com provas dadas no mercado de aplicações. Com uma evolução estável e utilizado por uma fatia significativa de programadores, disponível através de uma vasta selecção de linguagens de programação, com ferramentas, métodos de desenvolvimento e os mais diversos acessórios para a sua correcta utilização, POO é um paradigma que veio para ficar e triunfar. Este é um facto indiscutível.

Com os dois factos anteriores, chegamos ao ponto onde é comum a união das duas áreas, de um lado o poder das bases de dados relacionais e de outro a força da programação orientada a objectos. E talvez esta afirmação nos faça roçar o grande problema: ligar um sistema desenvolvido com base em objectos a um sistema que vê todos os dados como seres bi-dimensionais exige um esforço de tal forma significativo que, efectivamente, existe um fosso entre os dois paradigmas.

Consideremos o desenvolvimento típico de uma aplicação usando tecnologias que implementem os dois paradigmas:

O arquitecto do sistema irá pegar nos requisitos que os clientes indicaram, através de uma qualquer método, possivelmente UML, vai transformar esses requisitos num modelo de objectos que não só é um monumento à utilização de herança, polimorfismo e os demais conceitos de POO, como ainda torna os requisitos do cliente algo cativante de implementar. Munido desta obra prima, o arquitecto passa o fruto do seu trabalho ao administrador da base de dados que o irá dissecar, transformar num diagrama Entidade-Relacionamento, ou outro qualquer que lhe seja familiar, rever relações, criar tabelas e configurar um servidor que, em muitos casos, é uma peça de tecnologia tão afinada e sensível como um relógio suíço.

Findo estas duas etapas, o arquitecto e o administrador reúnem-se para perceber como é que dois modelos, tão diferentes podem comunicar como é esperado e dessa reunião nasce, tipicamente, uma classe para cada tabela, um conjunto de quatro procedimentos base por cada tabela (o comum CRUD: create, retrieve, update e delete) e uma imensa framework de código para gerir tudo isto.

Mesmo usando ferramentas de apoio, como as tecnologias de Object/Relational Mapping (ORM), todo este processo é moroso e complexo, e ainda não começámos a implementar o produto que o cliente pediu.

Para tornar o processo mais simples, um dos lados poderia ser alterado. Ou alteramos a linguagem de programação ou alteramos a base de dados. Se pensarmos em alterar a linguagem facilmente chegamos à conclusão que, qualquer que seja a linguagem, nunca será perfeitamente compatível com a visão tabular das bases de dados relacionais. Por outro lado, remover a base de dados poderá ser uma boa opção: e se, de alguma forma, pudéssemos ter uma base de dados que aceitasse os objectos que tão úteis nos são e não precisasse de os transformar?

Linguagem + Persistência

Sistemas de Bases de Dados para Objectos (OODBMS) são sistemas nos quais a representação da informação é feita através de objectos, tal como em POO, e aos quais é adicionada a facilidade de acesso por linguagens Orientadas a Objectos (OO) através da adição de capacidades de POO. Estes sistemas estão sempre associados a uma linguagem POO existente (ex: Java ou C#), fazendo uso da mesma representação interna de um objecto, e guardam directamente os objectos que os programadores usam no desenvolvimento. Embora ligadas a uma linguagem específica, muitos destes sistemas possuem mecanismos de conversão de modo a que uma aplicação feita numa linguagem possa ser convertida para outra linguagem diferente mantendo todos os registos existentes na base de dado.

Como tantas outras tecnologias, OODBMS surgem da investigação académica em meados dos anos 80, e progridem até à actualidade de forma algo atribulada. Sem grande adopção pelo mercado, vêm ser criada uma organização que pretendia desenvolver um standard que

promovesse a sua utilização. Essa organização, Object Data Management Group (ODMG), veio a ser substituída pela actual Object Management Group (OMG) que se torna assim na organização responsável por desenvolver e promover um standard para a tecnologia de bases de dados para objectos.

Neste momento, o OMG, pretende apresentar um standard que ajude o mercado de bases de dados para objectos, mas embora um standard definitivo e oficialmente adoptado ainda não exista, do esforço surgiram várias tecnologias actualmente em uso, tais como as Queries Nativas usadas em vários motores de bases de dados para objectos, ou sistema LINQ da plataforma .Net.

Vantagens

Bases de dados orientadas a objectos são ideais em conjunto com linguagem OO porque eliminam qualquer processo de conversão para a persistência dos dados. O modelo que o arquitecto do sistema desenvolveu será exactamente o mesmo que será usado na base de dados, as relações, os objectos, os atributos, tudo se manterá igual e para os programadores não há mais a necessidade de decorar estruturas de tabelas ou procedimentos de acesso, basta enviar os objectos para o motor de bases de dados e rever os mesmos objectos quando necessário se aplicar qualquer conversão. Isto acontece porque o motor usa exactamente o mesmo modelo de dados que a linguagem de programação.

Com o uso destas bases de dados ultrapassamos vários problemas:

1. Relações e objectos complexos. É possível guardar vários objectos relacionados por herança ou composição e com vários níveis de complexidade sem qualquer intervenção do programador;

2. Não há duas linguagens (uma para a BD e uma para a aplicação). Há apenas uma linguagem comum que funciona em todo o projecto. Menos linguagens traduzem-se em melhor desenvolvimento, facilidade de depuração e de eliminação de erros, necessidade de menores conhecimentos que se podem transformar em custos menores, etc.;

3. Remoção do problema de Impedence Mismatch. O tempo perdido a mapear objectos para tabelas e os problemas quando um objecto não é relacionado directamente para uma tabela (ver ponto 1), é completamente eliminado. Esta característica ajuda na performance final da aplicação;

4. Existe apenas um modelo de dados. Como podemos ver na introdução, dois modelos de dados distintos oferecem muitos problemas que aumentam a complexidade do projecto ou os seus custos;

5. Não há necessidade de identificadores únicos para cada objecto, como existe para cada registo de uma tabela através das chaves primárias, toda a identificação dos objectos é transparente ao programador.

OODBMS são também sistemas excelentes para guardar informação com relações e/ou representações complexas, que irão tirar todo o partido da linguagem OO e consequentemente da facilidade de utilização do motor bem como das optimizações que os motores contêm possibilitando um nível de performance superior ao dos RDBMS, tida por alguns autores como 10 a 1000 vezes superior.

Esta performance pode ser explicada, principalmente, por dois pontos:

- A falta de Impedence Mismatch, como mencionado acima;
- Optimização para Traversal. Este é o processo pelo qual se percorre o grafo que representa as relações dos nossos objectos, indo de nó a nó para obter os dados.

Na secção da bibliografia são apresentados recursos onde o leitor pode aprofundar o tipo de benchmarks e os resultados comparativos entre OODBMS e RDBMS, e do acesso nativo que os OODBMS oferecem em detrimento de acesso com ORMs e outras técnicas para RDBMS. No entanto, qualquer benchmark feito é dependente da tarefa a testar e se em alguns pontos os OODBMS podem mostrar ganhos significativos, noutros as diferenças serão negligenciáveis ou até demonstrar perdas.

Olhando para a globalidade dos valores, podemos aceitar que os OODBMS oferecem boas perspectivas de performance quando comparados com outros métodos de acesso a RDBMS e como regra geral devemos ter em atenção o tipo de dados e se estamos ou não a trabalhar com objectos. Quanto mais simples for o tipo de dados, maiores poderão ser as vantagens em usar um RDBMS e SQL, mas à medida que a complexidade aumenta, a velocidade obtida com OODBMS é vantajosa. Se trabalharmos com objectos não deveremos descartar os OODBMS.

Código/Exemplo

Vamos desenvolver uma pequena aplicação de gestão de contactos usando um motor de bases de dados para objectos livre, disponibilizado sob a licença GPL, chamado DB4O.

Descrição: Pretendemos uma aplicação desktop para gerir contactos pessoais. Deverá ser possível guardar nomes, números de telefone, moradas e datas de nascimento dos contactos. Precisamos de fazer algumas pesquisas pelo que

é valorizado a existência de um sistema simples que permita encontrar os contactos. Deverá ser possível guardar a lista de contactos para uso futuro. (O código completo estará disponível para download em: http://wiki.sergio-lopes.org/index.php?n=Projects.AgendaDB4O)

Com a descrição anterior criamos duas classes: a nossa classe principal, Agenda, responsável por fornecer todos os métodos de uma agenda digital, e a classe que representa os nossos contactos, Contacto.

Transformando em código:

```
public class Contacto {

    private long id;
    private String nome;
    private String apelido;
    private String dataNascimento;
    private String telefone;
    private String telemovel;
    private String endereco;
    private String codigoPostal;
    private String localidade;

    public Contacto() {
        //DO NOTHING
    }

    public Contacto(String nome,
String apelido, String dataNascimento,
        String telefone, String
telemovel, String endereco,
        String codigoPostal, String
localidade) {

        this.nome = nome;
        this.apelido = apelido;
        this.dataNascimento =
dataNascimento;
        this.telefone = telefone;
        this.telemovel = telemovel;
        this.endereco = endereco;
        this.codigoPostal =
codigoPostal;
        this.localidade = localidade;

        id = hashCode();
    }

    //Restantes getters, setters,
equals e hashCode
    (...)
}
```

A classe que nos interessa e que regista os dados é a classe Agenda. O nosso motor está acessível através de uma instância de ObjectContainer. Esta classe oferece os métodos base para guardar, remover e pesquisar objecto guardados e é o nosso ponto de ligação com o motor. Como podem reparar é um objecto Java, não um método para injectar SQL a ser executado num servidor.

```
public class Agenda {
    private ObjectContainer db; //<--
Motor de bases de dados

    public Agenda(String ficheiro) {
        abrirDb(ficheiro);
    }

    public void fechar() {
        if (db != null) {
            db.close();
        }
    }
//...
```

Neste exemplo estamos a usar uma base de dados representada por um ficheiro. Seria possível usar um servidor remoto e aceder por rede, no entanto, para simplificar o código e porque é apenas uma apresentação, será usado o acesso embutido e um ficheiro local.

Estamos também a usar o motor como parte da nossa aplicação, ele executará enquanto a aplicação estiver aberta e terminará quando a aplicação fechar. Todos os parâmetros de configuração são os de omissão. Se o ficheiro não existir será criado, se existir serão usados os dados já guardados.

Ao abrirmos a base de dados obtemos a instância de ObjectContainer que precisamos para trabalhar os nossos dados.

```
private void abrirDb(String ficheiro)
{
    db =
Db4oEmbedded.openFile(Db4oEmbedded.newC
onfiguration(), ficheiro);
}
```

E chegamos ao primeiro método que nos dará controlo sobre os dados, o método **store** permite guardar ou actualizar o nosso objecto. Se o objecto é novo então motor vai inserir o objecto, se é um objecto que já existe na base de dados e foi modificado, então o motor vai actualizar os dados na base de dados. A par com o método **store**, temos também o método **delete**, que como seria de esperar permite remover o objecto que é passado como parâmetro.

```
public void adicionarContacto(Contacto
contacto) {
    db.store(contacto);
}

public void removerContacto(Contacto
contacto) {
    db.delete(contacto);
}

public void
actualizarContacto(Contacto contacto)
{
    db.store(contacto);
}
```

Estes métodos mostram uma vantagem no uso deste tipo de motores de bases de dados. Neste caso o programador não precisa de saber SQL ou outra linguagem especial para manipulação de dados, lembrem-se: o motor está a guardar os nosso objectos Java e está ligado à nossa linguagem de programação Java.

Podemos usar directamente a linguagem para a pesquisa de dados e dentro do método de pesquisa podemos colocar a lógica que precisarmos para determinar se o objecto que nos é mostrado é o que queremos ou não.

O motor vai executar o método match para os objectos existentes e se o método devolver true coloca o objecto em questão numa lista que nos é devolvida no fim. Essa lista contém todos os objectos que o algoritmo de pesquisa identificou.

Este tipo de pesquisas é chamado de native queries, ou pesquisas nativas, porque faz uso directo da linguagem de programação que estamos a usar. Existem outros métodos que veremos mais a diante.

```
public List<Contacto> pesquisar(final
String termo) {
    return db.query(new
Predicate<Contacto>() {

        public boolean match(Contacto
contacto) {
            if
(contacto.getApelido().contains(termo)
|| contacto.getNome().contains(termo)
||contacto.getLocalidade().contains(ter
mo)
||contacto.getEndereco().contains(termo
)
||contacto.getNome().contains(termo)
||contacto.getTelefone().equals(termo)
||contacto.getTelemovel().equals(termo)
) {
                return true;
            }

            return false;
        }
    });
}
```

Além das pesquisas nativas existem também as queries by example, ou pesquisas por comparação, e pesquisas especiais que fazem uso da definição da classe para obter resultados. As pesquisas por comparação recebem um objecto do mesmo tipo do que pretendemos pesquisar, com os campos que queremos usar na comparação preenchidos, e devolve objectos que sejam similares. Por exemplo, se quiséssemos procurar todos os contactos com o nome "João", seria necessário cria um objecto do tipo Contacto, colocar o nome que pretendemos pesquisar e fornecer esse objecto ao motor. Depois de terminada a pesquisa iríamos receber uma lista com todos os contactos que tinham "João" no atributo nome.

```
public List<Contacto> queryByExample()
{
    //Numa situação real receberíamos
o objecto como parâmetro do método
    Contacto exemplo = new Contacto();
    exemplo.setNome("João");

    return db.queryByExample(exemplo);
}
```

As pesquisas especiais poderão ser dependentes do motor de bases de dados que estamos a usar, neste caso o DB4O oferece um mecanismo que nos permite obter todos os objectos de uma determinada classe. O método seguinte devolve todos os objectos do tipo Contacto que estejam guardados na bases de dados.

```
public List<Contacto>
listarTodosContactos() {
    return db.query(Contacto.class);
}
```

E com este exemplo simples conseguimos criar todo o sistema de persistência, tudo isto com dois métodos e um ou outro mecanismo de pesquisa. Usámos um método para guardar e actualizar dados, store, um para remover, delete, uma pesquisa nativa que faz uso da linguagem Java, e um método de pesquisa especial por classe. O restante código, que pode ser visto nos ficheiros do artigo, não é mais cobertura de açúcar para criar uma interface e oferecer alguns botões com ícones agradáveis.

Problemas

Mas... se isto é tão bom porque é que não é tão usado no mercado de aplicações?

Esta é uma pergunta difícil de responder. Naturalmente não existem sistemas perfeitos, e as bases de dados para objectos possuem algumas desvantagens. Se olharmos para o factor humano, podemos apontar alguns pontos que, subjectivos em natureza, afectam a escolha da tecnologia:

- Falta de conhecimento. Como foi dito no início do artigo, quase todos os programadores sabem o que são SGBDs, fazem parte das cadeiras base dos cursos de informática, quer na área geral quer na área de programação. O mesmo não acontece com OODBMs;
- Inercia e comodismo. É mais simples usar o que já conhecemos;
- Medo da tecnologia. O desconhecimento da tecnologia leva a que se tenha receio da sua utilização.
- Medo empresarial. Muitas das empresas que desenvolvem bases de dados para objectos são pequenas e com pouca expressão ou publicidade no mercado, e isso torna provoca algum medo quando é necessário decidir o que usar. O risco de apostar numa empresa pequena parece maior do que o risco de apostar, por exemplo, na Oracle.

Nota: Estes pontos podem também ser vistos como não tendo efeito ou até como vantagens uma vez que as pessoas envolvidas respondem de modo diferente a cada situação apresentada.

Se nos focarmos nos factores técnicos deparamo-nos com:

- Falta de interoperabilidade. Em parte devido à fraca adesão e à falta de um standard competente (ver também ponto seguinte);
- Forte ligação à linguagem de programação associada. Implica que os dados presentes num OODBMS só possam ser acedidos de uma linguagem de programação específica. Este problema pode ser eliminado em alguns motores que oferecem ferramentas para conversão do modelo de dados para outras linguagens;
- Falta de capacidade para pesquisas Ad-hoc. Em RDBMS é possível criar pesquisas que dão origem a novas tabelas através da junção de tabelas existentes (utilização de JOINs, implícitos ou explícitos e criação de tabelas temporárias). Este processo não é possível em OODBMS, não é possível juntar dois objectos de modo a que a sua combinação origine um novo objecto.

Resumindo

Este artigo teve como objectivo despertar interesse nos sistemas de bases de dados para objectos e demonstrar um exemplo prático da sua utilização. Não se pretende que o leitor considere o motor usado no exemplo como a única opção ou como a melhor opção. Existem muitos outros motores, para as mais variadas linguagens.

Como qualquer outra tecnologia, esta oferece vantagens e desvantagens que devem, sempre, ser avaliadas à luz do objectivo do projecto. Os resultados de testes de performance mostram que a tecnologia tem um bom desempenho face às alternativas existentes e que deve ser considerada, especialmente quando lidamos com objectos e linguagens POO.

Como objectivo principal, esperamos que o leitor tenha ficado sensibilizado para a existência da tecnologia de bases de dados para objectos e que a considere aquando da avaliação de tecnologias para os projectos que desenvolve.

Recursos consultados

- Barry & Associates. Consultado a 12/04/2010. http://www.service-architecture.com/index.html
- Obasanjo, Dare. Consultado a 12/04/2010. http://www.25hoursaday.com/WhyArentYouUsingAnOODBMS.html
- Stanford Linear Accelerator. 12/04/2010. http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/index.shtml
- Wikipedia. Consultado a 12/04/2010. http://en.wikipedia.org/wiki/Object_database
- Kratville, William C. Paper Selecting an RDBMS or OODBMS For use in a Distributed Voice Processing System. Webster University. 2, Julho 2000
- Benchmark db4o Open Source Object Database. http://www.db4o.com/s/benchmarkdb.aspx
- Kopteff, Mikael. The Usage and Performance of Object Databases compared with ORM tools in a Java environment. Haaga-Helia University of Applied Sciences and Floobs Ltd.

SOBRE O AUTOR

Formado no curso de Engª Informática pela Escola Superior de Tecnologia e Gestão de Leiria, é actualmente formador e programador freelancer e empreendedor, especializado no desenvolvimento de aplicações WEB, desenvolvimento para Desktop na plataforma Java e na área de usabilidade. Tem uma paixão especial por tudo o que esteja relacionado com a tecnologia Java e é um forte apoiante de software livre. Colabora com vários projectos de software livre no papel de programador, de tradutor ou de qualquer tarefa onde possa ser útil.

Sérgio Lopes

Padrões de Desenho Para Projectos Corporativos

Trabalhando em Camadas

A divisão da aplicação em camadas é uma técnica muito utilizada e difundida pelos projectistas de software e que usualmente pode ser útil para projectos de médio a grande porte. A pensar na sua utilidade podemos destacar:

• Facilidade de implementação: Podemos escrever um serviço que consumirá dados oriundos de uma base de dados qualquer sem conhecer necessariamente a sua estrutura, assim como, podemos modificar a base de dados sem que o nosso serviço seja alterado;

• Reutilização de código: Uma vez construído o nosso serviço, o mesmo poderá ser usado em alto nível pelos clientes que o consumirão em outras aplicações;

Esses factores são relevantes mas não podemos esquecer que há também aspectos negativos na utilização de camadas como:

• Alterações em cascata: As camadas encapsulam muito do desenvolvimento, mas por vezes uma simples alteração implica alterações em toda a estrutura acoplada. Para o caso de uma adição de uma coluna no banco de dados a camada lógica seria alterada, o nosso serviço também seria se o mesmo actuasse como "Objecto Valor" para persistência de informações e por conseguinte os clientes consumidores também o seriam;

• Desempenho: Abstracção para uso em camadas tem um custo no factor desempenho, que muitas vezes é crucial para determinadas áreas do projecto;

Para atenuar os pontos-contra desta técnica convém:

• Usar Baixo Acoplamento: Já que as camadas estarão aninhadas, melhor que seja somente pelos seus dados referentes, camadas acopladas em alto nível de dependência directa deverão ter suas dependências exportadas para outros projectos, dificultando assim a reutilização de código e alterações em cascata de mais do que se deve alterar;

• Não ser um projectista purista: O modelo de projecto divido em camadas é bom, mas não é uma verdade absoluta. Negócios requerem desempenho e em áreas onde a curva de desempenho cai é necessário buscar estruturas menos custosas, um Proxy (Proxy GOF) é uma boa saída se você precisa de instâncias de memórias constantes de áreas críticas, contudo, na maioria das vezes é necessário buscar estruturas de dados mais rápidas, e envolvê-las em um Adaptador (Adapter GOF);

Cabe salientar que padrões, quaisquer que sejam, demonstram imperfeição no paradigma que é utilizado, ou que o mesmo não é completo, o que por si só gera um conceito mais abrangente que foge do escopo deste artigo.

A estrutura mínima de funcionamento

A estrutura mínima de funcionamento corresponde às seguintes camadas:

• Apresentação: Camada onde os dados são visualizados e alterados pelo usuário, nessa camada fica toda a experiência do usuário com o que ele tem como sendo o software;

• Domínio: Modelo de negócio, aqui é o local ideal para mantermos representações lógicas do nosso modelo de dados e todo comportamento inerente a ele;

• Modelo de Dados: Aqui está nosso repositório de dados, geralmente representação física de um banco de dados;

Convém alertar para não confundir estas camadas base com o difundido MVC, (Model, View, Controller). Uma estrutura MVC, se utilizada, estaria entre o Domínio (que corresponde ao Model) e a Apresentação (correspondente a View) tendo o Controle da Aplicação (Controller) entre elas;

É plausível que estas camadas podem ser estendidas de acordo com a necessidade sumária do projecto, mas devemos ter cuidado para não ferir a utilidade do modelo que estamos criando. Um bom medidor para tal seria analisar uma interface de apresentação totalmente imprópria para o modelo original, como uma aplicação de linha de comando para um comércio electrónico. Se ainda assim essa implementação parecer viável no modelo actual tem um provável indicador que a sua arquitectura vai bem.

Organização do Domínio:

Então, como organizar o modelo de dados? Há algumas maneiras das quais se destacam:

• Modelo Procedimental: De fácil implementação pela sua simplicidade, funciona bem quando se tem como acesso uma estrutura fazendo o papel de gateway por cada tupla ou em estruturas desconectadas multi valoradas. O aspecto negativo é que é fácil ter código duplicado a partir do momento que o cenário passa a requerer lógica mais complexa. Essa duplicidade pode ser revista, mas ainda assim é fácil ter código virulento;

• Modelo de Domínio: A "mudança de paradigma" que os projectistas que saltaram para o mundo OO tanto falam parece nascer aqui. Em um modelo de domínio para um sistema de comércio electrónico, por exemplo, seria pertinente ter classes para clientes, produtos, tributações, arredondamentos, internacionalização, etc. Esta abordagem super valora o modelo, contudo, acentua a curva de aprendizagem. É comum ver projectistas trabalhando meses em projectos pequenos ou médios para habituar-se a tal modelo. Outra dificuldade inerente desse modelo é a persistência do seu modelo de domínio. Um modelo de domínio rico tende a ser bem diferente do modelo relacional disposto na sua camada de dados, o que implica em complexidade na persistência com gateways de dados e a provável necessidade de uso de uma camada adicional para realizar o mapeamento objecto relacional;

• Módulo tabela desconectado: Este é um intermédio entre as duas maneiras. Organizar os dados em estruturas desconectadas que representem tabelas alivia a repetição desnecessária de escrita das operações CRUD, além do que, possibilita o uso dos conceitos OO neste mesmo modelo. Outro factor implicante é que a maioria dos ambientes corporativos reluta em adoptar uma camada de dados orientada a objectos devido a instabilidade e relativa imaturidade dos produtos comerciais para tal fim existentes, comparando os mesmos a bancos de dados relacionais. Além disso várias ferramentas comerciais tendem a utilizar conceitos de objectos e estruturas desconectadas, o que, facilita a utilização deste modelo com tais ferramentas. Porém esse modelo só tem tais facilidades de uso em ambientes projectados para isso, com frameworks e bibliotecas para trabalhar com tais estruturas;

Cada modelo tem seus aspectos que o favorecem ou não. Cabe a nós projectistas perante o problema, a experiência da equipa, os factores de tempo, recursos e retorno provável sobre o investimento, avaliar e escolher a melhor forma de organizar a lógica de um projecto organizado em camadas.

Não ser extremista nas suas concepções, ouvir e valorizar as ideias de cada um da sua equipa, estudar e contar com erros e acertos do passado, o ajudará a não tropeçar no alvoroço que entorna a arquitectura de software e ter projectos realmente escaláveis utilizando com sapiência cada recurso tecnológico disponível.

SOBRE O AUTOR

João Dias de Carvalho Neto, Bacharelando em Ciência da Computação, apaixonado por arquitectura, Padrões de Desenho Corporativos, Gerência de equipas e projectos. Trabalha como Analista de Aplicações em Pernambuco – Brasil desde 2006.

João Dias

LUA - Linguagem de Programação - Parte 4

No artigo anterior desta série (terceiro artigo) foram apresentadas informações sobre a utilização dos recursos de laços de repetição. Desta forma, foram abordados os laços de repetição while, repeat e for. O tema central deste artigo relaciona-se ao uso de variáveis indexadas, além de abordar os temas sobre concatenações e precedências de operadores.

Concatenação

A linguagem Lua efectua operações de concatenação através do operador "‥" (ponto, ponto) que foi indicado no primeiro artigo desta série.

As operações de concatenação são feitas exclusivamente com dados do tipo carácter na formação de cadeias (strings). Se algum valor numérico for utilizado, a linguagem Lua tratará automaticamente este valor como sendo um carácter.

O programa seguinte demonstra o uso do efeito de operações de concatenação:

```
-- inicio programa CONCATENA

  print("Linguagem " .. "Lua")
  print(1 .. 2)
  print("Lua " .. 2010)

-- fim programa CONCATENA
```

Escreva o código do programa num editor de texto, gravando-o em seguida com o nome concatena.lua e execute-o com a linha de comando lua 5.1 concatena.lua..

O programa fará a apresentação das cadeias: Programação Lua, 12 e Lua 2010 em três linhas de texto distintas.

Precedências

Nos artigos já publicados foram mostrados os conjuntos de operadores usados em Lua, que são:

- Operadores aritméticos;
- Operadores relacionais;
- Operadores lógicos.

Cabe ressaltar a ordem de precedência de operadores que a linguagem Lua usa, sendo:

- ^ (exponencial)
- not (negação) - (unário negativo)
- (multiplicação) / (divisão)
- + (adição) – (subtracção)
- .. (concatenação)
- < > <= >= ~ = == (relacionais)
- and (conjunção)
- or (disjunção)

A alteração de precedência pode ser feita com o uso dos símbolos de parênteses.

Variáveis Indexadas

No primeiro artigo desta série foi abordado o conceito sobre variáveis na forma mais ampla. Na ocasião foi comentado que uma variável caracteriza-se por ser uma região de memória que tem por finalidade armazenar um determinado valor por um determinado espaço de tempo. E que uma variável pode assumir dois comportamentos operacionais em um programa: o comportamento de acção e o comportamento de controlo.

Já foi também comentado o facto de que uma variável armazenar apenas um valor cada vez, o que é restritivo, sendo esta a característica de uma variável simples. A restrição operacional de uma variável simples pode ser atenuada com o uso de variáveis indexadas.

As variáveis indexadas são conhecidas por outras formas de referência, como: vectores, arranjos, tabelas, listas, variáveis subscritas, matrizes, variáveis compostas, variáveis dimensionadas, entre outras denominações encontradas.

De forma sucinta, uma variável indexada é a definição de uma região de memória com um só nome de identificação, que possibilita o armazenamento de mais de um valor numa determinada região de memória.

O funcionamento de uma variável indexada ocorre com a definição do tamanho da dimensão que esta variável

possuirá.

Na linguagem de programação Lua o uso de variáveis indexadas ocorre por meio do uso de vectores associativos, onde o vector pode ser indexado por valores numéricos, por cadeias de caracteres ou por qualquer valor dos demais tipos suportados pela linguagem. O único valor que não se pode indexar em um vector é o valor nil.

Matrizes Unidimensionais

Uma matriz é representada por um nome de identificação e pela definição da dimensão de seu tamanho entre os símbolos de chavetas, tendo a seguinte sintaxe:

```
tabela = {}
```

O identificador tabela deverá ser o nome da variável indexada a se fazer uso, e os símbolos { } são os construtores que criam e definem uma variável indexada.

Para se fazer uso de matrizes na linguagem de programação Lua é necessário declarar a tabela no início do código do programa antes de usá-la. Note que não é necessário indicar a quantidade de elementos (valores) que serão armazenados na tabela, pois a própria linguagem efectua dinamicamente o ajuste da tabela à medida em que ela é utilizada.

A título de ilustração, considere um programa que efectue a leitura de dez elementos de uma tabela denominada A. Em seguida, construa uma tabela denominada B, em que a tabela B seja formada de acordo com a seguinte lei de formação: se o valor do índice da tabela for par, o valor deve ser multiplicado por 5; sendo o índice da tabela ímpar, deve ser somado com 5. No final, mostrar os conteúdos das tabelas A e B. Este exemplo demonstra como fazer o tratamento da condição do índice.

```
-- inicio programa TABELA 1

  A = {}
  B = {}
  io.write("Indice de Tabela")
  print()

  for I = 1, 10, 1 do
    io.write("Informe o ",
string.format("%2d", I), "o. valor: ")
    A[I] = io.read("*number")
  end

  for I = 1, 10, 1 do
    R = I - 2 * math.floor(I / 2)
    if (R == 0) then
      B[I] = A[I] * 5
    else
      B[I] = A[I] + 5
    end
  end

  print()
  for I = 1, 10, 1 do
    io.write("A[",
string.format("%2d", I), "] = ")
    io.write(string.format("%4d",
A[I]), " - ")
    io.write("B[",
string.format("%2d", I), "] = ")
    io.write(string.format("%4d",
B[I]), "\n")
  end

-- fim programa TABELA 1
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabelao1.lua e execute-o com a linha de comando lua 5.1 tabelao1.lua.

No programa são utilizados três laços de repetição for: o primeiro laço controla a entrada dos dados, o segundo verifica se cada elemento da matriz A é par ou ímpar e faz as operações, implicando os elementos calculados na matriz B, e o terceiro laço é utilizado para apresentar as duas matrizes.

A definição das matrizes A e B foram feitas com o estilo mais simples de definição, sendo: A = {} e B = {}.

No laço de processamento, é utilizada a instrução if (R == 0) then, sendo a variável R carregada com o resultado da equação I - 2 * math.floor(I / 2) que possibilita extrair o resultado do resto da divisão de inteiros do valor I sobre o valor 2. Qualquer valor dividido por 2 que resultar zero é par; se o resto for diferente de zero, o valor é ímpar.

Na sequência será desenvolvido um programa que efectua a leitura de cinco elementos de uma tabela A. No final, apresenta o total da soma de todos os elementos que sejam ímpares. Em relação ao primeiro exemplo este apresenta uma diferença: o primeiro pedia para verificar se o índice era par ou ímpar. Neste exemplo, é solicitado que se analise a condição do elemento e não do índice. Já foi alertado anteriormente para tomar cuidado e não confundir elemento com índice. Veja o programa.

```
-- inicio programa TABELA 2

  A = {}
  io.write("Somatorio de elementos
impares\n\n")


  for I = 1, 5, 1 do
    io.write("Informe o ",
string.format("%2d", I), "o. valor: ")
    A[I] = io.read("*number")
  end

  SOMA = 0
  for I = 1, 5, 1 do
    R = A[I] - 2 * math.floor(A[I] /
2)
    if (R ~= 0) then
      SOMA = SOMA + A[I]
    end
  end

  print()
  io.write("A soma dos elementos
impares equivale a: ")
  io.write(string.format("%4d", SOMA),
"\n")

-- fim do programa TABELA 2
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabela02.lua e execute-o com a linha de comando lua 5.1 tabela02.lua.

No laço de processamento, é utilizada a instrução if (R ~= 0) then, para verificar se o elemento informado pelo teclado é um valor ímpar; sendo ímpar, ele é acumulado na variável SOMA, que no final apresenta o somatório de todos os elementos ímpares digitados durante a execução do programa.

Matrizes Bidimensionais

Com o conhecimento adquirido é possível criar um programa que efectua a leitura das notas escolares de oito alunos, o cálculo da média de cada aluno e no final, apresenta a média da classe, utilizando apenas tabelas unidimensionais. Porém, dá algum trabalho, uma vez que é necessário manter um controlo de cada índice em cada tabela para um mesmo aluno.

Para facilitar o trabalho com estruturas deste porte, pode-se utilizar tabelas com mais de uma dimensão. A forma mais comum a ser usada é a tabela de duas dimensões.

Tabelas com mais de duas dimensões são de uso menos frequente, mas são fáceis de se usar quando se conhece bem utilização de tabelas com duas dimensões.

Um aspecto a ser considerado no uso de tabela unidimensional é o facto de estar em uso apenas uma única instrução de laço de repetição. Em relação às tabelas com mais dimensões, deve ser utilizado o número de laços de repetição relativos ao tamanho de sua dimensão. Desta forma, uma tabela de duas dimensões deve ser controlada com dois laços de repetição, uma de três dimensões deve ser controlada por três laços de repetição e assim por diante

Uma tabela de duas dimensões é aquela que possui um determinado número de colunas com um determinado número de linhas. Por exemplo, uma tabela com 5 linhas e 3 colunas, uma tabela de 5x3.

A definição de uma matriz de duas dimensões na linguagem de programação Lua deve ser feita seguindo a regra de definição seguinte:

```
-- inicio programa TABELA 3

  A = {}
  io.write("Leitura e apresentacao de
notas\n\n")

  for I = 1, 8, 1 do
    A[I] = {}
    print()
    io.write("Entre notas do ",
string.format("%2d", I), "o. aluno: ")
    print()
    for J = 1, 4, 1 do
      io.write("Nota ",
string.format("%2d", J), ": ")
      A[I][J] = io.read("*number")
    end
  end

  print()
  for I = 1, 8, 1 do
    print()
    io.write("As notas do aluno ",
string.format("%2d", I), " sao: ")
    for J = 1, 4, 1 do
      io.write(string.format("%2d",
J), ": ")
      io.write(string.format("%5.2f",
A[I][J]), " ")
    end
  end
  print()

-- fim programa TABELA 3
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabela03.lua e execute-o com a linha de comando lua 5.1 tabela03.lua.

Tabelas com Listas

O uso de variáveis indexadas como listas de valores em linguagem Lua pode ser feita com a sintaxe:

```
DIA_SEMANA = {
  "domingo",
  "segunda-feira",
  "terca-feira",
  "quarta-feira",
  "quinta-feira",
  "sexta-feira",
  "sabado"
}
```

A variável DIA_SEMANA está formada com sete índices. Se for pedido o uso da posição 1 como DIA_SEMANA[1] será obtido o valor domingo, como pode ser indicado no código de programa seguinte:

```
-- inicio programa TABELA 4

  DIA_SEMANA = {
    "domingo",
    "segunda-feira",
    "terca-feira",
    "quarta-feira",
    "quinta-feira",
    "sexta-feira",
    "sabado"
  }

  for I = 1, 7, 1 do
    print(DIA_SEMANA[I])
  end

-- fim programa TABELA 4
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabela04.lua e execute-o com a linha de comando lua 5.1 tabela04.lua.

Note ao executar o programa que o primeiro índice da tabela é o índice 1 e não o, como ocorre em algumas outras linguagens de programação. No entanto, se quiser definir o primeiro índice como zero, faça o seguinte:

```
-- inicio programa TABELA 5

  DIA_SEMANA = {
    [0] = "domingo",
    "segunda-feira",
    "terca-feira",
    "quarta-feira",
    "quinta-feira",
    "sexta-feira",
    "sabado"
  }

  for I = 0, 6, 1 do
    print(DIA_SEMANA[I])
  end

-- fim programa TABELA 5
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabela05.lua e execute-o com a linha de comando lua 5.1 tabela05.lua.

Note que nesta versão o dia da semana domingo passa a estar posicionado no índice zero da variável indexada, ou seja posicionado em DIA_SEMANA[0].

O programa tabela05.lua mostra a definição explicita do índice de uma variável indexada. Este procedimento pode ser usado para todos os valores de uma lista. Por exemplo:

```
ALIMENTO = {
  [1] = "cenoura",
  [2] = "batata",
  [3] = "beterraba",
  [4] = "arroz",
  [5] = "bacalhau"
}
```

Outra possibilidade de operação com listas em variáveis indexadas é a possibilidade de se usar como elemento da variável funções matemáticas.

O programa a seguir apresenta o resultado de algumas raízes quadradas. Observe o uso da função math.sqrt() como elemento da variável indexada RAIZES.

```
-- inicio programa TABELA 6

  RAIZES = {
    math.sqrt(16),
    math.sqrt(25),
    math.sqrt(36),
    math.sqrt(49)
  }
```

```
  for I = 1, 4 do
    print(RAIZES[I])
  end


-- fim programa TABELA 6
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabelao6.lua e execute-o com a linha de comando lua 5.1 tabelao6.lua.

Note a linha de código for I = 1, 4 do, veja que nesta versão está se omitindo o valor de incremento do contador. Quando isso é feito a linguagem Lua assume por padrão o passo de contagem 1.

Registros

Uma variável indexada do tipo registro é normalmente uma estrutura de dados que permite o uso de tipos de dados diferentes na mesma variável, por meio da definição de campos de dados.

A linguagem Lua não é rígida neste sentido, pois trata automaticamente o tipo do dado em uso. No entanto, é possível definir dados em forma de registros, de forma semelhante a outras linguagens.

O programa seguinte faz a entrada e saída do nome e de quatro notas escolares de um aluno. Observe atentamente o detalhe na definição da variável indexada ALUNO com os campos NOME e NOTAS.

```
-- inicio programa TABELA 7


  ALUNO = {
    NOME,
    NOTAS = {}
  }

  print("Cadastro de Aluno")
  print()


  io.write("Entre o nome .....: ")
  ALUNO.NOME = io.read()

  for I = 1, 4 do
   io.write(I .. "a. nota .: ")
   ALUNO.NOTAS[I] = io.read("*number")
  end

  print()
  print("Nome ...: " .. ALUNO.NOME)
  for I = 1, 4 do
    print("Nota " .. I .. " .: " ..
ALUNO.NOTAS[I])
  end

-- fim programa TABELA 7
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabelao7.lua e execute-o com a linha de comando lua 5.1 tabelao7.lua.

Variáveis indexadas do tipo registro podem ser acedidas pela identificação do nome do campo, como VARIÁVEL.CAMPO, bem como VARIÁVEL["CAMPO"]. Note o programa seguinte:

```
-- inicio programa TABELA 8

  L = {C1 = 100, C2 = "Lua"}
  print(L["C1"]) -- escreve 100
  print(L["C2"]) -- escreve Lua
  print(L.C1)    -- escreve 100
  print(L.C2)    -- escreve Lua
  L.C1 = nil     -- remove campo
  print(L.C1)   -- escreve "nil"
  L.C1 = 500     -- insere campo
  print(L.C1)    -- escreve 500

-- fim programa TABELA 8
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabelao8.lua e execute-o com a linha de comando lua 5.1 tabelao8.lua.

Observe que o programa tabelao8.lua faz uso de alguns elementos operacionais sobre variáveis indexadas. Note que este tipo de acção possibilita o uso de diversos recursos para manipulação de varáveis indexadas para operações com registros.

A linguagem Lua no que diz respeito à utilização de variáveis indexadas de registros permite outros graus de requinte, como o código seguinte:

```
-- inicio programa TABELA 9

  L = {C1 = 2010, C2 = "Linguagem "}
  L[1] = "Lua"

  R = {math.sqrt(81), math.sqrt(49)}
  R.TXT = L


  print(R[2])
  print(R.TXT[1])
```

```
  print(L.C2 .. R.TXT[1])

-- fim programa TABELA 9
```

Em seguida escreva o código do programa num editor de texto, gravando-o com o nome tabelao8.lua e execute-o com a linha de comando lua 5.1 tabelao8.lua.

O primeiro print escreve o valor 7 referente à raiz quadrada de 81, o segundo print escreve a palavra Lua e o terceiro print escreve a frase concatenada Linguagem Lua.

Perceba que a definição da variável R.TXT assume o valor da variável L[1].

Conclusão

Neste artigo foi dado atenção às acções de processamento baseadas no uso de concatenações, precedências de operadores e principalmente na utilização de forma básica de variáveis indexadas.

No próximo artigo serão tratadas as questões relacionadas ao uso de funções e suas definições..

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem experiência em ensino e desenvolvimento de programação de software desde 1986. É professor da rede federal de ensino no Brasil, no Instituto Federal de Educação, Ciência e Tecnologia. É também autor, possuindo na sua carreira várias obras publicadas na área da computação.

augusto.manzano@portugal-a-programar.org

Augusto Manzano

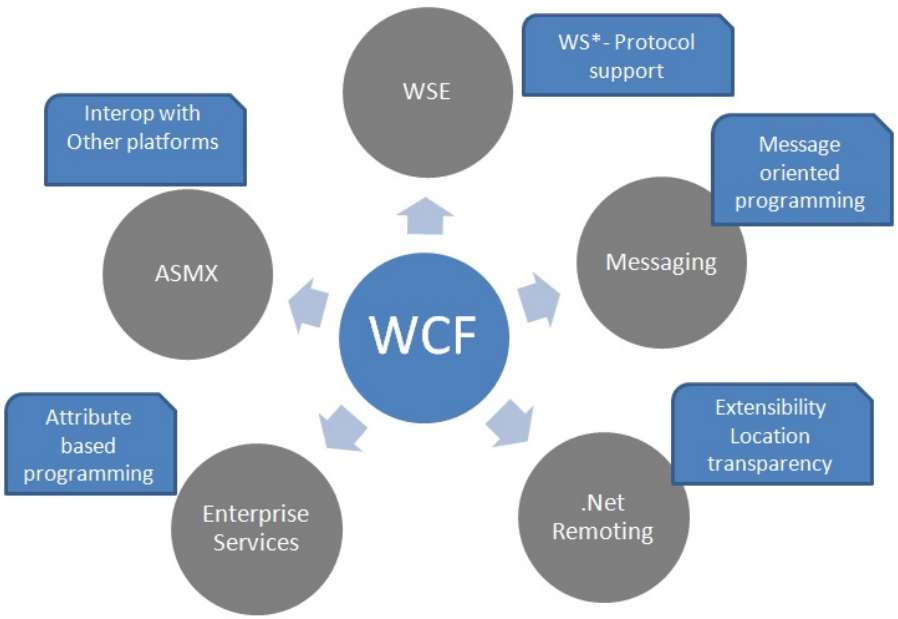
Introdução ao Windows Communication Foundation

Introdução

Com o crescimento da .NET framework, a tarefa de criar aplicações distribuídas ficou menos penosa. Seja pela facilidade de usar Web Services, pela performance e flexibilidade do .NET Remoting ou pela robustez do Enterprise Service (COM+), não esquecendo o MSMQ.

A Microsoft quando criou a .Net Framework 3.0, uma das novidades adicionadas foi o Windows Communication Foundation (WCF), que uniu as várias tecnologias de programação distribuídas na plataforma Microsoft, como por exemplo, Web Services Enhancements (WSE), ASP.NET Web Services, .NET Remoting, COM+ (Enterprise Services) e Message Queue (MSMQ), num único modelo, baseando-se na arquitetura orientada a serviços (SOA).

Com a chegada de uma nova versão da .NET Framework chega também uma nova versão do WCF, versão 4.



Nesta versão existem muitas novidades, mas neste artigo só vão ser abordadas quatro dessas novidades, como forma de estimular a curiosidade dos leitores, fornecendo assim uma plataforma de início de aprendizagem.

Todos os exemplos apresentados neste artigo são escritos usando C#.

O que é o WCF

O WCF é uma framework que nos possibilita construir e usar aplicações orientadas a serviços. Como principais características temos:

- modelo de programação unificado;
- suporte a SOA;
- interoperabilidade e fiabilidade baseada em padrões de mercado;
- segurança integrada;
- arquitetura flexível e extensível;
- maior flexibilidade e segurança em relação aos serviços ASP.NET Web Services.

Como principal desvantagem temos:

- Fazer o design certo para os requesitos por vezes torna-se um pouco dificil.

Serviços em WCF

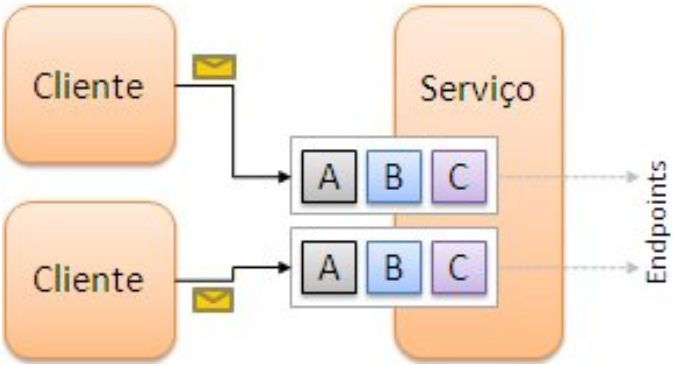
A estrutura de um serviço WCF não é muito complexa. Deve-se utilizar conceitos puros de programação .NET para a criação do contrato e da classe que representará o serviço. Além disso, o WCF também suporta a utilização de tipos complexos, como classes criadas para atender uma determinada necessidade.

Para definir um serviço em WCF pode-se usar a definição da Microsoft "Toda comunicação com um serviço WCF ocorre através de endpoints do serviço. Os endpoints fornecem aos clientes o acesso às funcionalidades oferecidas por um serviço WCF."

Lendo atentamente a definição, verifica-se que para usar serviços WCF é necessário estes serem expostos via endpoints.

Cada serviço precisa de um endereço (Address) para que possa ser utilizado. Cada serviço possui também um contrato (Contract) que define o que o serviço vai fazer e

uma vinculação (Binding) que define como se comunicar.



Em WCF o relacionamento entre o endereço (Address), o contrato (Contract) e a vinculação (Binding) é chamado de endpoint.

Address + Binding + Contract = Endpoint (o famoso ABC do WFC)

Hosting

Uma das grandes vantagens do WCF é a possibilidade de utilizar qualquer tipo de aplicação como host, não existem dependências de software, como o IIS (Internet Information Services), como acontece com os ASP.NET Web Services. O WCF pode expor serviços para serem acedidos através dos mais diversos tipos de protocolos, como por exemplo: HTTP, TCP, IPC e MSMQ.

Atualmente existem três alternativas de hosting: self-hosting, IIS e WPAS. Como há vários detalhes na criação e gestão do hosting, fica muito extenso publicar cada detalhe, vantagens e desvantagens que cada uma das técnicas possui. Na bibliografia, o leitor pode encontrar referências ao hosting.
O que há de novo no WCF 4

Tal como foi dito inicialmente, vão ser abordadas quatro das muitas novidades que podemos encontrar na versão 4 do WCF.

A primeira dessas novidades é a configuração simplificada.

Ao analisarmos o ficheiro de configuração usado da versão 3.5, verifica-se que por vezes o ficheiro de configuração consegue ser mais complicado que todo código que se escreve para criar e utilizar os serviços em WCF.

Exemplo de um ficheiro de configuração do WCF 3.x

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior
name="CalculatorServiceBehavior">
          <serviceMetadata
httpGetEnabled="True"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service
name="Microsoft.Samples.GettingStarted.C
alculatorService"

behaviorConfiguration="CalculatorService
Behavior">
        <endpoint address=""
binding="basicHttpBinding"
contract="ICalculator"/>
        <endpoint address="mex"
binding="mexHttpBinding"
contract="IMetadataExchange"/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Na versão 4, do WCF o ficheiro de configuração é simplificado. Não existe a necessidade de uma configuração muito extensa. Pode-se tirar vantagem de algumas das novas funcionalidades introduzidas pela Microsoft:

- Configuração automática de bindings e behaviours;
- Default endpoints;
- Não é necessario usar a tag <service>.

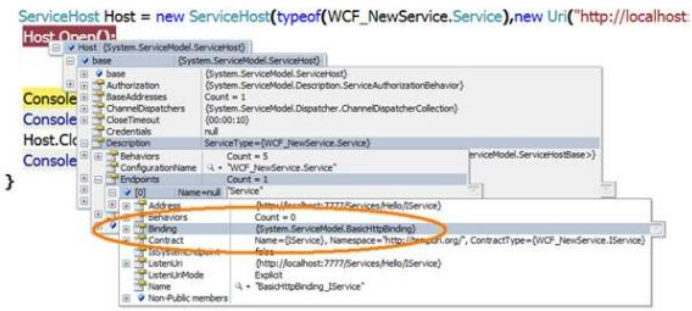
Exemplo de um ficheiro de configuração do WCF 4

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="">
          <serviceMetadata
httpGetEnabled="True"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Self hosting de um serviço WCF 4 com configuração simplificada

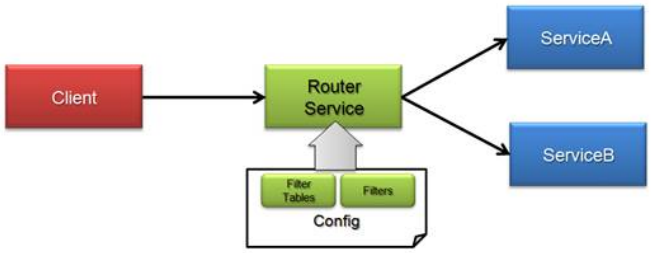
```
ServiceHost serviceHost = new
ServiceHost(typeof(HelloService),
    new
Uri("http://localhost:7777/Services/Hell
o"),
    new
Uri("net.tcp://localhost:7778/Services/H
ello"));
serviceHost.Open();

Console.WriteLine("WCF Service is
running.");
Console.WriteLine("Press <ENTER> to
terminate service.");
Console.ReadLine();

serviceHost.Close();
```

Como o leitor pode ver na imagem abaixo o WCF cria automaticamente toda a configuração sem existir necessidade de existir um ficheiro de configuração.



No WCF 4 existe a possibilidade de se definirem serviços de routing.

O uso desta funcionalidade permite que as aplicações cliente só tenham a necessidade de conhecer a existência de um único endpoint (os serviços de routing). Com base em tabelas de routing, pode-se instruir o WCF para reencaminhar os pedidos para os serviços correctos.



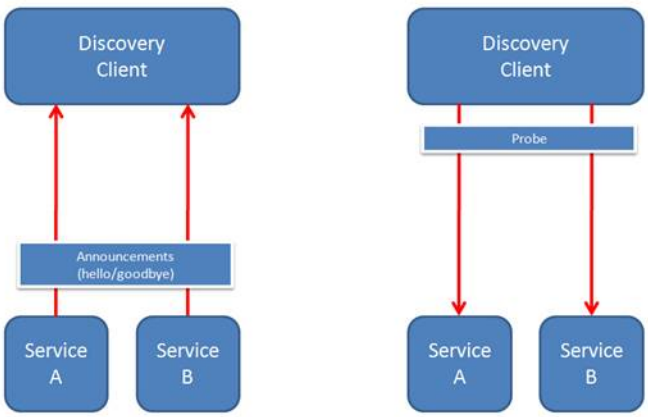
Exemplo de uma routing table em WCF

```
<client>
      <endpoint name="HelloService"
       address="http://..."
       binding="basicHttpBinding"
contract="*" />
</client>
<routing>
      <filters>
        <filter name="MatchAllFilter"
filterType="MatchAll" />
      </filters>
      <filterTables>
        <filterTable
name="mainRoutingTable">
          <add
filterName="MatchAllFilter"

endpointName="HelloService" />
    </filterTable>
      </routingTables>
</routing>
```

O WCF 4 implementa a funcionalidade de discovery.

Esta funcionalidade pode ser usada de duas maneiras diferentes:

- Ad-Hoc;
- Probing (Hello/Goodbye).



O discovery funciona pois os endpoint implementam o protocolo UDP. Com o WCF 4 é bastante fácil implementar esta funcionalidade, para tal basta que:

•Os serviços que queiram usar esta funcionalidade implementarem um discovery endpoint e permitirem o uso desta funcionalidade na sua configuração;

•Os serviços que desejam enviar "heartbeats" permitam o uso desta funcionalidade na sua configuração;

•Os clientes que queiram fazer "prob" dos serviços implementem a funcionalidade de discovery.

Exemplo de ficheiro de configuração do uso Ad-Hoc do discovery

```
<system.serviceModel>
      <services>
        <service name="HelloService"

behaviorConfiguration="serviceBehavior">

          <host>
            <baseAddresses>
              <add
baseAddress="http://localhost:7777/Servi
ces/Hello"/>
            </baseAddresses>
          </host>
          <endpoint
binding="basicHttpBinding"

contract="IHelloService" />
          <endpoint name="udpDiscovery"
kind="udpDiscoveryEndpoint"/>
        </service>
      </services>
      <behaviors>
        <serviceBehaviors>
          <behavior
name="serviceBehavior">
            <serviceDiscovery />
          </behavior>
        </serviceBehaviors>
      </behaviors>
</system.serviceModel>
```

Exemplo de ficheiro de configuração do uso Probing (Hello/Goodbye) do discovery

```
<behaviors>
    <serviceBehaviors>
       <behavior
name="serviceBehavior">
          <serviceDiscovery>
            <announcementEndpoints>
              <endpoint
name="udpAnnouncement"

kind="udpAnnouncementEndpoint"/>
            </announcementEndpoints>
          </serviceDiscovery>
       </behavior>
    </serviceBehaviors>
</behaviors>
```

Exemplo do código de cliente do uso Probing (Hello/Goodbye) do discovery

```
DiscoveryClient discoveryClient = new
DiscoveryClient("udpDiscoveryEndpoint");

FindCriteria findCriteria = new
FindCriteria(typeof(IHelloService));
FindResponse findResponse =
discoveryClient.Find(findCriteria);

if (findResponse.Endpoints.Count > 0)
{
    EndpointAddress address =
findResponse.Endpoints[0].Address;


ChannelFactory<IHelloServiceChannel>
factory =
      new
ChannelFactory<IHelloServiceChannel>(
        new BasicHttpBinding(),
address);
    IHelloServiceChannel client = new
factory.CreateChannel();

    client.SayIt("Hello from WCF4!");

    client.Close();
    factory.Close();
}
```

Conclusão

De certa forma, pode-se dizer que o WCF é até agora, o caminho certo, pois facilita o desenvolvimento e é totalmente desacoplado das regras de negócio que serão expostas pelos serviços.

A Microsoft quis com o WCF convergir todas as tecnologias de comunicação existentes numa única tecnologia que permitisse aos programadores terem as mesmas funcionalidades de antes, sem se preocuparem com as especificidades de cada uma das diferentes tecnologias. Como o leitor pode constatar, o WCF 4 ainda continua a ser WCF pois o ABC do WCF ainda está lá.

A funcionalidade de Discovery é interessante, especialmente para cenários empresariais. O Routing pode ser usado em arquiteturas mais complexas e como forma de abstrair do utilizador final alguma complexidade de configuração.

Não existem grandes mudanças, mas foi efetuado um grande investimento a nível da framework de suport ao WCF para permitir aos utilizadores uma maior versatilidade e um

desenvolvimento mais rápido e eficaz.

Bibliografia

O leitor pode encontrar mais informações em:

• Simplified configuration
http://blogs.thinktecture.com/cweyer/archive/2009/05/07/415326.aspx

• .svc-less Activation
http://blogs.thinktecture.com/cweyer/archive/2009/05/08/415327.aspx

• Dynamic service and endpoint discovery
http://blogs.thinktecture.com/cweyer/archive/2009/05/08/415329.aspx

• Standard endpoints
http://blogs.thinktecture.com/cweyer/archive/2009/05/08/415330.aspx

• Discovery announcements
http://blogs.thinktecture.com/cweyer/archive/2009/05/08/415332.aspx

• Routing Service
http://blogs.thinktecture.com/cweyer/archive/2009/05/08/415335.aspx

• Protocol bridging & fault tolerance with the Routing Service
http://blogs.thinktecture.com/cweyer/archive/2009/05/08/415341.aspx

• WCF Hosting
http://www.linhadecodigo.com.br/Artigo.aspx?id=1269

• Fudamentals WCF concepts
http://msdn.microsoft.com/en-us/library/ms731079.aspx

• Using Data Contracts
http://msdn.microsoft.com/en-us/library/ms733127.aspx

• Using Message Contracts
http://msdn.microsoft.com/en-us/library/ms730255.aspx

SOBRE O AUTOR

Residente em Fafe, João Brandão tem 11 anos de experiência em desenvolvimento de software e em montagem e elaboração de redes de computadores. Ocupa desde Fevereiro de 2009 o cargo de Senior Engineer na Critical Manufacturing SA empresa do grupo Critical.
Tem como principais actividades e responsabilidades o desenvolvimento software e gestão de equipas e de projectos.

João Brandão

Modelo3 – O IRS simplificado

Por vezes há projectos que pela sua simplicidade de uso, funcionalidade e utilidade nos despertam a atenção. Foi o caso do Modelo3, um projecto com poucos meses, da autoria de Celso Pinto, cujo objectivo é simplificar o preenchimento do modelo 3 do IRS.

Foi por isso que a Revista Programar entrevistou Celso Pinto a quem agradecemos a disponibilidade:

Revista Programar (RP) - Em breves palavras como descreve o Modelo3 para quem não conhece?

Celso Pinto (CP) - O Modelo3 é um serviço alternativo às aplicações das Finanças para preenchimento do IRS, mas bastante mais simples. A ideia principal é permitir que eu pegue no molho de papéis que tenho para incluir na declaração e que os passe um por um para o serviço que depois logo trata de gerar um ficheiro. Durante o preenchimento é feita uma simulação de IRS e também vou descobrindo que limites de deduções já atingi.

RP - Como surgiu a ideia de implementar o modelo3.pt?

CP - A ideia já tem uns anos, 2006 ou 2007, já não me lembro. Acho que estava de roda do preenchimento de uma declaração de IRS e fartei-me dos erros de validação porque na aplicação das Finanças é preciso declarar várias vezes o mesmo valor em sítios diferentes, para além de que essa aplicação faz muitos poucos cálculos, e comecei a pensar em fazer uma aplicação para consumo próprio que fosse ao encontro das minhas necessidades.

Isto foi ficando na gaveta, até há uns tempos quando propus na 7syntax que o serviço fosse desenvolvido como brincadeira no Codebits do SAPO, em 2008, mas o Rock Band falou mais alto :-) Entretanto a meio do ano passado quis começar a dar vazão a projectos que engavetados há algum tempo e este foi o primeiro.

RP - Que tecnologias utilizou na programação desta ferramenta?

CP -Infelizmente não há no Modelo3 nada de particularmente interessante :-) No server-side está uma Play Framework por detrás de um nginx que serve ficheiros estáticos. A Play foi escolhida porque quis usar o Drools para as simulações e precisava de algo em Java.

Podia ter sido um Tomcat ou outra coisa qualquer, a única coisa que faz é converter JSON para objectos Java e vice-versa, para além de gerar o ficheiro da declaração. No entanto a Play não tem configurações praticamente nenhumas e gostei bastante da aproximação deles ao desenvolvimento sobre a framework. Do lado do cliente, como disse atrás, são tudo ficheiros estáticos com Javascript aqui e ali para animar as páginas e falar com o server-side. Os ficheiros estão exactamente como me chegaram às mãos, o trabalho que a Quodis desenvolveu é fantástico, só tive de importar os ficheiros de Javascript necessários para fazer bind aos elementos certos.

RP - Teve alguma receptividade ou apoio por parte das finanças?

CP - Sim, mostraram alguma curiosidade e disponibilidade. O objectivo principal era integrar o Modelo3 com o Portal das Finanças para que fosse possível submeter a declaração imediatamente a partir do site, mas não foi possível, olhando para trás vejo que entrei em contacto com a DGCI demasiado em cima da hora, estas coisas levam o seu tempo, o que é natural.

RP - E da parte dos utilizadores, como tem sido o feedback?

CP - Fantástico, o melhor feedback que tive foi atingir o objectivo estabelecido para o número de declarações geradas em metade do tempo, antes do início da 2ª Fase de entregas. Lembro-me também que houve quem me dissesse que estava desconfiado porque o preenchimento do IRS tinha sido demasiado fácil :-) Mesmo nos primeiros dias quando houve alguns problemas sempre senti imenso apoio das pessoas, muitas das quais completamente anónimas para mim. Recebi bastantes sugestões e, em trocas de email, surgiram algumas ideias sobre outras funções que podem ser pagas, o que torna sempre tudo mais interessante.

Também recebi um bom número de manifestações de preocupação com a privacidade dos dados, o que é perfeitamente compreensível. No entanto o Modelo3 não tem qualquer base de dados portanto nada fica registado. Tenho de trabalhar e procurar formas de credibilizar o serviço.

RP - Quais as maiores dificuldades para implementar o projecto?

CP - Esta pergunta é fácil: falta de tempo :-) Tirando a surpresa inicial com a mudança de formato do ficheiro que o Portal das Finanças aceita, o desenvolvimento tem corrido sem sobressaltos mas sem a rapidez desejável. Algo a rever em breve.

RP - Que projectos tem para o futuro do Modelo3?

CP - Para já analisar todo o feedback recebido, informações no Analytics e o que foi sendo escrito sobre o Modelo3. Daqui espero retirar conclusões sobre algumas formas de rentabilização do projecto, mas ainda é cedo para falar de medidas específicas.

http://modelo3.pt/

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

Vai a

www.revista-programar.info

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

revistaprogramar @portugal-a-programar.org

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

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org